RobMoSys

H2020-ICT-732410

RobMoSys

Composable Models and Software for Robotics Systems

Deliverable D4.3:

Final 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:	D4.3
Document title:	Final Report on Pilot Cases
Version:	1.0
Delivery date:	31 December, 2020
Nature:	Report (R)
Dissemination level:	Public (PU)
Editor:	Daniel Meyer-Delius (Siemens)
Authors:	Daniel Meyer-Delius (Siemens), Bernd Kast (Siemens), Vincent Dietrich (Siemens), Alfio Minissale (COMAU), Sergi Garcia (PAL), Alessandro Di Fava (PAL), Selma Kchir (CEA), Matteo Morelli (CEA), Christian Schlegel (THU)
Reviewer:	Marco Jahn (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 showcase the benefits of the RobMoSys approach, by applying RobMoSys methodology, models and software tools in concrete industry-relevant cases. Therefore, Pilot cases are an important instrument 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.

During the project, the core partners created 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 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 have a better overview on the features that each pilot covers a coverage matrix of the Pilot cases has been compiled. This matrix was also used as an instrument to guide the focus of the different Pilot applications. In an effort to assess the benefits of the RobMoSys approach in the context of the consortium Pilots, the so-called "Goal-Question-Metric (GQM)" approach was used. In this approach a set of goals is first defined. Then, for each goal, a set of questions is specified to asses the achievement of the goal. To answer these questions, a set of metrics is used. Once the goals, questions and metrics have been specified, a benchmarking plan was elaborated to asses the overall benefit of the approach in the context of each Pilot.

Interactions with the academic partners, the main developers of RobMoSys models and tools (i.e., the RobMoSys software baselines), was strong throughout the project. In this sense, the Pilot cases were the main *drivers* of the motion, perception and world model stacks, pushing concrete requirements and priorities on functionalities and the tools, which were necessary for the realization of each case. In particular, the world model was 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. In the same vein, Pilot partners have spent 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, defined in the first report on Pilot cases (Deliverable D4.1), in terms of models and meta-models to use.

Finally, Pilot cases served as reference and starting point for most of the Integrated Technical Projects (ITPs) in the second round of ITPs. Many ITPs build upon the Pilot cases and skeletons developed by the consortium, using, for example, models already available.

The most important result of the Pilot cases is that they clearly showcase the composability of models developed applying the RobMoSys approach. Many Pilot cases explicitly targeted this important aspect of system development from the beginning. But more convincing are the demonstrations of composability that took place spontaneously during the project. An example is the utilization of the Behavior.Tree.CPP framework, developed by one of the ITPs in the first round by several ITPs in the second round, and even by the Pilots of the core partners.

This Deliverable presents the final report 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 kinds of applications (and hence requirements), centered around the core applications of "navigation" and "(mobile) manipulation".

During the project, the core partners created 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 evolved together with the technical developments of the project. While some of Pilots follow the RobMoSys approach from the initial design and implementation, others 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 was 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. The coverage matrix was also used as an instrument to guide the focus of the different Pilot applications by helping to indentify white spots and areas of large overlap. The coverage matrix was continously updated during the run of the project as new technical features and results from the ITPs became available, and the focus of the Pilot cases changed.

- **Ecosystem Roles** such as **Component Supplier** and **System Builder** 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).

1.2 Key Performance Indicators and Benchmarking

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 #3 Software/Hardware component replacement	User story #1 Assistive robotic system composition	User story #2 Replacement of components	User story #1 Easy application design	User story #2 Easy end-effector design	User story #3 Easy web-integration	User story #1 Context-Aware Robustness for Pick and Place	User story #2 Task-Based Risk Assessment	User story #3 Context-aware Robustness for Assembly	User story #1
			10.5	10	13	10.5	10.5	13	7	13	18.5	9.5	17	20
Ecosystem Roles	Behaviour Developer	7												
	Component Supplier	6												
	Function Developer	4												
	Performance Designer	4												
	Safety Engineer	2												
	Service Designer	8.5												
	System Architect	6.5												
	System Builder	6												
Metamodels	Robotic Behaviour	6												
	Communication-Object	7.5												
	Communication-Pattern	7.5												
	Component-Definition	9.5												
	Deployment	3.5												
	Functional Architecture	1												
	Cause-Effect-Chain and its Analysis	1.5												
	Platform	1												
	System Service Architecture and Service Fulfillment	4												
	Service-Definition	5												
	System Component Architecture	3												
Robotic Domain Models	Motion	0.5												
	Perception	5												
	World Models	4.5												
	Flexible Navigation	2.5												
	Active Object Recognition	5.5												
	Digital Data Representation	4.5												
Composition	Task Composition	7												
	Service-based Composition	8.5												
	Composition of algorithms	2.5												
	Managing Cause-Effect Chains in Component Composition	2.5												
	Coordinating Activities and Life Cycle of Software Components	0.5												
Tools and Software Baseline	SmartMDSD	6												
	Papyrus for Robotics	4												
	Groot	1												
	BehaviorTree.CPP	4												

