RobMoSys

H2020-ICT-732410

RobMoSys

Composable Models and Software for Robotics Systems

Deliverable D4.2:

Second Report on Pilot Cases

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement N732410.

Project acronym:	RobMoSys
Project full title:	Composable Models and Software for Robotics Systems
Work Package:	WP 4
Document number:	D4.2
Document title:	Second Report on Pilot Cases
Version:	1.0
Delivery date:	30 June, 2019
Nature:	Report (R)
Dissemination level:	Public (PU)
Editor:	Enea Scioni (KUL), Daniel Meyer-Delius Di Vasto (Siemens)
Authors:	Daniel Meyer-Delius Di Vasto (Siemens), Alfio Minissale (COMAU), Sergi Garcia(PAL), Selma Kchir (CEA), Dennis Stampfer (HSU)
Reviewer:	Maria Teresa Delgado(EFE)

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement N°732410 RobMoSys.

Executive summary

Pilot cases are fundamental indicators of the benefits of the RobMoSys approach, by applying RobMoSys *methodology*, *models* and *software tools* in **concrete industry-relevant cases**. Therefore, Pilot cases are a major mean for dissemination and adoption of the RobMoSys approach, playing a fundamental role in the creation of the RobMoSys community. The presence and the commitment of industrial players in the RobMoSys consortium, such as Siemens, PAL Robotics and COMAU, helps in that regards, as convincing users and developers of advanced robotics systems.

The RobMoSys project is very ambitious regarding the number of **application domains** and **user stories** that the RobMoSys approach addresses, from the development of single software components, up to the creation and the configuration of complex and predictable software systems. To ensure to address to the majority of user cases, both with respect to the RobMoSys roles and with respect to RobMoSys models and meta-models, a **coverage matrix** of the Pilot cases has been compiled. Moreover, *Key Performance Indicator*s (KPI) has been defined to establish the development status and to evaluate the RobMoSys benefits, concretely, within the scope of the specific case and user story.

Interactions with academic partners, which are developers of RobMoSys models and tools (i.e., the RobMoSys software baselines), is strong and active. In this sense, the Pilot cases are main *drivers* of the motion, perception and world model stacks, pushing concrete requirements and priorities on functionalities and the tools, which are necessary for the realisation of each case. In particular, the world model has been subject of several discussions, since it happens to be the most common stack in all Pilots. A result of these interactions is a better design of a composable world model stack; its details are discussed in Deliverable D3.2. In the same vein, Pilot partners have spent a significant efforts as early adopters of the RobMoSys software baselines, from which they started to build *Pilot skeletons*, that is, a refinement of the Pilot cases, already defined in Deliverable D4.1, in terms of models and meta-models to use.

Finally, Pilot cases are sources of inspiration for the **2nd call of Integrated Technical Projects**. In particular, it has been suggested to integrate and to test ITP results in the context of one or more Pilot cases. Therefore, a stronger relation and interaction between ITP projects and Pilot cases owners is expected, and this will help to strengthen the birth of a RobMoSys community.

This Deliverable presents the current progress on the development of each pilot case, handled by industrial and academic partners of the RobMoSys consortium.

Contents

1. The RobMoSys Approach and Pilot Cases: an Overview

Pilots are application-centric systems aimed to demonstrate the use of the proposed model-driven methodology through the development of full applications. Pilots span different domains and different kind of applications (and hence requirements), centered around the core applications of "navigation" and "(mobile) manipulation".

During the project, the core partners create Pilot "skeletons" conforming to the proposed methodology and built out of the basic building blocks developed in the project. These Pilots are examples of different robotic applications and use cases with different focus proposed by the partners according to their own individual background, motivation and requirements.

The project Pilots evolve together with the technical developments of the project and have thus different degrees of maturity. While some are still at a rather early development stage others provide already a wide range of functionality. Some of Pilots follow the RobMoSys approach from the initial design and implementation, while other involve legacy systems that are not RobMoSys conform.

Five different Pilots have been developed during the project and each Pilot considers one or more different use cases that focus on different aspects of the RobMoSys approach. Section 1.1 presents an overview of the focus of the different project Pilots and use cases, and the remainder of this report describes in detail each individual Pilot.

In addition to the description of the application scenario, setup, focus and technical details, each Pilot also proposes several key performance indicators aimed at evaluating then benefits of the RobMoSys approach in its own context. Section 1.2 introduces the approach that is to be followed for the benchmarking of the approach in the context of the Pilots.

The Pilots not only demonstrate and validate the use of the RobMoSys approach, they are also the basis for some of the project's Integrated Technical Projects (ITPs). The Pilots provide specifications of appropriate levels of interfacing conforming to the RobMoSys approach for the ITPs to build upon or extend.

1.1 Pilot Coverage Matrix

The aim of having different pilot scenarios is to show the multiple benefits of the RobMoSys approach, and to ensure that the approach is generic enough to be adopted by different stakeholders operating in different (robotics application) contexts. To have a better overview on the features that each pilot covers, a **pilot coverage matrix** is presented in Tab. 1.1. The coverage matrix highlights which **roles**, **metamodels**, **robotic domains** are used in each pilot, which **tools** are used and up to which level of **composition** the pilot targets to. In particular, RobMoSys features can be found in every pilot scenario, but the aim of the coverage matrix is to indicate the major features (or selling points) which have a major impact on the concrete pilot.

- **Ecosystem Roles** involved in the development of the robotic application, as described in the relative RobMoSys wiki page;
- **Metamodels** that are fundamental for the pilot development. Even if all RobMoSys metamodels are needed for the development of any robotic application, each pilot focuses the

attention on a subset of those, which are necessary for the specific problem addressed by the considered scenario;

- **Robotic Domain Models**, as the concrete models employed for the development of the robotic applications. Those models are various, and it is simply reported a convenient grouping of those. More details will be discussed in each pilot case in the following chapters.

- **Composition**. This denotes the type and level of composability required by the pilot, and it strongly differs from pilot to pilot. For example, some pilot focuses on the building of a fully-fledged application, employing existing RobMoSys compliant software component and focusing on the "Task Composition"; other pilots focus instead on the development of a concrete functionality by means of composition between components or even within a single software component, and dealing with all possible consequences of a different software configuration;

- **Tools and Software Baseline**, as the list of software currently employed (or planned to be used) for the realisation of the pilot. This aids to identify which pilots can be referred as "tutorial" of each single tool developed by the RobMoSys project partners (core-consortium and ITP projects).

To each of the elements described above, the following color code is applied to indicate the focus and the impact in each pilot/user story:

- Green: the user story focuses directly on the considered element (a feature, a role, a model, etc) and it shows its benefits, or the specific functionality/model has even been developed on purpose for the pilot;

- Cyan: the user story uses the indicated element (a feature, a role, a model, etc), but it is not major focus of the user story, nor an enabler of the benefits that the user story aims to cover;

- Yellow: the element (feature, role, model, etc) could be employed for the pilot case, but currently there is not plan or focus on such an element;

- White/empty: the specific element is not considered or non-relevant for the pilot case.

Further details of each pilot and user story is presented in the following Chapters of this document (Ch. 2-Ch. 6). The coverage pilot matrix will be updated during the remaining duration of the RobMoSys project.

1.2 Key Performance Indicators

Robotic systems are complex systems and there is no straight-forward way to characterize them so that two different systems can be compared in a meaningfull way. Comparing the development (process) of robotic systems is even more difficult since it often requires metrics about the development of many different systems for different applications, in different setups and scenarios, and by many different users in different roles.

		Pilot name	Flexible Assembly Cell		Healthcare Assistive Robot		Modular Educational Robot			Human Robot Collaboration			Intralogistic Robot Fleet
		User story Impact	User story #1 Task-oriented programming	User story #2 Hardware component replacement	User story #1 Replacement of components	User story #2 Task coordination	User story #1 Easy application design	User story #2 Easy end-effector design	User story #3 Easy web-integration	User story #1 Safety at Design Time	User story #2 Safety at Runtime	User story #3 Flexibility and resistance to low-level changes	User story #1
			5	5	6	4	13	7	13	11	12	14	19
Ecosystem Roles	Behaviour Developer	6											
	Component Supplier	2											
	Function Developer	3											
	Performance Designer	4											
	Safety Engineer	3											
	Service Designer	6											
	System Architect	4											
	System Builder	6											
Metamodels	Robotic Behaviour	6.5											
	Communication-Object	6											
	Communication-Pattern	5											
	Component-Definition	8											
	Deployment	4.5											
	Functional Architecture	1											
	Cause-Effect-Chain and its Analysis	3											
	Platform	3											
	System Service Architecture and Service Fulfillment	2											
	Service-Definition	4											
	System Component Architecture	3											
Robotic Domain Models	Motion	1											
	Perception	4.5											
	World Models	5.5											
	Flexible Navigation	2.5											
	Active Object Recognition	2.5											
	Digital Data Representation	3.5											
Composition	Task Composition	6											
	Service-based Composition	5											
	Composition of algorithms	2											
	Managing Cause-Effect Chains in Component Composition	1											
	Coordinating Activities and Life Cycle of Software Components	0											
Tools and Software Baseline	SmartSoft World	5											
	Papyrus for Robotics	3											
	Groot	1											
	BehaviorTree.CPP	1											

Table 1.1: Pilot Coverage Matrix

In an effort to assess the benefits of the RobMoSys approach in the context of the consortium Pilots, we follow the "Goal-Question-Metric (GQM)" approach[1]. This approach has been widely used for product and process assesment, including improvement assessment. It has also been used as evaluation framework in the ECSEL JU and EC funded project AMASS[2] as well as in the first open call integrated technical project eITUS[3].
In the GQM approach, a set of goals is first defined. Then, for each goal, a set of questions is specified to asses the achievment of the goal. To answer these questions, a set of metrics are defined. Once the goals, questions and metrics have been specified, a benchmarking plan can be elavorated assesing the overall benefit of the approach being evaluated.
In the following sections, the first version of the goals, questions, metrics and benchmarking plan for each individual Pilot are presented. During the next pahse of the realization of the Pilots, the benchmarking plans will be individually reviewed, refined and ultimately executed in synchronization with the benchmarking plans of all other Pilots, technical progress of the project and results of the ITPs.

[1] R. van Solingen, E. Berghout: The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development, McGraw-Hill (1999)
[2] https://www.amass-ecsel.eu
[3] https://robmosys.eu/e-itus

2. Flexible Assembly Cell (Siemens)

2.1 Progress Summary

The efforts during M12-M36 were mostly focused on the development of the base functionality and a minimal application. This setup allows us to realize the "flexible assembly cell" scenarios addressed in this Pilot, showcasing the benefits of the RobMoSys approach. Specifically, these scenarios aim to show how task-level coordination and composition is employed in an industrial use case. The main efforts allow us to progress in the definition and the selection of the models employed in the Pilot case, the development of the Pilot scenario, and exposure of basic (legacy) functionalities implemented in proprietary, industrial environment.

.

2.2 Pilot Scenario and Use Cases

Modern automation devices such as advanced robotic arms, perception systems and high-end industrial PCs do not rely on simple I/O signals for communications as their predecessors did, but they provide full-fledged, high-level application programming interfaces to access the device's features and functionality. This is also due to the fact that modern automation devices are now complex systems by itself, composed by a multitude of subsystems for motion and sensing. Therefore, developing modern automation systems means developing individually more complex devices and functionality that can be utilized in a wide range of different application scenarios.
Complex devices and functionality must be combined with other existing devices, providing an overall set of functionalties to perform a given task in a flexible way. Those complex "systems of systems" must be reliable, not only from the functionality point of view, but also regarding non-functional features of the overall solution: real-time capabilities, latencies, semantic compatibility of the shared data, traceability of each step in the industrial process just to mention a few.
This Pilot focuses specifically on highly-flexible industrial automation. It showcases the development and programming of advanced automation systems that can perform a large range of different tasks with high demands on performance an adaptability.

The Pilot Scenario

The stage of this Pilot is the ongoing industrial revolution largely driven by the increasing demand for flexible automation. This revolution is characterized by constantly increasing numbers of product variants, constantly decreasing product life cycles and constantly decreasing lot sizes. End-to-end automation using classic approaches is not always feasible in this context, leading to low degrees of automation in many phases of production. One of the phases still executed mostly manually is discrete manufacturing where automation, using classical approaches, is not cost-effective for large number of product variants due to the associated high engineering costs. One of the target systems of this Pilot is a flexible assembly cell for manufacturing different complex components. The cell has a high degree of autonomy and does not rely on special-purpose tools or sensors.
Flexible intralogistics and material transport is another fundamental requirement for a flexible production. Classical solutions such as conveyor systems cannot fullfill the increasing demand for flexibility. Constant reconfigurations of the shopfloor and material flow are becomming increasingly

common. The second target hardware in this Pilot is a mobile manipulator system for flexible material transport. The system consists of a an advanced robotic arm for material handling ontop of a mobile platform. The system can use its sensors for navigation and does not requred fixed tracks on the floor for navigating - collision free - from one location to another.
One final but important aspect of our Pilot scenario is the interaction with existing "legacy" system. A large number of automation scenarios extend or closely interact with existing systems. In this so-called "brownfield" scenatios, new systems (and approaches) must take into account, coexist and very frequently even rely on systems that are already in operation and cannot be easily modified. For Siemens, as the leading European provider of factory automation products, systems and solutions, the seamless interaction with legacy system is one of the most relevant aspects of system development.

The use cases

Orthogonally, two use cases are considered in both scenarios: task programming and the problem of replacing a hardware or software component. In the first use case, the "user" plays the role of the *Behavior Developer*, who specifies different assembly tasks using reusable and composable task blocks, without knowing the details of the software and hardware components that will be ultimately used for realizing the task. That is, this first use case focuses on *Task-Level Coordination and Composition*.
In the second use case, the "user" plays the role of the *System Builder*, who replaces a software component. For example, the possible causes of the replacement of a software component are due to a replacement in the equipped hardware, or a different implementation choice of the same functionality. After the replacement of a software component in the system, there is the need to check whether both functional and non-functional requirements are still met: this is addressed in this second use case.

2.3 Setup Description

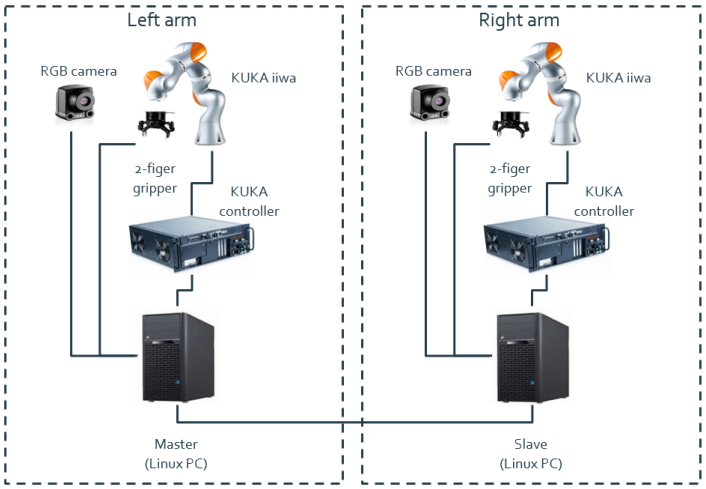

Figure 2.1: Hardware components of the flexible assembly cell setup: two advanced robotic arms, each equipped with a 2D camera for perception and a gripper for object manipulation.

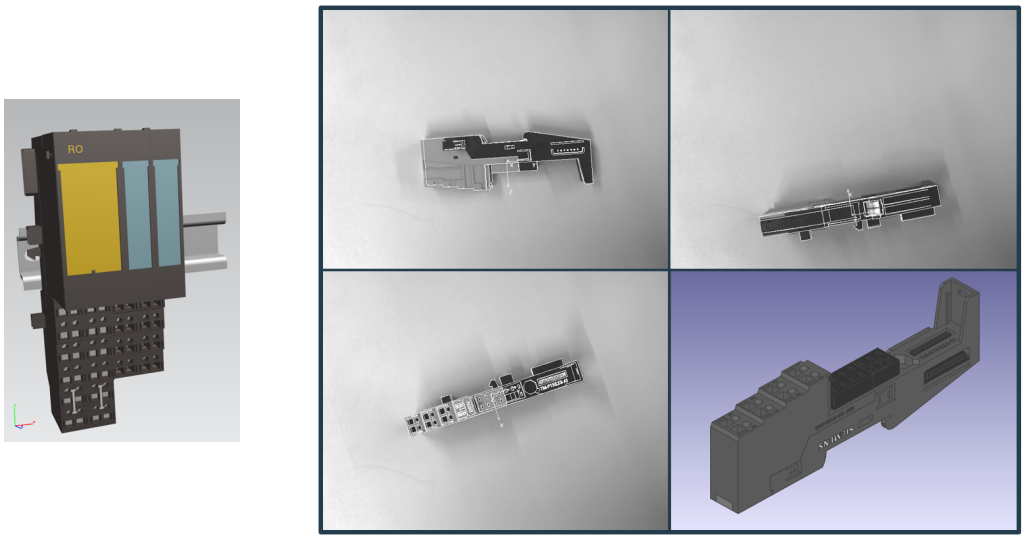

Figure 2.2: Exemplary flexible assembly cell task: mounting top-hat rail modules on a control cabinet. The left image depicts an example control cabinet with three different modules of two different types. The image group on the right shows the results of the object detection functionality.

The setup for this Pilot consists of two systems: (i) a flexible assembly cell and (ii) an advanced mobile manipulator. Figure 2.1 depicts the hardware setup of the flexible assembly cell system used in this Pilot which consists of two advanced robotic arms, each equipped with a 2D camera for perception and a gripper for object manipulation. The cell is also equipped with a 3D camera for monitoring the work space. The intended application of this cell is a discrete assembly. Concretely, we consider the task of mounting top-hat rail modules on a control cabinet such as the one depicted in Figure 2.2. The task consists in clipping the modules on the rail. The type and number of modules, as well as their positions on the rail is part of the task specification. The modules are initially located in storage positions and the rough initial position of the top-hat rail is also known in advanced. The task consists in picking the right modules in the right order and clipping them on the rail. The perception system is used for accurately determining the poses of the modules and the rail. The right-hand side of Figure 2.2 shows the results of the model-based object detection component currently in use.
The hardware setup of the machine tending system is depicted in Figure 2.3. It consists of a mobile manipulator including a mobile based with an advanced robotic arm equipped with a 3D camera for perception and a gripper for object manipulation. A CNC milling machine is also part of the setup. The intended application of this mobile manipulator is a machine tending task where the mobile manipulator has to (1) fetch a work piece from their storage location (2) feed the machine with the work piece (3) retrieve the processed work piece from the machine (4) and bring it to a possibly different storage location. . Just as in the flexible assembly application, the initial location of the work piece is only roughly specified. In this setup, the 3D camera is used for accurately determining its pose. Furthermore, a navigation system is used to move the mobile base from one location to the other using laser and odometry data for localization. To communicate with the milling machine OPC UA is used basic interaction with the CNC control unit, for example, for running the milling program.
For realizing both the machine tending and the flexible assembly applications, most of the basic

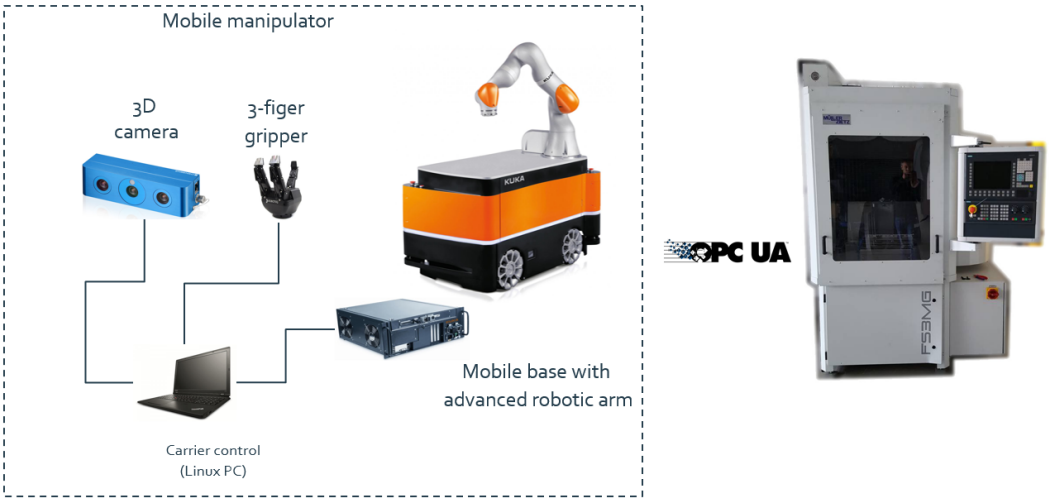

Figure 2.3: Hardware components of the machine tending cell setup: mobile based with an advanced robotic arm equipped with a 3D camera for perception and a gripper for object manipulation.

Figure 2.4: Machine tending task. The mobile manipulator has to feed the CNC milling machine and run the appropriate program. Afterwards, the processed work piece is retrieved from the machine and transported to a storage location

functionalities are realized directly in the programming environment provided by the hardware manufacturers. Those functionalities are then exposed as components of the software system as described in Section 2.5. This choice has been made to show how the RobMoSys approach can deal with industrial, off the shelf hardware, and related proprietary/legacy software. Furthermore, third-party libraries (some of which are proprietary) are also used to implement functionalities such as object detection, motion planning, object manipulation and mobile robot navigation.

2.4 Pilot Focus and Coverage of RobMoSys Features

The main focus of this Pilot is to show composition of software and hardware components for industrial production, in particular, composition at the task level. The first use case concerns task programming. In this context, the main RobMoSys Ecosystem Role is the one of the *Behavior*

Developer. The Behavior Developer can specify or program a task, such as assembly (or machine tending) using the functionalities provided by an existing component-based architecture and a selection of existing software components that realizes it. These functionalities are accessible through *Skill Interfaces*. Skills are, in fact, the building blocks available to the *Behavior Developer* to program the robotic application. Concretely, skills abstract a particular subset of software components in the component-based architecture (and its configuration) that realize a specified functionality. This abstraction allows us to separate the concrete component-based implementation (and the relative choices) with respect to the task definition, which is now independent from implementation aspects.
The motivation is that the Behavior Developer does not need to know the details about the software and hardware components that will be employed to ultimately realize the robotic application, delegating those choices to the System Builder. Moreover, the same task specification can be performed by another robotic system, consisting of different components (software and hardware), but having analogous *capabilities*.
The list of skills (i.e. the *Skill models*) available to the Behavior Developer for programming the tasks are part of the *Digital Datasheet* of the system. Following the RobMoSys work flow:

- a user in the *Domain Expert* role defines the set of relevant skills in the form of *Skill Definitions* models;
- the *Component Supplier* is the role in charge of providing the *Skill Realizations* that implement the previously defined Skill Definitions;
- a user in the role of a *System Builder* composes multiple components, ensuring that the resulting (software) component composition provides the required set of skills;
- finally, the Behavior Developer can select and compose the skills required for realizing the task at hand.

In this first use case we focus on the role of the Behavior Developer. Concretely we demonstrate how a user in the role of the Behavior Developer can program different applications by composing tasks from a catalog of available skills.
The Behavior Developer role relates directly to the *Robotics Behavior Metamodel*. This model defines structures for modeling the sequence of tasks the system must execute in order to realize a given application. These tasks can be organized hierarchically, but at the lower level of the hierarchy the tasks are actually executed by orchestrating subordinate software components. Skills serve as the interface between the software components and the tasks. *Behavior* models represent the functionality of the system on a symbolic level. They define how a certain task is archived by coordinating and configuring the software components of the system thus making use of the functionality realized within the components. The skill models use an explicated coordination/configuration interface to interact with the components in the system. This interaction includes, for example, run-time configuration using modeled parameters, activation of the activities within the component and control of the components life-cycle. Given that the focus of this first use case is to demonstrate task-level composition, the Robotics Behavior Metamodel is one of the most relevant models for this use case. By demonstrating how the Behavior Developer combines tasks to realize an application we show how the Behavior models make composition possible.
The machine tending application should have a *Domain Model* on its own and in general intersect and depends on several robotic Domain Models. In our particular scenario we limit ourselves to

the *Motion*, *Perception*, *World Models* and *Flexible Navigation* domains. However, contributing to these domains is not the focus of this use case. The Skills Definitions for (mobile) manipulation and machine tending required for realizing the application of this use case are only required for the implementation and are not meant to extend existing domains nor to propose new ones.
This first use case focuses on *Task-level Composition* of behaviors. Tasks can be executed in sequence or in parallel (horizontal composition) and can build hierarchies (vertical composition). Task hierarchies ca be static (i.e. scripted) or be computed dynamically by a symbolic planner, for example. In this use case tasks (hierarchies) are static. The Behavior Developer explicitly specifies the sequence of tasks that realizes the desired application. This includes sequences for dealing with contingencies. In this use case, *Behavior Trees* where used to script the tasks. Other relevant composition types and aspects such as *Service-based Composition*, *Managing Cause-Effects Chains in Component Composition* and *Coordinating Activities and Life-Cycle of Software Components* are not in the scope of this use case.
For developing the RobMoSys-conform models and software components required for realizing the application in this use case, the *SmartMDSD* toolchain was chosen as integrated development environment. The SmartMDSD toolchain was the most adequate RobMoSys-conform tool currently available for modeling and implementing the required skills and other software components. In addition to a relatively large software baseline of existing RobMoSys-conform models and software components, the toolchain offers convenient features such as automatic code generation and deployment. For defining and executing the Behavior Trees that realize the applications, the BehaviorTree.CPP framework was used. Groot?