Table 1.1: Pilot Coverage Matrix. A value is assigned to each color code: Green = 1, Cyan = 0.6, Yellow = 0, White = 0, and the sums of these values are displayed for each row and column in the 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 the 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 elaborated assesing the overall benefit of the approach being evaluated.
In the following sections, the goals, questions, metrics, benchmarking plan and results for each individual Pilot are presented. An initial version of the benchmarking metrics and plans was presented in the previous report. This initial version was then reviewed, refined and finally implemented taking into account the benchmarking plans of all other Pilots, technical progress of the project and results of the ITPs.

1.2.1 Benchmarking Challenges

During the definition and implementation of the benchmarking plans, several practical difficulties became evident. The most common issues were the following:

- Most quantitative metrics found such as number of components or number of configuration parameters are better suited for assessing the resulting system and not so much for assessing the development process itself. This process is, however, where the focus of the RobMoSys approach primarily lies.
- Most metrics don't have an absolute reference value for interpretation. In these cases, assertions about the benefits of the RobMoSys approach can only be made by comparing the values against a reference or baseline value. During benchmarking, it was often difficult to obtain these reference values since this would in essence require developing the system twice: once following the RobMoSys approach and once following a different approach, keeping track, both times, of all the data required for computing the metrics. Developing two systems in parallel or the same system twice was not practicable. Some of the Pilot systems where either developed using the RobMoSys approach from the beginning or consisted of hybrid systems where only a component of the system was developed using the RobMoSys approach on top of existing components.
- In addition to the difficulties in obtaining reference values for the metrics, another pitfall faced was the choice of baseline approach to compare the RobMoSys approach against. For a fair comparison the reference approach must be selected appropriately, considering the aspects of the approaches that will be compared. The selection of the baseline approach is not only a theoretical challenge but also a practical one since for some of the Pilot applications the baseline system was fixed from the beginning.

Despite the weaknesses of the benchmarking strategy, its strength is that it helps in assessing the advantages of the RobMoSys approach in a focused way, by clearly stating the scope of the

[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

evaluation by clearly defined goals. It is of key importance to keep the context of the goal and the focus of the application in mind when interpreting the benchmarking results.
To further support the benchmarking results and in complement of the quantitative metrics, we include in this section a selection of testimonials about the RobMoSys approach collected through the execution of the project as a qualitative level metric. A more complete list of testimonials can be found in the RobMoSys webpage[4]:

- "The methodology of RobMoSys allows us to improve how we build and assemble systems with components. This goes beyond the limitations of most software middleware used in robotics, which gives little support to help integrators figure out how to combine components effectively.", Lorenzo di Natale, IIT
- "We are proud of being part of RobMoSys as you become part of a bigger ecosystem which can improve the impact of your work. Compared to other research formats and projects, RobMoSys is more demanding, but at the same time more effective.", Matteo Matteucci, Politecnico di Milano
- "The Intralogistics Industry 4.0 Robot Fleet Pilot is already conforming to RobMoSys and is supported by the SmartMDSD Toolchain. That allowed us to directly showcase our ITP contributions to RobMoSys in that complex and industry-relevant application with no additional effort.", Cristina Vicente-Chicote, Universidad de Extremadura
- "I am quite impressed with the maturity of the RobMoSys tools (here SmartMDSD Toolchain). I can recommend it to grasp the practical consequences of RobMoSys and apply the concepts in an effective way. Even in the first steps, you will understand the power of the approach and how you can gain from RobMoSys: It makes the composition of systems easier and I see the benefit of separation of roles. I feel at home with RobMoSys now.", Bouke Krom, TNO Netherlands Organization for Applied Scientific Research

[4] https://robmosys.eu/testimonials/

2. Flexible Assembly Cell (Siemens/TUM)

2.1 Summary

The primary focus of this pilot is the development of the basic functionality for a minimal application. This setup allows us to realize the "flexible assembly cell" scenarios, 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 functionalities 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 intra logistics and material transport is another fundamental requirement for a flexible production. Classical solutions such as conveyor systems cannot fulfill the increasing demand for flexibility. Constant reconfigurations of the shop floor and material flow are becoming 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 on top of a mobile platform. The system can use its sensors for navigation and does not require 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" systems. A large number of automation scenarios extend or closely interact with existing systems. In this so-called "brownfield" scenarios, 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.