2.5 Technical Details

In this section some technical details about the current state of the Pilot development are presented. In particular, in this deliverable only details on the machine tending scenario are presented, with particular focus on the modeling aspects. Fig. 2.5 depicts the system component architecture diagram, as modeled in the SmartMDSD Toolchain. The system consists mostly of mixed-port components such as the `ComponentSkillLocateMaterial`, `ComponentSkillPick` and `ComponentSkillMovePlatform` components for accessing the base system functionality of the system. The `ComponentTaskController` component realizes the machine tending application by orchestrating the components implementing the system skills.
For some of the devices employed, a ROS interface has been previously developed, as part of "code legacy" of the RobMoSys partner leading this Pilot (Siemens). To speed up the development time, in this first phase the ROS interface is used to provide access to these functionalities for the machine tending and flexible assembly applications, since those have been previously implemented. This is not a limitation of the Pilot itself, given the focus that the Pilot considers (ie, composition at task level, task programming). Besides, this Pilot also shows how RobMoSys can handle and "play nicely" with legacy code, even if the RobMoSys benefits will be limited by the capabilities and the design of the existing code base (eg, verification of non-functional properties, limited reconfigurability/composability, etc). However, some software components may be replaced with native RobMoSys components to highlight the benefits to the overall system that the RobMoSys approach enables.
In order to access to those base functionality from RobMoSys tooling and components over ROS, mixed-port components (with ROS) were implemented. A software component for the specification and execution of the machine tending task is used. An alternative realization using a Behavior

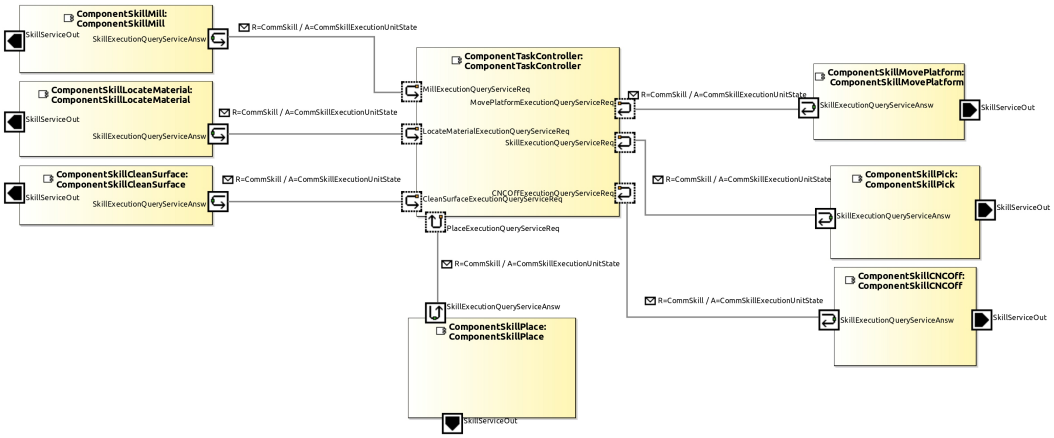

Figure 2.5: System component architecture diagram. The system consists largely of mixed-port components that provide access to the system functionality realized in ROS (legacy). An executive control component executes the application by orchestrating the other components.

Tree has been implemented as well.

In details, mixed-port components access the functionality implemented in ROS and provide a service for this functionality that can be then used by other RobMoSys component, acting as a bridge between the ROS and RobMoSys-based systems. The base system functionalities are grouped as skills, making the interfacing of both systems be at the same level.

Although each mixed-port component could directly interface with its corresponding ROS counterpart, in the current realization of the system, a single ROS service was used for accessing all the skills implemented in ROS. Through the parameters passed to this ROS service, the intended skill is then identified and executed within the ROS system. Fig. 2.6 depicts the structure of the `ComponentSkillLocateMaterial` and the `ComponentSkillPick` components, as modeled in the SmartMDSD Toolchain. As expected, the structure of the two components is identical. Only the names of the component Modes, the parameters and the results differ. For example, the parameter of the `ComponentSkillLocateMaterial` consist of the id of the type of object to be located whereas the parameter of the `ComponentSkillPick` consists only of the id of the object to be picked.

In the current system, the management of most world model data takes place within the ROS system. For instance, the content of the result of the `ComponentSkillLocateMaterial` component is the id of the object that was located. The actual (6D) pose of the located object, for example, remains within the ROS system and is never translated into a RobMoSys object to be used by other RobMoSys components. The `ComponentSkillPick` component then uses the id of the located object. The association between the object id and its pose takes place within the ROS system. The RobMoSys component responsible for the orchestration of the other components simply passes the results from a one component to another.

The approach for managing the world model information described above is just an t temporary work around for the world model. The approach only works in well structured, static scenarios and doesn't support for real contingency planing. A `World Model` component is needed for managing and providing access to world model information to other RobMoSys components in the system. For example, the `ComponentSkillLocateMaterial` component should add as result of its execution a new instance of the located object into the world model indicating its pose and

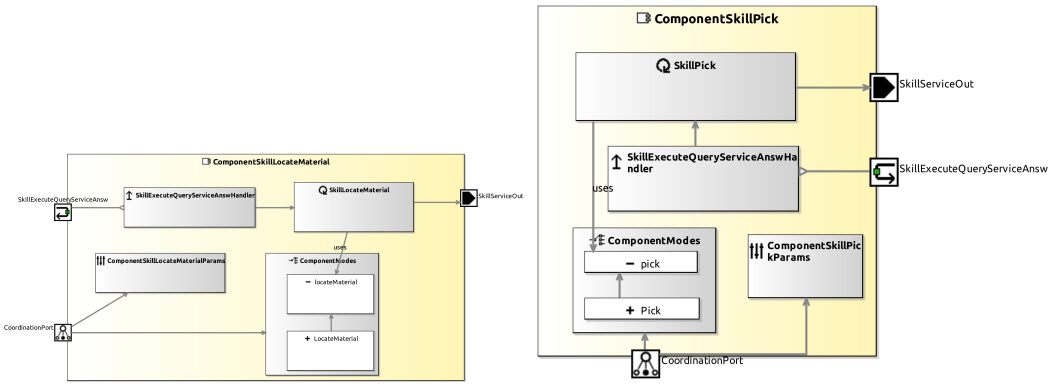

Figure 2.6: Structure of the `ComponentSkillLocateMaterial` and the `ComponentSkillPick` mixed-port components for bridging the ROS and RobMoSys systems.

other relevant information such as a time stamp. The `ComponentSkillPick` component could then directly query the `World Model` to obtain the pose of the object to pick.

The most important functionality of the world model is to keep track of the poses objects in the world over time with respect to an associated coordinate frame. For example, the 3D pose of the manipulated object. This pose could be represented, for instance as a 3D position with a 3D orientation where the 3D position consists of 3 values for the x,y and z coordinates of the object and the 3D position is represented as 3 values for the yaw, pitch and roll of the object.

The navigation system of the mobile platform used for the machine tending application poses interesting challenges for the world model component. The `ComponentSkillMovePlatform` component requires for example a map of the environment for navigating. This map is a two-dimensional representation of the environment where line-segments are used to represent the obstacles in the environment. The different locations of interest in the environment such as the storage location and the location of the milling machine are specified relative to the map as a 2D position with an orientation. For fine localization, locations have their own associated local map, where other geometrical features in addition to line segments are used for more precisely estimating the pose of the platform within the local maps. Clearly, the 6D pose of the objects to be manipulated must also be included in the world model. So the `World Model` component needs to model not only the geometry of the world but also the topology of the world consisting of the system, the map, the local maps, and the object poses.

As the mobile platform navigates through the environment, its pose in the world model has to be updated. If an objects is being transported, then the pose of the object has to be automatically updated with the pose of the mobile platform. Any other component should be able to query the state of mobile platform, the object being transported, or any other object of interest. And the query should be possible using different reference frames. Once the transported object is "detached" from the mobile platform, its pose doesn't change anymore when the pose of the mobile platform changes. These are just a few of the functionalities that the world model needs to realize for this scenario.

2.6 Key Performance Indicators

In this section we present the set of key performance indicators that are to be used for evaluating the benefits of the RobMoSys approach in the context of this Pilot. Since this Pilot focuses on task-level programming and composition, the proposed performance indicators try to asses the effort associated to programming and system composition at task-level.

2.6.1 Goals

At this stage, we focuss on the following overall goal of the RobMoSys approach:

SIE-G1 Reduce application programming efforts by increasing the harmonization and interoperability (composability) of the system functionalities.

The reduction of application programming and in general system engineering efforts is, for Siemens, one of the most promissing expected benefits of the RobMoSys approach. Studies show that about 35% of the total cost of robotics systems are associated to the engineering and programming of the system.

2.6.2 Questions

Based on the overall goal presented in the previous section, the following questions were formulated to help estimate the level of achievement of the goal in the context of this Pilot. The questions are formulated from the point of view of the user in the *Behavior Developer*

- Questions related to goal `SIE-G1: Reduce application programming efforts`

  SIE-Q1 How do I choose the right functionality?
  One important and time-consuming step during task-level programming is selecting the appropriate functionality needed for realizing the desired task. Enabling and supporting the user in the *Behavior Developer* to decide on the required functionality would lead to a reduction of the overall programming effort. Related questions would be, for example: How do I know what system functionality is available, and how can I be sure that the functionality does what is advertises to do?

  SIE-Q2 How can I find out which functionality/component is not performing as expected?
  Another important, and also time-consuming step in task-level programming is verifying that the selected (functional) components are behaving as expected. This is true not only for the case of a mal-functioning component or system (i.e. debuggin) but also for analzing the general behaviour of the system and its individual components. A related question for the user in the rolo of the *Behavior Developer* would be: How can I be sure that my application does what it is intended to do?

2.6.3 Metrics

The following metrics should help to answer the questions posed in the previous section. The formulas are not real formulas and only intended to better explain what the metric sis trying to meassure. As with many metrics, without a reference value, they are only to be understood

as an indicator. Furthermore, most of them are variations of the same underlying metric of complexity. Even without a reference value, the reduction of complexity is a valid goal. The appropriate structures and an adequate separation of roles and concerns as proposed by the RobMoSys approach should mitigate the problems associated to the complexity of a system.

SIE-M1: Number of functional components

$$M_{\text{sie-m1}} = \text{Number of available functional components} \tag{2.1}$$

The number of functional components to choose from at the moment of programming an application can be used as an indicator of the associated programming effort. The larger the number of functional components, the larger the number of possible component compositions. This highlights the importance of appropriate tools for automating the composition and validation of applications. However, even with the support of such tools, the larger the number of components, the more numerous the number of choices the user in the *Behavior Developer* role.

SIE-M2: Number of configuration parameters

$$M_{\text{sie-m2.1}} = \text{Total number of configuration parameters} \tag{2.2}$$

$$M_{\text{sie-m2.2}} = \text{Average number of configuration parameters pro functional component} \tag{2.3}$$

Similar to the number of functional components $M_{\text{sie-m2-1}}$, the number of configuration parameters relevant for the task of programming an application can be used as an indicator of the associated programming effort.
Each components has its own set of configuration parameters that specify the behavior of the realized functionality. The larger the number of configuration parameters, the larger the effort related to correctly parameterizing the application.

SIE-M3: Number of traceable properties

$$M_{\text{sie-m3.1}} = \text{Total number of traceable properties} \tag{2.4}$$

$$M_{\text{sie-m3.2}} = \text{Average number of traceable properties} \tag{2.5}$$

The traceable properties of a functional component are used for monitoring and analyzing the behavior of the component. The number of traceble properties can be used as an indicator of the associated programming effort, in particular testing and debugging efforts: the larger the number of properties, the more complex it is to monitor and analyze behaviors. However, small numbers of traceable properties don't necessarily correspond to less programming efforts. Too few traceable properties can lead to "blind" testing and debugging.

SIE-M4: Number of different implementations of the same functionality

$$M_{\text{sie-m4}} = \text{Number of implementations/Number of functionalities} \tag{2.6}$$

It is possible to have more than one implementation of the same functionality. Choosing between different implementations for the same functionality is another design choice for the users in the *Behavior Developer* and *System Builder* roles. The larger the number of different implementations of the same functionality the higher the programming efforts.

Goal SIE-G1: Reduce application programming efforts			
Question	Metric	Current	Target
SIE-Q1: How do I choose the right functionality?	SIE-M1: Number of functional components	High	Low
	SIE-M4: Number of different implementations of the same functionality	High	Low
	SIE-M5: Variation between functionally similar components	High	Low
SIE-Q2: How can I find out which functionality/component is not performing?	SIE-M2.1: Total number of configuration parameters	High	Low
	SIE-M2.2: Average number of configuration parameters pro functional component	High	Low
	SIE-M3.1: Total number of traceable properties	High	Low
	SIE-M3.2: Average number of traceable properties	High	Low

Table 2.1: Benchmarking plan for goal SIE-G1: Reduce application programming efforts.

SIE-M5: Variation between functionally similar components

$$M_{\text{sie-m5}} = \sum \text{Different configuration parameters} + \sum \text{Different traceable properties} \quad (2.7)$$

The same functionality ("object detection" functionality, for example) can be realize by different components. If the configuration parameters and traceable properties of these components where identical, the user in the *Behavior Developer* role could simply replace one component with another without having to change anything else in the system. The larger the number of differences between the components that realize the same functionality, the more involved is the change.

2.6.4 Benchmarking Plan

The following table presents the first version of the benchmarking plan to asses the overall benefit of the RobMoSys approach in the context of this Pilot. During the next pahse of the project, with the implementation of the Pilot application, this plan will be revised and at the end, concrete values should be provided to the metrics.

3. Healthcare Assistive Robot (PAL)

3.1 Progress Summary

During the period M12-M36 the main effort was spent on the development of the pilot skeleton with minimal capabilities and a basic demo. The scenarios about the *Healthcare Assistive Robot* were sligthly redesigned and tuned respect the last report on the Pilot progress, such as to better show the benefits of the RobMoSys approach. The work done made us progress in the realization and the selection of the components involved in the Use Cases. A lot of work was done also to select the models, the services, the configurations needed to exposure the basic functionalities of the TIAGo robot related to the healthcare environment in the RobMoSys framework.

3.2 Pilot Scenario and Use Cases

This Pilot use case focuses specifically on robotic health-care adaptation. It showcases the development and programming of assistive mobile robots in unstructured and dynamic environments where the robot has to perform complex tasks combining several capabilities such as mobility, perception, navigation, manipulation and human-robot interaction. Furthermore, these capabilities have to be customised for an individual and for an specific apartment.
The key to the success on these changeable enviroment is to encapsulate the different functionalities at design time, to combine them at deployment time and to leave them interact at runtime.

The Pilot Scenario

The system addressed in this Pilot is a TIAGo mobile base manipulator. Figure 3.1 depicts the TIAGo robot in a standard apartment. The TIAGo robot consists of a wheeled mobile manipulator equipped with odometry, a laser scan, a depth camera, and a thermal sensor. The different capabilities can be test in simulation and on the real robot.

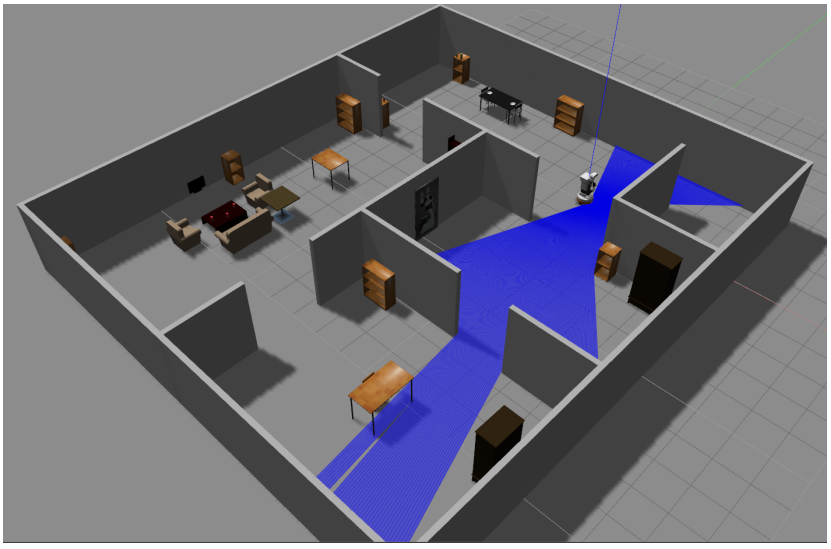

Figure 3.1: Simulation of TIAGo robot in a standard apartment

The use cases

Two use cases are considered: replacement of components and task coordination.
In the first use case, the user plays the role of the *System Builder* that exchanges components with the same functionality. The demonstrator prepared consist of an application that detects people using different combinations of software and hardware components. Like for example exchanging the component of the RGBD camera that use deep learning to detect people with another component that is represented by the thermal camera that doesn't use the deep learning techniques but that is more expensive.
In the second use case, the "user" plays the roles of the *System builder*, who configures the appropriate criteria to build the robot system, and the *Behaviour Developer* who prepares the coordination of the system components to look for standard objects in an apartment.

3.3 Setup Description

The setup consists of a TIAGo assistive mobile manipulator robot navigating in private apartment to assist the person living in the flat, like in the example of Figure 3.2. The setup can be tested both in simulation and on the real robot in order to make available the pilot to a broader audience or to test it before hand if the details are known in advance.

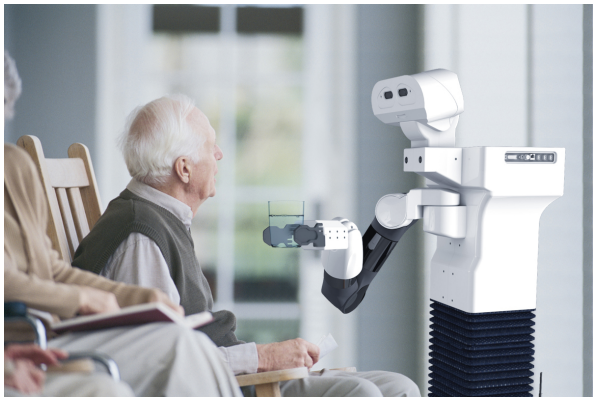

Figure 3.2: Example of TIAGo robot assisting an old man

The platform is a TIAGo equipped with a differential mobile base and a manipulator, such as the one depicted in Figure 3.3. This robot integrates a PAL 7 DoF arm equipped with a 6 axis force torque sensor and an end-effector: a PAL Hey5 hand, a PAL parallel gripper or a Schunk WSG32 gripper. The base is includes a prismatic column, a pan-tilt head with a RGB-D camera and, a thermal camera. Several human robot interfaces are integrated: a speaker and an stereo microphone. And also GPU can be attached to the robot.
From the control software point of view, PAL provides several off-the-self functionalities such as: joystick base teleoperation, torso lifting, head and end-effector.
The available controllers allow commanding the wheels in velocity. Head and torso in position and the arm in position and effort.
For the navigation stack, TIAGo provides a path planning with self-collision avoidance, mapping and self-localisation and from the Human Robot Interaction point of view, there are several functionalities available such as Text-to-Speech, automatic speech recognition, people and face recognition.

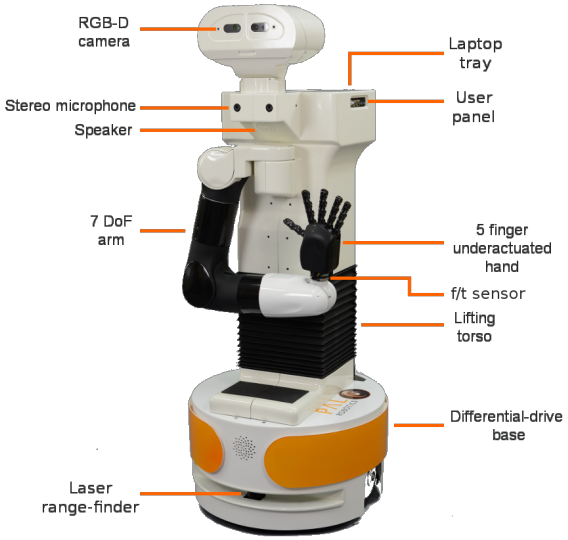

Figure 3.3: TIAGo main components

Most of these functionalities are exposed to SmartSoft through a software bridge developed by PAL. However it might be further generalised using the SmartSoft concept of ROS Mixed Port SmartSoft implementation.
Except some third party libraries for the text to speech and automatic speech recognition, most of the software is provided as a BSD or Apache license that allows easy integration into existing proprietary or open source code.
The development and deployment of the application can be done easily from a docker containers specifically created by PAL which contains everything needed:

- Ubuntu 16.04
- Gazebo simulator
- SmartSoft tooling
- ROS Kinetic
- PAL TIAGo configuration
- machine learning libraries and datasets

3.4 Pilot Focus and Coverage of RobMoSys Features

3.4.1 Use case 1

The focus of this first use case pilot is to show replacement of components to address the environment variability that arises in the health-care scenario.
In this context, the main RobMoSys Ecosystem Role is the one of the *System Builder*.
The most important source of variability for a service robot is the environment variability. That can be overcome choosing the appropriate set of hardware and software components.

The motivation of this pilot is that a new robot has been commissioned for a new apartment where a elderly person lives in need of assistance.
The system builder wants to provide the optimum cost-effective combination that fullfil the requirements.
In this use case, the replacement of components will be illustrated by exchanging the people detection module. On one hand, if the elder person is prone to loss consciousness and fall down, then the skeleton tracker algorithm is not effective (it is optimised for the stand up person detection case). On the other hand the robot can be equiped with an expensive thermal camera that allows detecting people easily in different configurations as illustrated in the Figure 3.4.

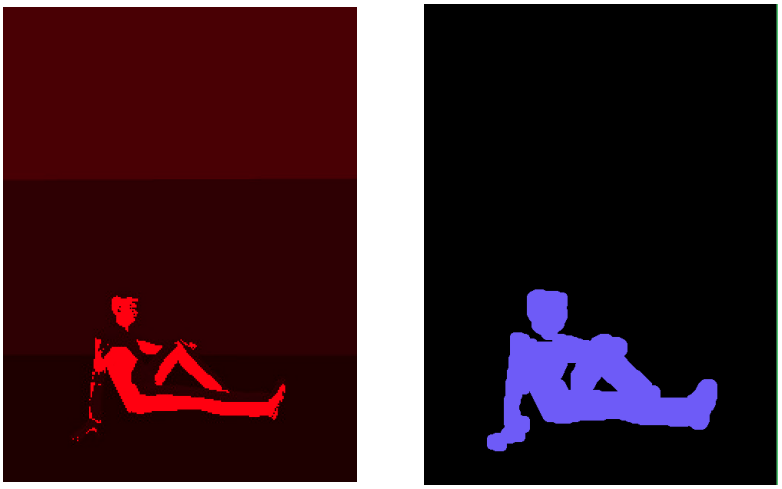

Figure 3.4: Simulation example of people detection with the thermal camera. The output of the thermal camera is on the left. The output of the people detection algorithm on the right indicates the color of the temperature gradient of the person detected.

The *System builder* has to choose the best combination of hardware components to identify people based on the requirements for an specific apartment: welcoming visitors, detect fallen people or both.
During the commission of a TIAGo robot to be installed on a new apartment, the RobMoSys work-flow will be as follows:

- A Domain Expert user defines the people who have to be identified for a specific scenario.
- A *Component Supplier* will be in charge of providing the software and hardware components to fullfil that requirements.
- Finally a user in the role of *System builder* combines the building blocks.

Based on a system architecture, the *System Builder* selects several components from the component suppliers that realise the needed services.
This role ensures that the services chosen guarantee the required person detector accuracy requested by the Domain Expert on a specific environment.
The RobMoSys composition relies on the *Component-Definition Meta-model* and the *System Component Architecture Meta-model*.

The TIAGo health-care application depends on several robotic Domain Models. In our particular scenario, the appropriate domains consist of the *Perception*, *World Models* and *Flexible Navigation* domains. However, in this pilot the existing domains are not going to be extended.
This first use case is focused on *Composition* of components. The *System Builder* will choose to substitute the people detection algorithm to be executed from the RGB-D camera or from an thermal camera.
For developing the RobMoSys-conform models and software components required for realising the application in this use case, the *SmartMDSD* toolchain was chosen as integrated development environment.
In order to reuse the existing services of the TIAGo robot a software bridge component to the legacy system has been deployed. This is not a limiting factor because the whole SmartSoft infrastructure is been used.
In order to access to those low-level functionality from RobMoSys tooling a bridge to the TIAGo base was implemented.
The use case is open to the possibility to add the Performance Designer Role to add the management of the non-functional properties analysis.

3.4.2 Use case 2

The second use case is focused on system composition at service level and on task coordination at the task level. The idea is to combine several services to handle environment variability. Specifically, the aim is to identify objects in an apartment just by telling the robot which type of object. The list of objects is fixed at runtime but can be extended at configuration time.
This use case is based on three different tiers for sytem composition:

- Tier 1 Basic structures independent from the robotic domain. Mainly, the concept of service composition, the concept of component and the composition workflow.
- Tier 2 Defines the structures within service robotics. In this scenario, the oject recognition and the localization and mapping. Specifically, for example, details on how the pose of the TIAGo robot is represented.
- Tier 3 This is where the PAL contribution takes place. Concrete services that provide object recognition, camera movement and robot patrol in an apartment.

In this second use case, there are four roles involved. The *Domain Expert* who designed the individual skill definitions for use by Tier 3 roles.
Component Developer(Tier 3) models and implements components. In this case, for example, the object recognition component and the head movement service. The *Component Developer* also is in charge of providing the *Skill Realizations* based on the skill definitions. For the purpose of this pilot, the skill definitions will be fully implemented during the realization of the application of the pilot. The coordination interfaces of each component are already defined fully, it is only missing the skill datasheet definition to be exported as list of skills for the *Behaviour Developer*.
System Builder(Tier 3) builds the system. This is a SME company such as PAL Robotics commissioning a health care robot for a hospital.
Behaviour Developer(Tier 3) builds applications, selecting and composing the skills needed for the task.
Service definitions act as link between roles and activities. With service definitions, the system builder can identify current solutions to build applications. For example, are there any available

services for service robot to identify common use objects. The key aspect here is that the services decouple interactions so it is possible to compose components free of interferences.
This second pilot focuses on *service-based composition* realised by the *System Builder* composing components implemented by the *Component Developer*. The RobMoSys composition counts on the *Component-Definition Meta-model* and the *System Component Architecture Meta-model*.
This pilot relies also on the *Robotic Behaviour Metamodel* that is responsible for specifying the run-time behaviour of the robot. This model defines structures for modeling the sequence of tasks. Skills are used to link the tasks to the software components. The *Domain Models* are the same ones that are used in the first use case.
This use case is also using the SmartMDSD tooling. The skill interfaces from SmartMDSD will be used and also the pilot skeleton will be extended by the final developers to integrate the behaviour coordination approach using the SmartTCL sequencer or the Behavior Trees available with the SmartMDSD tooling.

3.5 Technical Details

In this section some technical details about the current state of the Pilot development are presented. In particular, in this deliverable only details on the health-care scenario are presented.
Figure 3.5 depicts the system component architecture diagram corresponding to use case 2, as modelled in the SmartMDSD Toolchain. The system component architecture is mostly similar to the one used for the use case 1, the only difference is that the *ComponentBridgeTiagoInference* can be exchanged with the component for the thermal camera.