Use cases

Two use cases are considered: 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

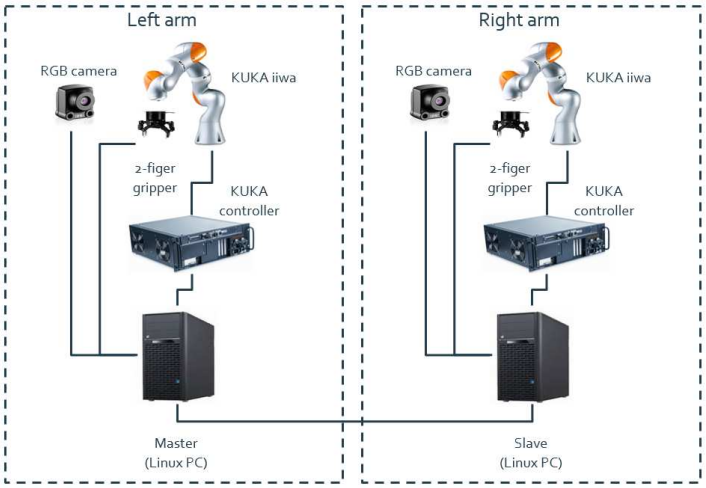


Figure 2.1: Hardware components of the flexible assembly cell setup: two advanced robotic arms, each equipped with a 2D camera for perception and a gripper for object manipulation.

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

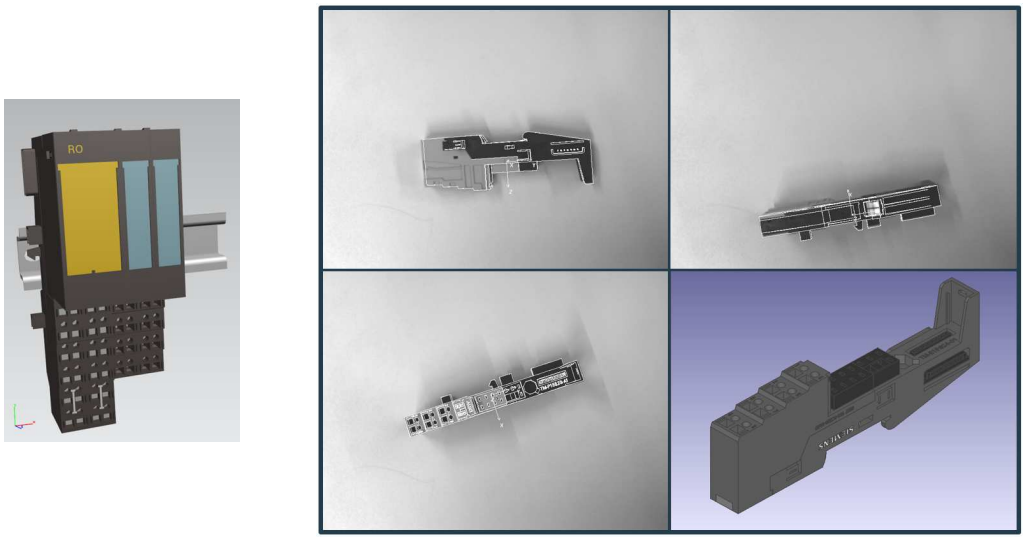


Figure 2.2: Exemplary flexible assembly cell task: mounting top-hat rail modules on a control cabinet. The left image depicts an example control cabinet with three different modules of two different types. The image group on the right shows the results of the object detection functionality.

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 as 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 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