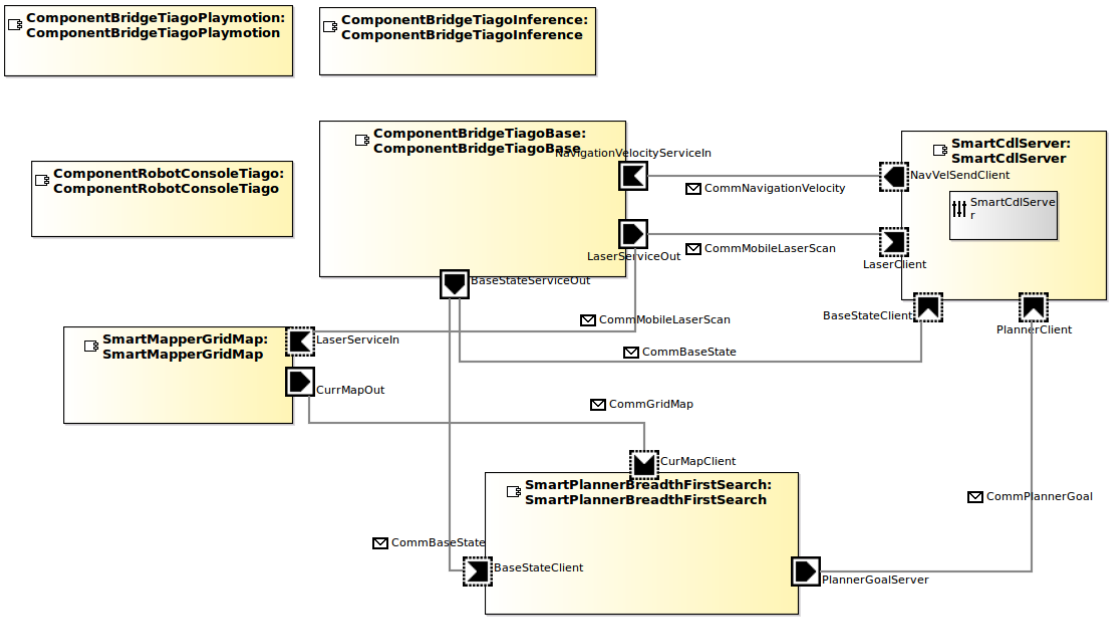

Figure 3.5: Object detection system architecure

In the first use case PAL has developed several components that model the bridge with two

different algorithms, using two different sensors that can be swapped or combined:

- RGB-D camera + Deep learning people detection
- Thermal camera + OpenCV people detection

The idea is to choose the right combination to fulfil the requirements.
To speed up the development time, the components are bridge made by PAL, as ROS interfaces, to access the legacy code existing for the TIAGo robot. It also shows how RobMoSys works finely with legacy code that have been previously implemented, even if the RobMoSys benefits will be narrowed by the capacities and the design of the existing code. Nonetheless, some components may be replaced with native RobMoSys components like in our architecture where we are using the RobMoSys navigation components, replacing our navigation stack that is developed in ROS inside the TIAGo framework. The *ComponentBridgeTIAGoBase* allows communicating with the TIAGo base to get the information from the laser sensor, the base position and velocity and to send to the robot velocity commands.
In the current structure, the management of the world model data is done within the ROS system. Like, for example, the association of the object id and the pose in the world, or the association between the motion id and the joints configurations of the arm, head and torso is done at ROS level combining our play motion package, the tf ROS package and the controllers. For the second use case there is also the possibility of the use of a Text-To-Speech component based on the *Loquendo* propriertary libraries that was developed by PAL to have a bridge with the Loquendo software included in the TIAGo robot. Due to that the software is propriertary it is only possible to test the component on the real robot and not in simulation.
In the current status, only the RobMoSys navigation components expose the Skill Realization. The other components bridge with TIAGo robot, realized by PAL, are implementing the coordination interface thus to be predisposed for the Skill Realization. These components implement the operation modes of the component to be activated and deactivated when needed and also the instantiation of some trigger parameters to receive the command to be executed. The coordination interface is integrated in the new robot console that we created extending the existing component.
Summarizing, these are the several functionalities developed for the use cases:

- *ComponentBridgeTiagoBase* is the bridge with the TIAGo base to get the information from the laser sensor, the base position and velocity, and to send to the robot velocity commands.
- *ComponentBridgeTiagoInference* does the object and people detection given a compressed image, using the deep learning algorithms, as shown in the Figure 3.6.
- *ComponentBridgeTiagoThermal* does the people detection using the image from the thermal camera and the OpenCV algorithms.
- *ComponentRobotConsoleTiago* calls the coordination interfaces to run the navigation demo, play motion demo and object detection demo.
- *ComponentTiagoPlaymotion* executes the motion expressed inside the robot by joints configurations. It can be used, for example, to orientate the TIAGo head to detect objects or people.
- *ComponentTTSLoquendo* is a bridge with the Text-To-Speech Loquendo libraries.

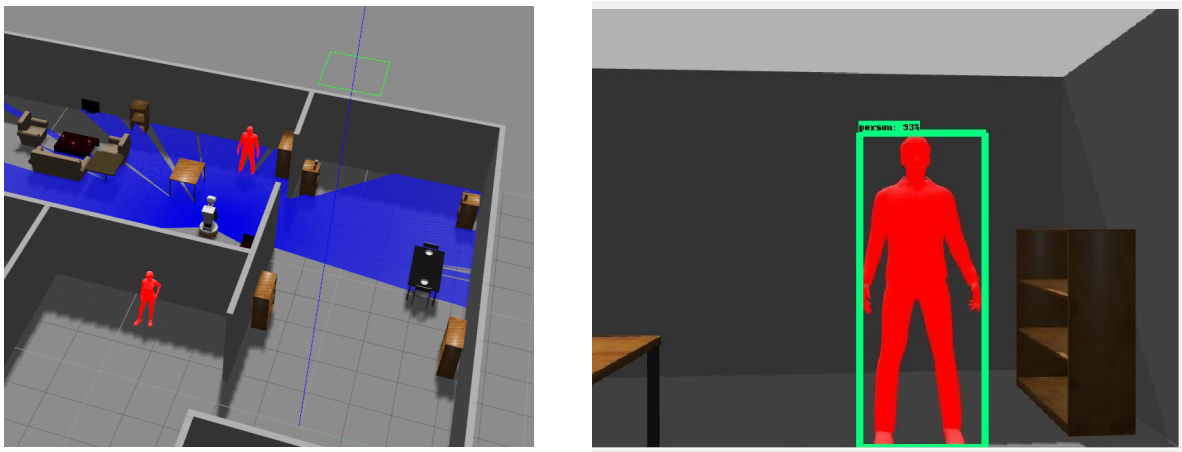

Figure 3.6: Simulation example of people detection with the deep learning component

3.6 Key Performance Indicators

The performance indicators for the first use case will measure the reducing of the time of the robot manufacturing that fulfils the functional requirements. The performance indicators for the second use case will measure the gains in the development productivity using the RobMoSys approach for task coordination and composability.

3.6.1 Goals

The following goals of the RobMoSys approach for the TIAGo healthcare pilot:

PAL-G1 Reduce robot manufacturing time by reducing the integration effort.

PAL-G2 Increase development productivity by increasing task coordination and composability (decoupling roles and concerns).

3.6.2 Questions

The following questions help to estimate the level of achievement of the goals:

- Questions related to goal `PAL-G1`

  PAL-Q1 How the composable approach reduces the time effort of the component replacement with respect to non model driven approach?

  PAL-Q2 How do I compare the performance of people detection with the different setups?

  PAL-Q3 How can I be sure that the application keeps under certain bounds after the replacement of the component?

- Questions related to goal `PAL-G2`

  PAL-Q4 What is the time effort to create a new task that fulfils the requirements?

  PAL-Q5 How many people do I have to involve to define a new task and how separation of roles helps it?

PAL-Q6 How the skill abstraction (hiding implementation) allows reusing components when a new task is designed?

3.6.3 Metrics

This section describes the metrics used for answering the questions posed in the previous section

PAL-M1: Time effort to select a component for a particular setup

The metric is the working days spent to select, exchange, configure and deploy the component.

$$M_{\text{PAL-M1}} = \frac{\text{Time to select, exchange, configure and deploy the component with RobMoSys}}{\text{Time to select, exchange, configure and deploy the component without RobMoSys}} \tag{3.1}$$

PAL-M2: People detection identification ratio

The metric is the percentage of success detections from the total of captured images for a specific setup.

$$M_{\text{PAL-M2}} = \frac{\text{People identified in the scenario}}{\text{Total people in the scenario}} \tag{3.2}$$

PAL-M3: Average time spent to execute the application for a particular setup

The metric is the average time spent, for a particular setup, to execute the application consisting on: navigate through specific waypoints, detect the person and go back to the starting point.

$$M_{\text{PAL-M3}} = \frac{\text{Average time to run the application with RobMoSys}}{\text{Average time to run the application without RobMoSys}} \tag{3.3}$$

PAL-M4: Time effort to combine the existing skills to generate the object detection task

The metric is the working days spent to generate a task assuming that the different functionalities are already implemented.

$$M_{\text{PAL-M4}} = \frac{\text{Time to run the application with RobMoSys}}{\text{Time to run the application without RobMoSys}} \tag{3.4}$$

PAL-M5: Number of people/roles involved in the design of a new task

How many roles are affected by a new task creation.

$$M_{\text{PAL-M5}} = \frac{\text{Number of roles involved in the design of the application with RobMoSys}}{\text{Number of roles involved in the design of the application without RobMoSys}} \tag{3.5}$$

PAL-M6: Number of lines of code that have to be written to reuse the components for a new task

How many lines of code have to be written to reuse an existing functionality.

$$M_{\text{PAL-M6}} = \frac{\text{Number of lines of code written to reuse and existing functionality with RobMoSys}}{\text{Number of lines of code written to reuse and existing functionality without RobMoSys}} \tag{3.6}$$

PAL-M7: Component reusability

Number of instantiations used on task level for each skill.

$$M_{\text{PAL-M7}} = \frac{\text{Number of instatiations used on task level for each functionality with RobMoSys}}{\text{Number of instatiations used on task level for each functionality without RobMoSys}} \quad (3.7)$$

3.6.4 Bechmarking Plan

Goal PAL-G1: Reduce robot manufacturing time by reducing the integration effort			
Question	Metric	Current	Target
PAL-Q1: How the composable approach reduces the time effort of the component replacement with respect to non model driven approach?	M1: Time effort to select a component for a particular setup	Low	High
PAL-Q2: How do I compare the performance of people detection with the different setups?	M2: People detection identification ratio	Low	High
PAL-Q3: How can I be sure that the application keeps under certain bounds after the replacement of the component?	M3: Average time spent to execute the application for a particular setup	Low	High

Goal PAL-G2: Increase development productivity by increasing task coordination and composability (decoupling roles and concerns)			
Question	Metric	Current	Target
PAL-Q4: What is the time effort to create a new task that fulfils the requirements?	M4: Time effort to combine the existing skills to generate the object detection task	Low	High
PAL-Q5: How many people do I have to involve to define a new task and how separation of roles helps it?	M5: Number of people/roles involved in the design of a new task	Low	High
PAL-Q6: How the skill abstraction (hiding implementation) allows reusing components when a new task is designed?	M6: Number of lines of code that have to be written to reuse the components for a new task	Low	High
	M7: Component reusability	Low	High

4. Modular Educational Robot (COMAU)

4.1 Progress Summary

During the reporting period the main target has been the development and setup of a flexible pick and place scenario. The pilot targets to educational applications in public environments as schools or educational institutes supplying a new robotics platform that enables teachers and students to perform and design several robotics applications with different levels of complexity. The models applied to the pilot case, the physical development of the pilot scenario and application on educational enviroments will be the main focus of activities.
As of now, the Pilot application is still being designed and developed, so not may details can be reported.

4.2 Pilot Scenario and Use Cases

The pilot is intended to showcase the flexibility and modularity of the system via composition of software components in order to build a complete running application in an easier and faster way respect to standard methodologies. The pilot combines the SmartMDSD Toolchain and the existing software infrastructure of the e.DO robot addressing the RobMosys approach in this new robotics platform and enable teachers and students on performing and designing complex robotics applications. The objectives are to enable:

- Developers to easily design new educational applications
- Students to develop their own functionalities
- Users to extend the robot capabilities with new hardware
- Users to easily integrate the robot with an user interface

4.3 The Pilot Scenario

The pilot case will be based on the open architecture of e.DO platform, a new robot developed for educational purpose that will use a ROS node to connect the Smartsoft environment with the robotics framework. Different uses-cases can be taken into account. Developing customised software functionalites on different levels:

- Basic coding (scratch programming) using task composition
- Emulation of industrial lines to speed up the integration
- Implementation and test of advanced control algorithm

4.4 Setup Description

The pilot is currently located in COMAU plant in Grugliasco (Italy). The pilot can be easily repeated or moved to other locations due to the easy integration and installation of e.DO platform. The available skeleton for the pilot is based on the following components and features:

- e.DO robot platform with open source control logic (based on Raspberry Pi running Raspbian Jessie)
- ROS node for e.DO (Kinetic Kame distribution)
- First integration of e.DO platform with SmartMDSD toolchain IDE
- First set of basic building blocks and models for pick and place application

4.5 Pilot Focus on Coverage of RobMoSys Features

The main focus of this Pilot is to show composition of software and hardware components for educational purpose with a specific focus on easy programming and composition of task level. The main roles for this scenario will be the Behavior Developer that will define tasks using the functionalities provided by components, accessible through Skill Interfaces. The main role of behavior developer would be to program several applications composing tasks from an available database of defined skills. Then the System builder that put together systems from building blocks selecting components (provided by component suppliers) from the ecosystem that realize the needed services. Matchmaking must be made on the basis of offered services and on other properties. Moreover, system builders package everything together (components, models, et) making the system ready for deployment. Another role that are present in the use case will be the performance designer, that is responsible to configure performance-related system properties. The modular educational scenario have a Domain Model based on motion, perception, world model and active object recognition. The SmartMDSD toolchain was the most adequate RobMoSys-conform tool currently available for modeling and implementing the required skills and other software components.

4.6 Technical Details

Main target of the use case is the usage of mixed-port components that will access the functionality implemented in ROS and provide a service for this functionality that can be then used by other RobMoSys component, acting as a bridge between the ROS and RobMoSys-based systems.

4.6.1 Key Performance Indicators

4.6.2 Goals

The following overall goals of the RobMoSys approach are focus of the Pilot:

- **COM-G1** Increase the robot programming ease using model-driven approach
- **COM-G2** Improve the flexibility and opening of robotics platform

4.6.3 Questions

The following questions help to estimate the level of achievement of the overall goals:

- Questions related to goal `COM-G1`

  - **COM-Q1** How a model-driven approach could increase ease of use?
  - **COM-Q2** How task composition could reduce programming efforts ?

- Questions related to goal `COM-G2`
  - **COM-Q3** Why an open approach can help the user?
  - **COM-Q4** How flexibility could decrease effort?

4.6.4 Metrics

This section describes the metrics used for answering the questions posed in the previous section.

- Metrics related to goal **COM-Q1**:
  - **COM-M1**: Effort on programming phase (time)
  - **COM-M2**: Usability (survey)
- Metrics related to goal **COM-Q2**
  - **COM-M3**: Effort of Training phase (time)
  - **COM-M4**: Number of tasks (number)
- Metrics related to goal **COM-Q3**
  - **COM-M5**: Usability (survey)
  - **COM-M6**: Effort on components integration (time)
- Metrics related to goal **COM-Q4**
  - **COM-M7**: Effort on set up phase (time)
  - **COM-M8**: Number of product changes (number)

5. Human Robot Collaboration for Assembly (CEA)

5.1 Progress Summary

In the past period, the focus of the pilot was to prepare the following material for open call 2 experiments:

- A virtual machine with all necessary tools, simulators, software, examples and documentation.
- A simple Pick and Place task with simulated Isybot robot to use, to modify or to build upon.
- Eclipse and Papyrus tool, ROS Lunar and ROS stack for Isybot, A specification document for describing the use case and the modeling steps, and a preliminary version of a risk assessment.

The examples were built to show several aspects of RobMoSys related to task specification, world models and safety functionality.

5.2 Pilot Scenario and Use Cases

In the context of human-robot collaboration, the operator interacts with the robot with no fences and influences the task execution. Thus, taking into account the context and more generally the environment for task definition is both mandatory and challenging at the modeling level.
Human-robot collaboration raises also important safety requirements related to the robot, the tool, the task and the environment. Therefore, safety and more particularly risk assessment (see Figure 5.1) is a major feature that this pilot aims to realize for reducing risk occurence.

5.2.1 The Pilot Scenario

The pilot demonstrates task and environment definition for a human-robot collaboration use case: Pick and Place through RobMoSys tools. The interaction between the robot and the operator is direct (with no fences) for carrying a heavy object from a given starting position to a target one. This pilot uses Isybot collaborative robot but can be extended with any other collaborative robot performing a pick and place task for including more modules like perception and object identification.

5.2.2 The use cases

Two use cases are considered. The first use case is related to safety at design time and the second deals with task robustness at runtime.

Use case 1: Context-aware robustness

The user plays the role of a *Behaviour developer* for modeling the pick and place task in a behavior tree based on the robot skills. Then, as a *System architect*, the user designs the world model where

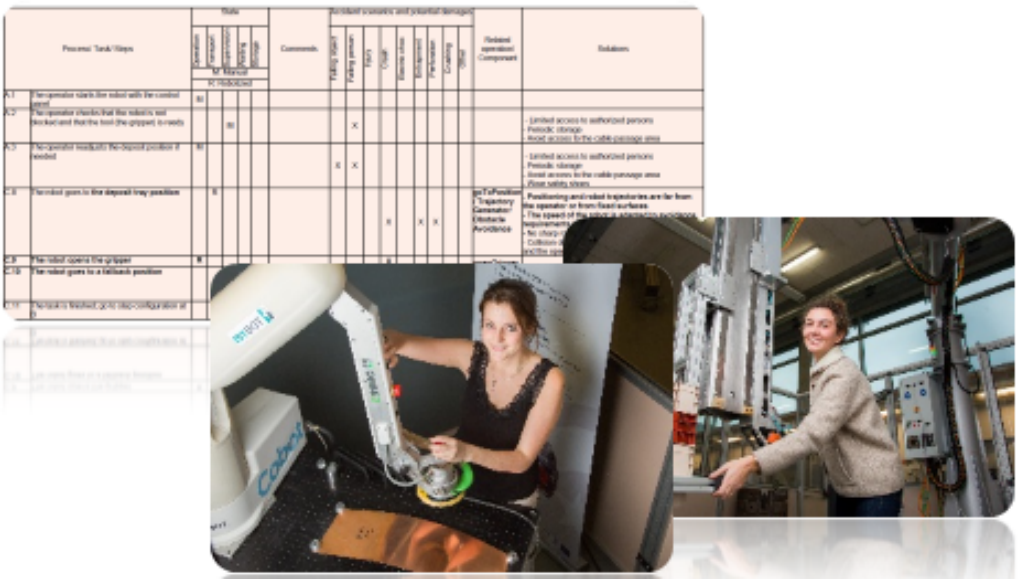

Figure 5.1: Human-Robot Collaboration Examples

the agents (i.e: the robot and the operator) and the objects participating to the task including their affordances are specified. At design time, the task consistency is checked with respect to the world model and the agents' (i.e: the robot and the operator) skills. At run-time, the task is executed with respect to the behavior model and the task constraints are checked continuously based on the environment actual data.

Use case 2: Task-based Risk Assessment and validation of risks' mitigation

The user plays the role of a *System architect* and designs the world model where the agents and the objects participating to the task including their affordances are specified. The *System architect* follows Papyrus4Robotics guidelines that are conformant to safety standards for identifying possible feared events related to the task execution in the environment. The system architect validates the system after implementing all the necessary safety measures.

5.3 Setup Description

The pilot is intended for the use with ISybot collaborative robot[1] but can be extended to any collaborative robot with the same capabilities thanks to the generic task specification implemented in robmosys.
This pilot focuses on task specification and safety functions. In order to start with simple and concrete examples, we chose a pick and place task for objects manipulation. The gripper shown in Figure 5.2 is used for grasping paper. Another gripper can be used to manipulate other categories of objects. The operator interacts with the robot before task execution for teaching the task to the robot or during during task execution for co-manipulation. This pilot will provide a basis that can be enriched with other functionalities in the second open call. A robot arm can be placed on a mobile base for example. A camera or other sensors can be added for object identification.

[1] https://www.isybot.com/

The development and deployment of the application can be done easily through a virtual machine created by CEA that contains all the necessary tools, simulators, software, examples and documentation:

- A Pick and Place task with Isybot robot to use, to modify or to build upon, the risk assessment and the world model
- Eclipse and Papyrus4Robotics tool
- ROS Lunar and ROS stack for Isybot
- A specification document for describing the use case and the modeling steps
- A preliminary version of a risk assessment.

The examples were built to show several aspects of RobMoSys related to task specification, world models and safety functionality.

5.4 Pilot Focus and Coverage of RobMoSys Features

The pilot is intended for open call 2 contributors to showcase context-aware robustness of the system and safety checking at design time. Once the task and the environment are modeled and checked with robmosys tools, the system architect can take advantage of the safety standards integrated into robmosys tools for making a risk assessment and for identifying the feared events and mitigation actions. Once the system validated and deployed, the task is executed conforming to the world, task and data models.
This pilot uses Papyrus4Robotics to comply with robmosys methodology.
The technical expected benefits of the pilot are:

- Easy task description
- Task reusability: Task invariance to slight changes of the environment and/or hardware choices;
- Using most updated norms in order to validate the configuration (environment/robots/humans).

5.5 Technical Details

The pilot is fully supported by Papyrus4Robotics toolchain[2], a Papyrus[3]-based domain specific modeling language for robotics. Papyrus4Robotics features a modeling front-end which conforms to robmosys foundational principles of separation of roles and concerns. Generally, modules like object identification and localization are needed for objects manipulation. In our case, we use programming by demonstration functionality to configure task parameters. We can distinguish the learning phase from the task execution phase in CEA pilot. In the learning phase, the human and the robot collaborate to configure the task parameters (e.g: object position, etc.). In the execution phase, the robot executes the task autonomously or as a co-worker to assist the human in his task. Programming by demonstration is a promising approach to program robots in a fast

[2] https://robmosys.eu/wiki/baseline:environment_tools:papyrus4robotics
[3] Papyrus is an industrial-grade open source Model-Based Engineering tool.

and simple way when the task is known by the user. In this manner, the operator who wants to perform an action with the robot, does not need to know programming languages and control strategies. He only needs to teach the trajectories to the robot by doing it once, and then saving it. The first concept of programming by demonstration functionality has been developed at the CEA Interactive Robotic Lab since 2013. This concept is extended to an established component-based architecture in order to communicate with the robot controller. As result, the programming by demonstration functionality is implemented inside the robot controller and triggers the different control modes depending on the current state of the robot.
Figure 5.3 shows the models that build CEA pilot. The modules Perception and Navigation are marked with blue stars because they can be added to this pilot. The world model presents all the objects participating in the task. The task model defines the actions that will be performed by (or with) an object. There is a tight link between the world model and the task model. First, the world model presents the robot skills. Then, the task model presents the actions that have to be performed in order to achieve the task goal based on the robot skills (if we consider that a task is executed by one robot) with the objects in the environment.
An object can be an actor or a subject based on the context as shown in Figure 5.4. For instance, a gripper is an actor when it grasps an object but is considered as a subject when attached to a robot arm moving to a target position for example. The knowledge from the world model is used when an actor/subject is assigned to the task. For example, ISybot collaborative robot maximum payload is about 10kg. It is then mandatory to define constraints on the manipulated objects and to forbid the robot to perform actions with objects if their weight is higher than 10kg. A set of rules are included at design time in order to define constraints based on the actors/subjects capabilities and the task requirements.
The context allows to define the scope and the consequence of each action executed in the task. The world model in this pilot presents the affordance interfaces attached to each participant in the world model. Figure 5.5 shows an example of the affordance interfaces that will be modeled in CEA pilot using RobMoSys tools.
The world model for task description at design time presents the actors (i.e: objects, humans or robots) participating to the task. Their properties are updated at run-time based on the *data provenance* meta-model. For instance, if we need the operator pose in a component, we have to indicate at design time that this information comes from a Kinect camera for example. This is also the case for the programming by demonstration functionality for configuring the starting point of the task and the object approach position for example.
In addition to the task and world models dependency, the knowledge from both models is used to perform a risk analysis. This functionality is provided by Papyrus4Robotics tool and allows to define a set of possible damages as a consequence of a skill execution.
At run-time, the world model data are updated continuously to reflect the task states change. The World model mediator plays the role of a broker responsible for collecting the data and forwarding it to target the component that requires these data.
Open-call participants can use this pilot as a backbone on top of which new models and software can be contributed to deal with more complex scenarios. These include (but are not limited to) scenarios where objects and their properties are different for several mission runs (perception module), or where the robot moves around in the environment (navigation module).
For more information, see the following resources:

- Pilot description in the RobMoSys Wiki[4]

[4] https://robmosys.eu/wiki/pilots:hr-collaboration

- Safety assessement using fault injection in Papyrus4Robotics in eITUS project[5]
- A video that shows some of the mentioned previous models with skills modeling for robotic behavior featuring the pilot and agile risk assessment[6]

In the following, we present the models that were performed for behavior specification and for risk analysis.

5.5.1 Behavior Specification

Papyrus for Robotics provides a viewpoint for behavior designers, based on the behavior tree (BT) representation[7]. The BT can be modeled directly in Papyrus, so that it can be easily linked with additional models representing complementary concerns, like safety, resource allocation and real-time properties. Figure 5.6 shows the BT model for the pilot pick and place task. The manipulated object is a stack of paper grasped at the output tray of a printer and deposited in the work in progress tray.
The task demands the execution of specific procedures to initialize and prepare the robot (Figure 5.6 left side). The actual pick-and-place task description (Figure 5.6 right side) prescribes a set of robot movements to enter and exit the printer and deposit spaces (these spaces are known and assigned as input parameters through programming by demonstration functionality to the BT leaves representing concrete actions). To maintain the stack of paper on the gripper during its movement and to deal with paper consistency, the robot closes its gripper when moving. For the deposit, the robot opens the gripper first then deposits the paper in the deposit position.

5.5.2 Task-based Hazard Analysis and Risk Assessment (HARA)

Task-Based HARA is performed following ISO 10218-2:2011[8]. For each action in the behavior tree, we list all the relevant hazards and compute their risk index. The risk analysis table structure is extracted from ISO/TR 14121-2:2007[9]. It contains the following information: Task, Hazard, Origin, Hazardous situation, Hazardous event, Possible harm, Occurrence, Avoidance, Frequency, Severity, Criticality as shown in Figure 5.7.
After computing the risk criticality, the safety engineer provides risk reduction measures for each hazard associated to an action as presented in Figure 5.8.