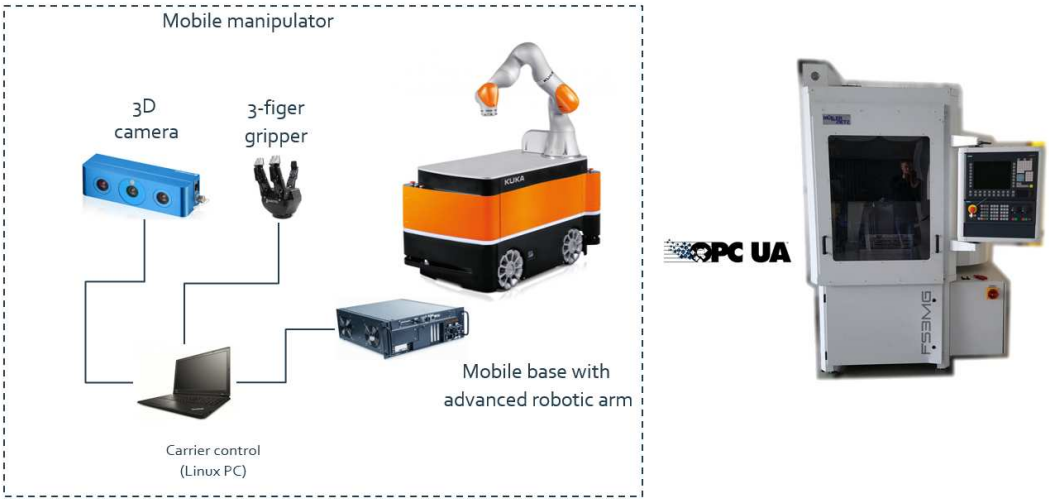


Figure 2.3: Hardware components of the machine tending cell setup: mobile based with an advanced robotic arm equipped with a 3D camera for perception and a gripper for object manipulation.

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

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 se-

lection of existing software components that realize it. These functionalities are accessible through *Skill Interfaces*. Skills are 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 *Domain Expert* defines the set of relevant skills in the form of *Skill Definitions* models;
- the *Component Supplier* provides the *Skill Realizations* that implement the previously defined Skill Definitions;
- 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 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 component's 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. In addition to a relatively large software baseline of existing RobMoSys-conform models and software components, the *SmartMDSD* 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 with Groot as a graphical frontend for a userfriendly definition of the execution logic.

2.5 Technical Details

In this section we discuss the technical details of our two demonstrators that highlight the different user stories. We first discuss a machine tending scenario, which highlights the importance of separation of concerns, the reduction of complexity with expressive models, explicit representation of parameters, and encapsulation within different modules.
Secondly, we highlight the flexibility of the model driven approach, discussing the replacement of a component within our existing system. This easy exchange of components not only facilitates experiments and the evaluation of new algorithms, but also enables even small groups to contribute to an open platform and gain access to large markets.

2.5.1 User story: Task-oriented programming

In todays industrial applications, the use of autonomous machines is oftentimes too complex to setup due to the large number of components to pick from and the difficulties to parametrize them. In this user story, we discuss the model driven approach of RobMoSys for the task-oriented programming, which encapsulates and models behaviors to facilitate the composition of new systems from existing modules.
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

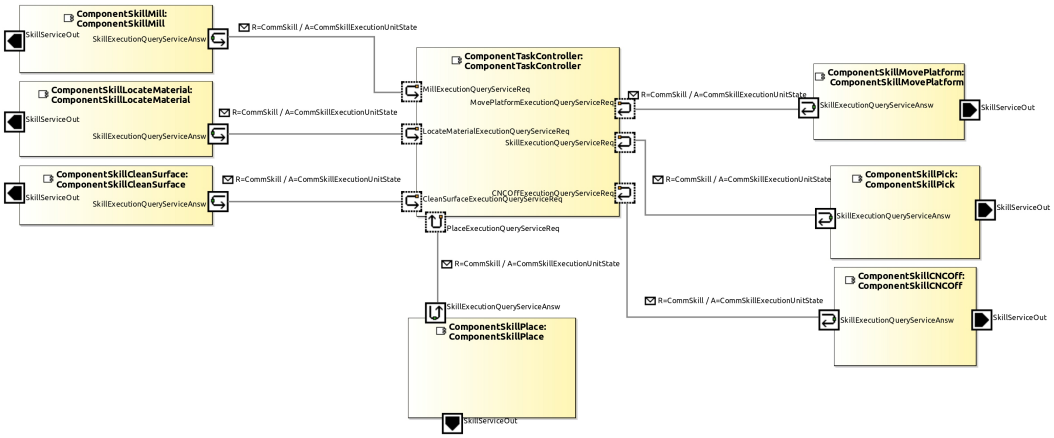


Figure 2.5: System component architecture diagram. The system consists largely of mixed-port components that provide access to the system functionality realized in ROS (legacy). An executive control component executes the application by orchestrating the other components.

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 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 components, acting as a bridge between the ROS and RobMoSys-based systems. The base system functionalities are grouped as skills, allowing both systems to interact on the same level of abstraction.

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

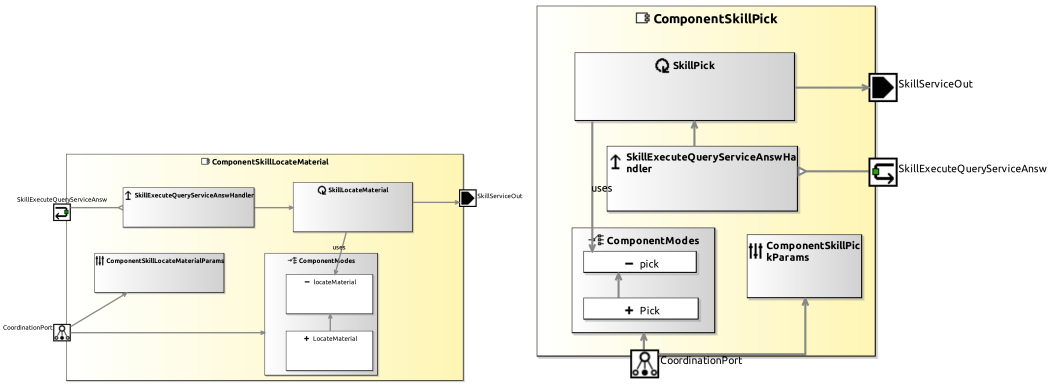


Figure 2.6: Structure of the `ComponentSkillLocateMaterial` and the `ComponentSkillPick` mixed-port components for bridging the ROS and RobMoSys systems.

is the `ComponentTaskController` component. This component triggers all other components with the correct parametrization and at the right time. Due to the standardized modeled interface, more advanced algorithms and user interfaces can be used. This eases the configuration of complex systems, for example by using a graphical editor to configure behavior trees Figure 2.7.

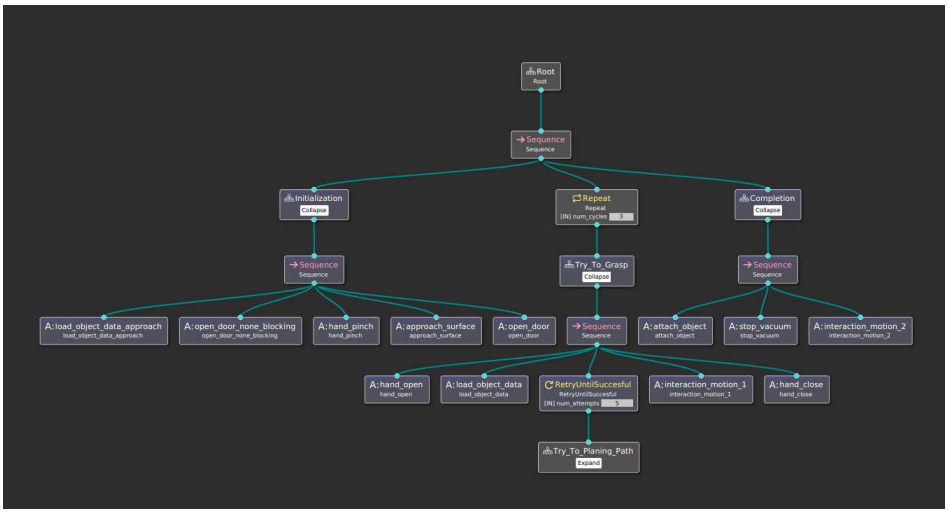


Figure 2.7: Graphical editor for the complex task specification using a behavior tree.

The approach for managing the world model information described above is just an 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 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.
Another scenario for this user-story, which highlights the general principle of composability within the RobMoSys approach can be found in our perception stack. With this scenario a typical human-robot-interaction situation is presented, where a worker conducts assembly operations with the help of a robot. The worker places finished items in a designated area for the robot to grasp them in a specific order. Following, the robot picks each item by using a smart object detection algorithm and performs inspection in order to check whether the item is correctly assembled. The robot places the correctly assembled item in a dedicated area, otherwise the robot returns the incorrectly assembled item back to the worker for additional repair. This scenario represents collaboration between two EU projects, RobMoSys and HORSE. RobMoSys' tool Papyrus4Robotics was used for the perception components, which were integrated in a modular way with the task planning module from HORSE. This allows easy composition of new scenarios using existing components

2.5.2 User story: Hardware and Software component replacement

In this user story, the replacement of components is required due to different requirements of the current task or limited availability of components.
The first scenario builds upon the perception pipeline example of our previous user-story. The RobMoSys approach allows the workers to exchange different perception implementations in order to adjust for increased accuracy depending on the situation and target items involved. Additionally, RobMoSys allows us for an easy hardware replacement, such that we can use of different grippers for specific item types. We demonstrate the ease of modifications in human robot collaborative workcell for flexible assembly processes. For different types of assembly items a set of two grippers were used together with the specific perception component. All the software and hardware components follow the approach of RobMoSys composability, allowing to the user to

easily adapt for the requirements of the assembly process.