5.6 Key Performance Indicators

We can consider that in RobMoSys, there are general KPIs like reducing programming efforts for example. In this section, we present the KPIs that are specific to CEA pilot. In order to identify the relevant metrics for this pilot, we followed a Goal-Question-Answer-Metric approach.

5.6.1 Goals

`CEA-G1`: The goal of this pilot is to increase the robustness of robotics systems.

[5] https://robmosys.eu/wiki/community:safety-analysis:start
[6] https://www.eclipse.org/papyrus/components/robotics/
[7] https://arxiv.org/abs/1709.00084
[8] https://www.iso.org/standard/73934.html
[9] https://www.iso.org/standard/42712.html

5.6.2 Questions

In order to achieve the goal cited above, the following questions were formulated to help estimate its level of achievement in the context of this Pilot.

- `CEA-Q1`: How can we reduce the safety risk of the system?
- `CEA-Q2`: How can we reduce the effort for validating safety requirements at design time?
- `CEA-Q3`: How can we increase the system robustness at run-time?

5.6.3 Metrics

In the following, we discuss the possible answers and the metrics that should help to measure them.
For `CEA-Q1` about system reliability, we can consider two possible answers:

- `CEA-Q1-A1`: Reducing the safety risk by correctly identifying risks
- `CEA-Q1-A2`: Assistance to integrators (e.g: sensor is missing at a given location, guidance about the architecture, etc.).

Regarding `CEA-Q2`, we can consider the following answers:

- `CEA-Q2-A1`: Model checking for ensuring the system consistency

For `CEA-Q3`, we can consider the following answers:

- `CEA-Q3-A1`: System supervision: the system state at run-time is continuously updated
- `CEA-Q3-A2`: Deviation detection: if the system does not behave as expected, the error should be detected and reported to the system supervisor.

Based on CEA-Q1-A1, CEA-Q1-A2, CEA-Q2-A1, CEA-Q3-A1 and CEA-Q3-A2, the following metrics have been identified. The metrics list is not exhaustive and can be extended in the future.

CEA-M1: Number of Correctly Identified risks

This metric will allow determining how many risks can be identified thanks to the use of RobMoSys tools.

$$M_{\text{CEA-M1}} = \frac{\text{Correctly Identified Risks}}{\text{Total risks}} \tag{5.1}$$

CEA-M2: Effort in making a risk analysis

This metric will allow to compare the time spent by a safety expert in making a risk assessment without RobMoSys tools to the time spent with RobMoSys tools.

$$M_{\text{CEA-M2}} = \frac{\text{Effort for making a risk assessment with RobMoSys}}{\text{Effort for making a risk assessment without RobMoSys}} \tag{5.2}$$

CEA-M3: Effort for validating the system architecture

This metric will allow to compare the effort spent with RobMoSys tools for cheking the system consistency to the effort spent without RobMoSys tools.

$$M_{\text{CEA-M3}} = \frac{\text{Effort for validating the architecture with RobMoSys}}{\text{Effort for validating the architecture without RobMoSys}} \tag{5.3}$$

CEA-M4: Effort for detecting deviations at run-time

This metric will allow to compare the effort for detecting deviations at run-time with RobMoSys tools to the effort spent without RobMoSys tools.

$$M_{\text{CEA-M3}} = \frac{\text{Effort for detecting deviations at run-time with RobMoSys}}{\text{Effort for for detecting deviations at run-time without RobMoSys}} \tag{5.4}$$

5.6.4 Benchmarking Plan

Goal CEA-G1: increase the robustness of robotic systems			
Question	Metric	Current	Target
CEA-Q1: How can we reduce the safety risk of the system?	CEA-M1: Number of Correctly Identified risks	Low	High
CEA-Q2: How can we reduce the effort for validating safety requirements at design time?	CEA-M2: Effort in making a risk analysis	High	Low
	CEA-M3: Effort for validating the system architecture	High	Low
CEA-Q3: How can we increase the system robustness at run-time?	CEA-M4: Effort for detecting deviations at run-time	High	Low

Figure 5.2: RobMoSys ISybot collaborative robot

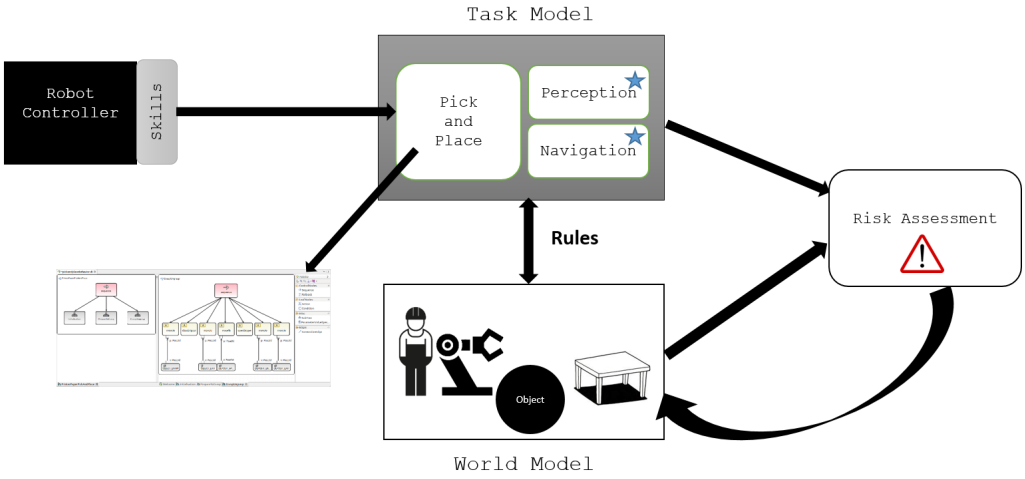

Figure 5.3: RobMoSys Models in CEA pilot

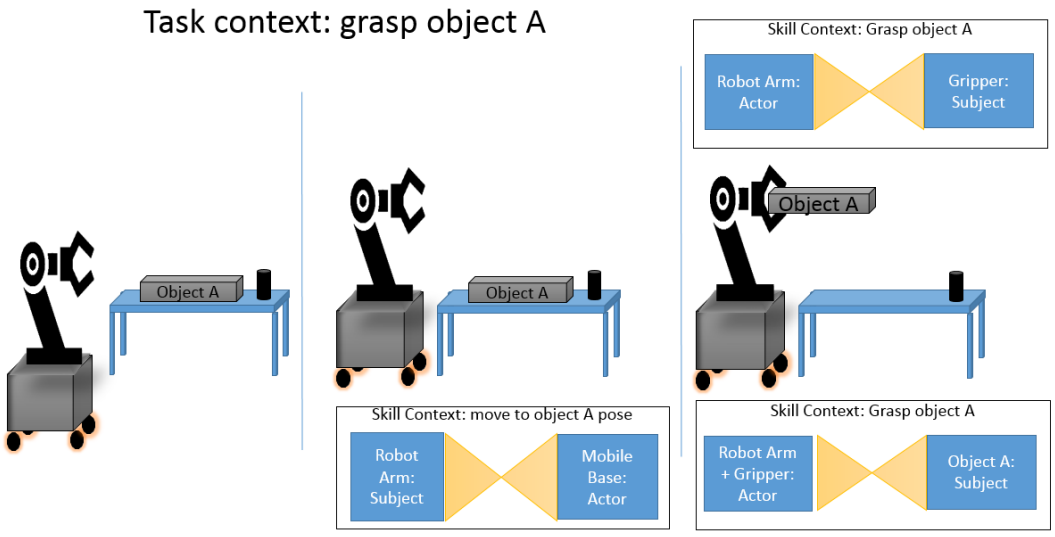

Figure 5.4: Context-aware world model

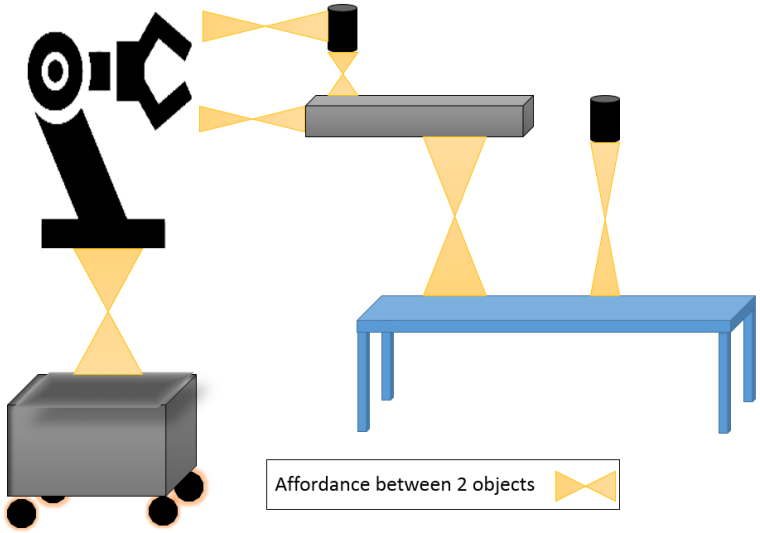

Figure 5.5: Context-aware world model

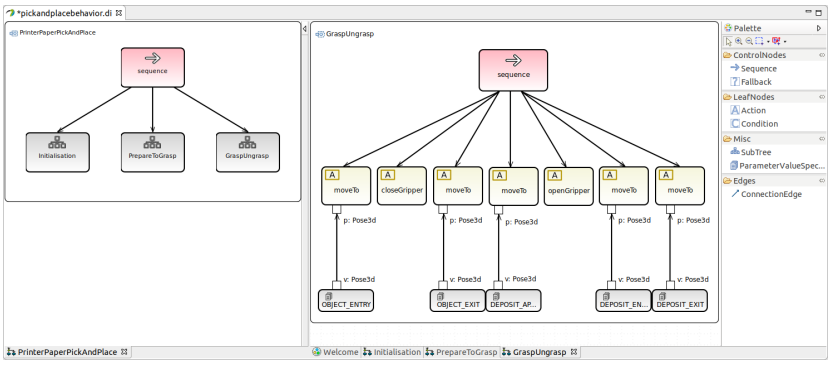

Figure 5.6: Pick and Place Behavior Tree

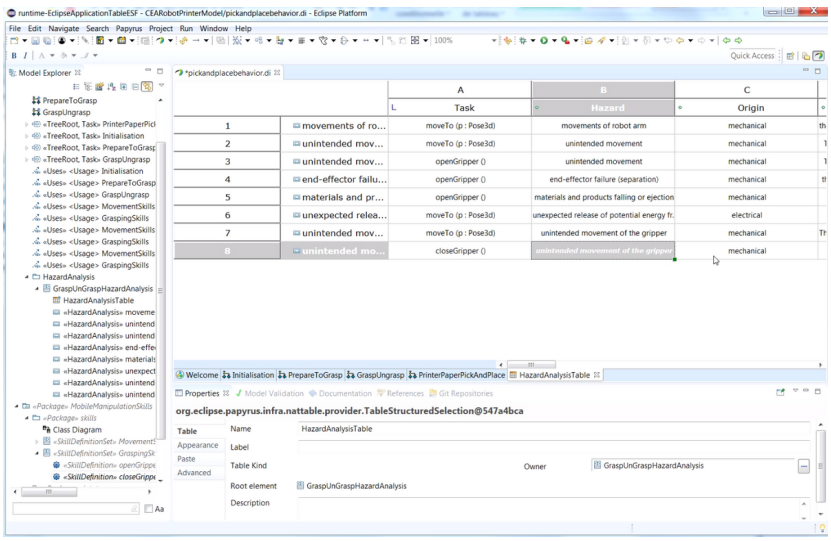

Figure 5.7: HARA table

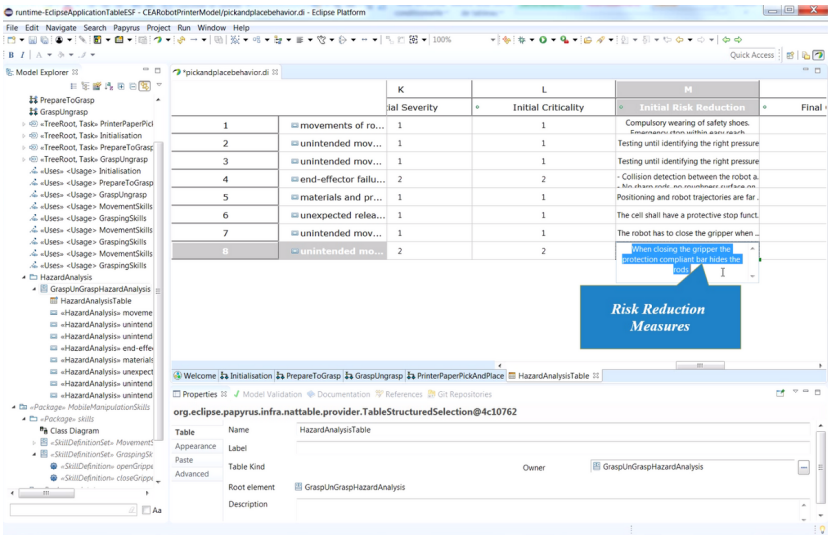

Figure 5.8: Risk Reduction Measures

6. Intralogistics Industry 4.0 Robot Fleet (HSU)

The Intralogistics Industry 4.0 Robot Fleet Pilot is about goods transport in a company, such as factory intra-logistics. It can be used to showcase robotics navigation, e.g. to show the performance of goods delivery and according non-functional requirements. It can be extended to object recognition and manipulation.