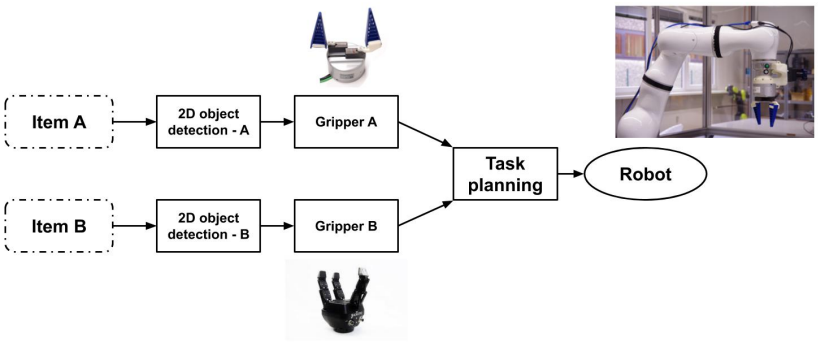


Figure 2.8: Software/Hardware component replacement schematic.

All the components are implemented using the RobMoSys tool Papyrus4Robotics and are available in the perception stack.
In the secend scenario we replace the controls of the robot because of the limited availability of one input method. The example application is teleoperation of the flexible assembly cell, as depicted in Figure 2.9. In the existing setup a joystick (Component A) is available and can be used to teleoperate the robot. Therefore, the commands of the joystick are converted into a set of motion constraints, which then are converted into robot axis commands. These conversions and computations are performed by the constraint control module. The robot axis commands are send to the robot controller and executed on the robot hardware.

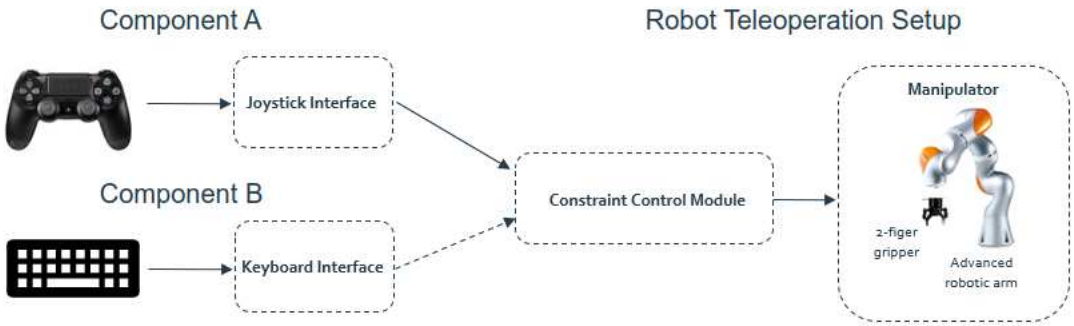


Figure 2.9: The target application for the hardware component exchange is teleoperation. Here, the robot is teleoperated using a joystick (Component A). When transferring to a different user without joystick, it is required to replace it with a keyboard (Component B).

A different user of the system does not have a joystick at hand and needs an alternative in order to be able to teleoperate the robot. Therefore, an exchange of the joystick with a keyboard (Component B) in combination with the associated driver software modules is required. Without clear interfaces and abstractions, this steps can only be performed by an expert, which can dive into the code of the overall system. In order to allow a non-code component exchange, the system is provided with models using the RobMoSys approach. As tooling, the RobMoSys conform

SmartMDSD is used.

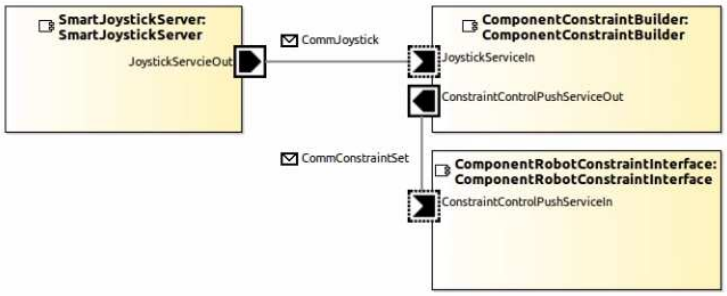


Figure 2.10: The intial SmartMDSD system model which represents the software view on the system. The joystick server provides target commands via CommJoystick messages, which are converted to robot commands in ComponentConstraintBuilder and ComponentRobotConstraintInterface.

In Figure 2.10 the model of the original system with joystick is displayed. Here the Constraint Control Module is again separated into two different software components, the ComponentConstraintBuilder and the ComponentRobotConstraintInterface. The former converts joystick commands into motion constraints. The latter converts these motion constraints into axis commands and communicates with the robot.
The joystick is connected via the ComponentKeyboardJoystick, which receives the commands from the joystick and converts them into messages of type CommJoystick. This component is already part of the SmartMDSD model library and can therefore easily be reused without implementation effort. The constraint control components and the keyboard interface were newly modeled for the application at hand. However, with existing code as in the presented example, this is associated with minor effort.

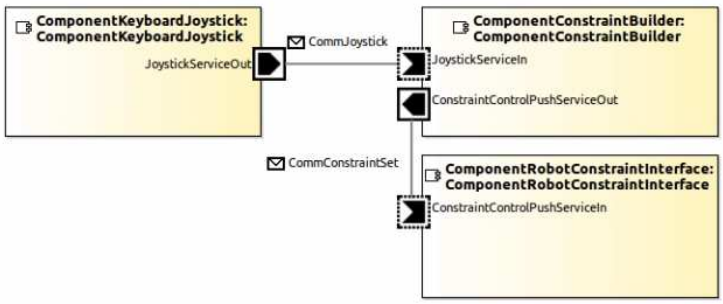


Figure 2.11: Here the new system model with the new ComponentKeyboardJoystick is displayed. Due to the reuse of model, e.g., of the the CommJoystick message, the system adaptation is easy to realize.

Now, with models readily available, the exchange of the joystick can be done by a system designer

(Tier 3) without touching the code of the system. Only the joystick component has to be removed and replaced with the keyboard component via the visual editing tool. The result is displayed in 2.11. It can be seen, that no other change was necessary apart from the component exchange at the system design level.

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 focus 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 promising 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* role.

- 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. debugging) but also for analyzing the general behavior of the system and its individual components. A related question for the user in the role 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. 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 for 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.

SIE-M5: Variation between functionally similar components

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

The same functionality ("object detection" functionality, for example) can be realized 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 Benchmark

The benchmarking results according to the Goal-Question-Answer-Metric approach are displayed in Table 2.1. We will first discuss the user story #1 for task-oriented programming and then proceed with user story #2 for hardware component replacement.
The user story #1 demonstrates the strengths of the RobMoSys approach for the composition of new systems on complex industrial tasks. Now, the application of autonomous systems becomes viable, as the unmanageable number of components and parameters is structured and reduced. With our metrics we can observe, how the separation of concerns, implemented with our modeling approach eases the setup of new scenarios. As stated in the last column of Table 2.1, the target of all scenarios was to reduce the qualitative number of the metrics as good as possible. For the first scenario, the baseline implementation of column *Prior* was improved for most metrics. However, some of the targets are conflicting. For example, less components result in aggregated modules, which potentially have more degrees of freedom to be parametrized. For sure, the goal is to not only shift the complexity by this bundling, but to actually reduce it. This is what we can observe in the first scenario: The total amount of parameters is reduced, however the average number slightly increases. Thus the complexity is reduced by the RobMoSys approach. It is, however, no silver bullet and slightly increases the complexity per component.
In the case of the user story #2, where the component exchange is targeted, only low numbers of components are available. Furthermore, configuration parameters and traceable properties have not been part of the user story. Therefore, the metrics do not fully reflect the advantages that the RobMoSys approach displayed in the use case. By employing the model-based software development approach it was possible to change the system design without a single line of code. Furthermore, there is a clear separation between the user roles and the interfaces they see. This reduces the overall engineering effort and prevents to some extent that system design decisions are taken from an unappropriate user role. Furthermore, it became clear that convenient engineering tools such as the SmartMDSD toolchain play an important role in proliferating model-based software and system design in domains such as robotics.
The interpretation of the benchmarking results is only valid withing a well-defined context. In this Pilot the focus is the reduction of application programming effort (`SIE-G1`) from the point of

Goal SIE-G1: Reduce application programming efforts						
Question	Metric	#1 Prior	#1 Current	#2 Prior	#2 Current	Target
SIE-Q1: How do I choose the right functionality?	SIE-M1: Number of functional components	23	4	3	3	Low
	SIE-M4: Number of different implementations of the same functionality	1	1	2	2	Low
	SIE-M5: Variation between functionally similar components	2	1	None	None	Low
SIE-Q2: How can I find out which functionality/component is not performing?	SIE-M2.1: Total number of configuration parameters	46	24	N/A	N/A	Low
	SIE-M2.2: Average number of configuration parameters per functional component	2	6	N/A	N/A	Low
	SIE-M3.1: Total number of traceable properties	138	48	N/A	N/A	Low
	SIE-M3.2: Average number of traceable properties	6	12	N/A	N/A	Low

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

view of a user in the *Behavior Developer* role. A reduction in the number of parameters doesn't mean that the parameters are eliminated, it means that the parameter is not visible or relevant for the user in the *Behavior Developer* role. These parameter just become then relevant for a user in a different role. Following the RobMoSys approach doesn't make the complexity of a problem disappear. The approach structures and formalizes the problem resolution methodology with the goal of reducing the overall effort through the whole development life-cycle is reduced. In this Pilot only a small aspect of the whole is being considered.