The Industry 4.0 Robot Fleet Pilot is not a single and specific application. Rather than that, all provided software assets contribute to the larger area of intralogistics robot fleets. The individual hardware and software assets available can be used to build a number of different systems and applications.

The pilot has been described in detail in the previous version of this deliverable, D4.1. See examples and videos of the pilot in action in the RobMoSys wiki[1].

Figure 6.1: Intralogistics Industry 4.0 Robot Fleet Pilot

6.1 Progress Summary

The pilot skeleton is available to use. The pilot is physically located at Ulm University of Applied Sciences (Germany) and may be used on site or remotely. An excerpt is available in simulation for off-site use.

The pilot skeleton consists of hardware (e.g. robot fleet, different platform vendors, camera, manipulator), software component models, skill models, system models, and other indirect/composed models. The models are fully conformant to RobMoSys and most of them come with executable software. There are few cases where the implementation of software components is becoming RobMoSys conformant because the implementation is under migration from a previous version.

[1] https://robmosys.eu/wiki/pilots:intralogistics

The pilot has been adopted by three out of six RobMoSys Integrated Technical Projects (ITP). The ITPs have used the pilot to showcase their results and benefits of RobMoSys. Thanks to the pilot being conformant to RobMoSys, the ITPs were able to showcase the benefit of their result in a complex scenario with no additional effort. The RobMoSys wiki documents showcases of ITP results using the industry 4.0 pilot[2]:

- Robotic Behavior in RobMoSys using Behavior Trees and the SmartMDSD Toolchain (MOOD2BE ITP)
- Dealing with Metrics on Non-Functional Properties in RobMoSys (RoQME ITP)
- Using the YARP Framework and the R1 robot with RobMoSys (CARVE ITP)

6.2 Pilot Scenario and Use Case

The concrete scenario of this pilot is about goods transport in a company, such as factory intralogistics. It features the delivery of a set of orders: a fleet of robots collaborate to deliver orders (fig. 6.2). A list of addressed user stories can be found in the previous Deliverable D4.1.
The pilot includes a set of different robots and stations that interact:

- stations to deliver boxes autonomously
- stations to pick items/goods
- robots to pickup, transport, and deliver boxes
- robots for mobile manipulation to do order picking of goods into boxes

(a) Picking items (b) Transport and autonomous delivery of boxes with goods (c) Order picking into boxes

Figure 6.2: Excerpts of the Intralogistics Industry 4.0 Robot Fleet Pilot.

6.3 Setup Description and Pilot Skeleton

Software components are available[3] for use with the SmartMDSD Toolchain for immediate composition:

[2] https://robmosys.eu/wiki/community:start
[3] https://robmosys.eu/wiki/model-directory:start

- Hardware Abstraction for several Robot platforms and other sensors/actuators
- Mapping
- Planning
- Obstacle avoidance
- Object Recognition
- Mobile Manipulation
- Software component templates for manipulation and object recognition

Documentation, tutorials, and screencasts are available for the SmartMDSD Toolchain[4] which this pilot is built with. The tutorials and screencasts show excerpts of the pilot for the purpose of education and learning.

6.4 Pilot Focus and Coverage of RobMoSys Features

The pilot is intended for open call 2 contributors to showcase the ease of system integration via composition of software components to a complete robotics application. The following are realized use-cases to illustrate the benefit of RobMoSys. More information on the following material present on the RobMoSys wiki.

- Software components and system composition: e.g. composition of previously developed software components and/or exchange of software components to address new needs.
  - see the excange of a hardware and software component demonstrated using the industry 4.0 pilot[5]
  - see how PAL Robotics modifies the pilot to use the software with its TIAGo robot[6]
- Ecosystem collaboration including the different roles that participants can take
  - see how the system builder and the behavior developer can interact[7]
- Mixed-Port Components
  - see how the CARVE ITP used the SmartMDSD Toolchain to develop software components to access components from the YARP middleware[8]
- Task level coordination; skills; robotic behavior
  - see Robotic Behavior in RobMoSys using Behavior Trees and the SmartMDSD Toolchain (MOOD2BE ITP)[9]

[4] https://wiki.servicerobotik-ulm.de/tutorials:start
[5] https://www.youtube.com/watch?v=e4uC0mEWxCk
[6] https://www.youtube.com/watch?v=FCvK9dAZXPo
[7] https://robmosys.eu/wiki/community:behavior-tree-demo:start
[8] https://robmosys.eu/wiki/community:yarp-with-robmosys:start
[9] https://robmosys.eu/wiki/community:behavior-tree-demo:start

  - see Support of Skills for Robotic Behavior[10][11]

- Managing of non-functional properties
  - see how the RoQME ITP monitors non-functional properties on this pilot[12]
  - see how the SmartMDSD Toolchain supports in non-functional properties showcased in this pilot[13]

- Dependency graphs for composed components to enable predictability for navigation

6.5 Technical Details

The pilot is fully supported by the SmartMDSD Toolchain[14], an Integrated Software Development Environment (IDE) for system composition in an robotics software business ecosystem. This ensures full conformance to the RobMoSys methodology when using this pilot.
The pilot is intended for the use with FESTO Robotino[15] but can be extended to any robot thanks to the Flexible Navigation Stack[16]. It covers navigation via the Flexible Navigation Stack and mobile manipulation using the Mobile Manipulation Stack.
At the core of the pilot is the flexible navigation stack[17] (fig. 6.3,6.4).
For more information, see the following resources:

- Pilot description in the RobMoSys Wiki[18]
- Flexible Navigation Stack[19]
- Software Components for use with the pilot[20]
- Skills for robotic behavior featuring the pilot[21]
- Tutorials for the SmartMDSD Toolchain featuring excerpts of the pilot[22]
- Tutorial for using software components of the pilot with the TIAGo robot base[23]

[10] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:skills:start
[11] https://robmosys.eu/wiki/composition:skills:start
[12] https://robmosys.eu/wiki/community:roqme-intralog-scenario:start
[13] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:cause-effect-chain:start
[14] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:start
[15] https://wiki.openrobotino.org/index.php?title=Main_Page
[16] https://robmosys.eu/wiki/domain_models:navigation-stack:start
[17] https://robmosys.eu/wiki/domain_models:navigation-stack:start
[18] https://robmosys.eu/wiki/pilots:intralogistics
[19] https://robmosys.eu/wiki/domain_models:navigation-stack:start
[20] https://robmosys.eu/wiki/model-directory:start
[21] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:skills:start
[22] https://wiki.servicerobotik-ulm.de/tutorials:start
[23] https://robmosys.eu/wiki/baseline:scenarios:tiago_smartsoft

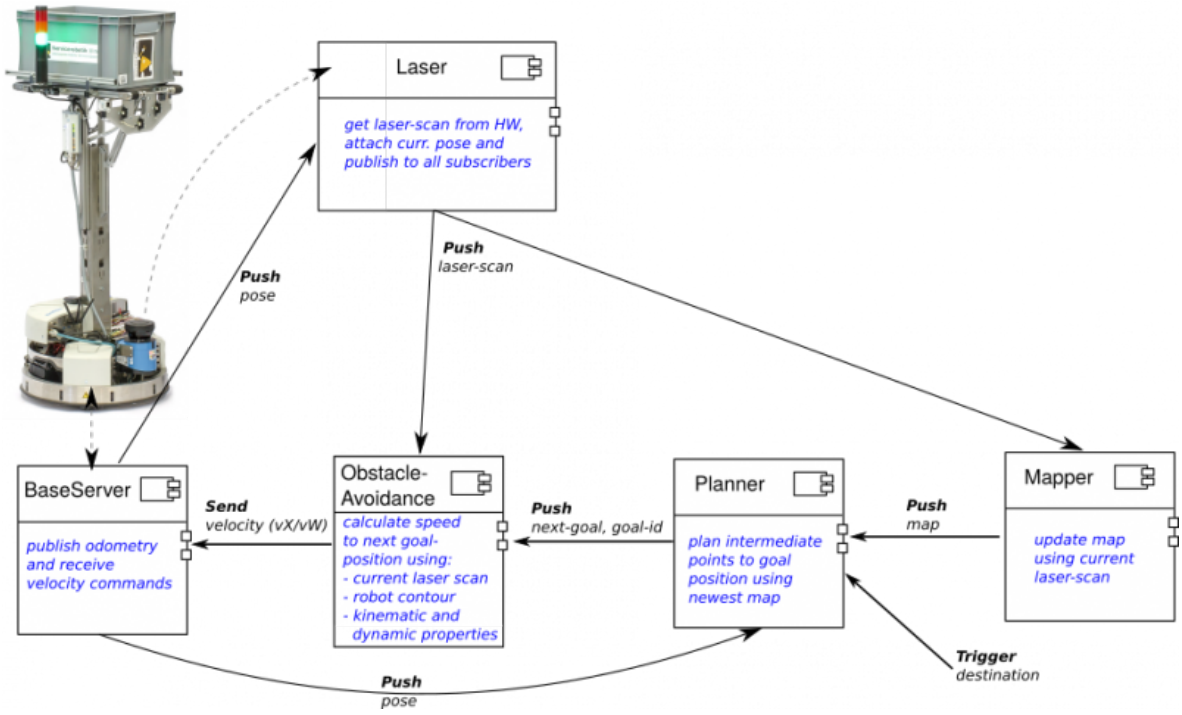

Figure 6.3: The flexible navigation stack used in the Intralogistics Industry 4.0 Robot Fleet Pilot

6.6 Key Performance Indicators

This pilot demonstrates the suitability of the RobMoSys composition structures for system composition.

- Goal: The pilot shall demonstrate the benefit of RobMoSys
  - Question: How flexible can systems be composed?
    - Metric: Number of functional systems built with assets provided by this pilot.
  - Question: How much of the RobMoSys methodology does this pilot demonstrate?
    - Metric: Coverage of composition structures: Percentage of RobMoSys metamodels that are used in pilot models.
    - Metric: Percentage of reusable development assets conformant to RobMoSys.
  - Question: How many user stories[24] have been demonstrated??
    - Metric: Number of user stories that have been demonstrated using the pilot scenario or its skeleton.
- Goal: The pilot shall provide a complex system to the RobMoSys community such that they can start with an already complex scenario by day one.
  - Question: Is the pilot used by the RobMoSys community?
    - Metric: System-level adoption: Number of ITPs using the pilot scenario

[24] https://robmosys.eu/wiki/general_principles:user_stories

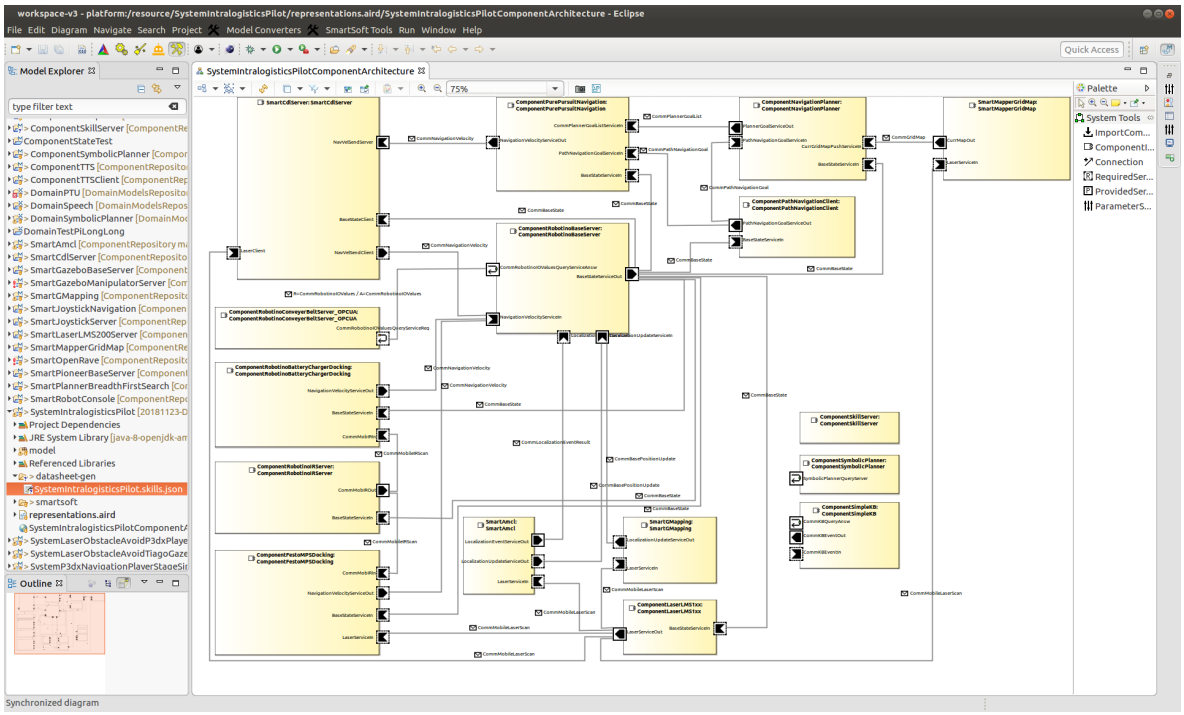

Figure 6.4: The system composition view of the SmartMIntralogistics Industry 4.0 Robot Fleet PilotDSD Toolchain for composing a pilot application

* Metric: Building-blocks-level adoption: Cases in which development assets or parts of the skeleton of this pilot are being used by external partners