Difficulties

The benchmarking process is generally associated with difficulties in mapping the benefits of the model-based development approach to quantitative properties. The user story #2, for instance, can be better represented using qualitative statements, such as "no code is needed for the exchange of a component on system level". Furthermore, the engineering effort has to be set into relation with the modeling effort and know-how needed. There is only benefit, if the time saved by model-driven development is less than the effort needed in acquiring the know-how. However, it is hard to quantify the prior level of know-how of a developer and estimate a percentage of effort reduction with a statistically relevant number of users. One possible solution to this is to look into the overall number of users. Because a high number of users that independently decides in favor of a proposed development methodology means that the methodology provides benefit.

3. Healthcare Assistive Robot (PAL)

3.1 Progress Summary

During the period M37-M48 the main effort was devoted to the final development of the pilot use cases on top of the pilot skeleton developed in the past years. The Healthcare Assistive Robot scenarios were slightly redesigned and adjusted respecting the last pilot progress report, to better show the benefits of the RobMoSys approach. The role of Behaviour Developer was omitted due to some difficulties related to the maturity of the Behaviour tools and their use, but also to better focus on the healthcare scenario. Much work went into configuring the pilot to highlight the benefits of RobMoSys and extend the core functionalities of the TIAGo robot related to the healthcare environment under RobMoSys. A benchmarking plan was identified and generated to evaluate the objectives of the pilot.
An updated version of our docker container with our pilot skeleton and with the basic tools to work on the RobMoSys framework was released and made available for some ITPs like HRICAR to easily develop their components.

3.2 Pilot Scenario and Use Cases

This Pilot specifically focuses on robotic adaptation for healthcare scenarios. It showcases the development and programming of assistive mobile robots in dynamic and unstructured environments where the robot has to act, combining several capabilities such as mobility, perception, navigation and human-robot interaction. Following this goal, the integration and deployment of robots with various components and different configurations is a central element that involves assistive robots such PAL Robotics' TIAGo manipulator[1].
Creating assistive robots means not only combining various subsystems but also combining them into robots that can be adapted to follow the specific requirements of the environment in which the person lives, as in the examples shown in Figure 3.1. Generally any new hardware and / or new software from a third party has to be "tuned" and the integration process involves a considerable effort in terms of person hours and hardware acquisition process. The ability to "compose" and deploy a new platform by using already developed compliant HW/SW components shortens the time to market considerably.

Figure 3.1: Examples of TIAGo robot assisting an old man and an old lady respectively.

[1] https://pal-robotics.com/robots/tiago/

Scenario

The platform addressed in this Pilot is a TIAGo mobile base manipulator. Figure 3.2 depicts the TIAGo robot in a standard simulated apartment where two people are present. 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 tested in simulation and on the real robot. The fluorescent red color of the two people in the figure represents a modification made to the simulated person model options to allow thermal detection of the two people by the sensor.

Figure 3.2: Simulation of TIAGo robot in a standard apartment.

Use cases

Two use cases are considered: Assistive robotic system composition and Replacement of components.
In the first use case, the "user" mainly plays the role of the System Builder composing and manufacturing the mobile manipulator to be used in the apartment where the person needing assistance lives. In this use case the Component Supplier role is implicitly involved, to provide the components for the assistive robotic system. The demonstrator prepared consists of an application where the robot patrols the apartment, executes motions for human-robot interaction and detects people using different combinations of software and hardware components. As for example the component that uses the RGBD camera and the deep learning techniques to detect people and another component that is represented by the thermal camera to improve the detection accuracy.
In the second use case, the “user” plays the role of the System Builder that exchanges components with the same functionality. The demonstrator prepared consists of an application where the robot navigates in the apartment using the automatic navigation components that are then exchanged for the manual navigation components as is without the need to make additional changes.

3.3 Focus and Coverage of RobMoSys Features

The pilot aims to show the ease of building systems through the composition of software and hardware components to complete assistive robotic systems and the ease with which this approach

brings to replacement of components. This pilot focuses on the Service Based Composition of the RobMoSys approach and primarily covers the System Builder role. In the first use case the Component Supplier role is also implicitly involved, to provide the components for the assistive robotic system. The pilot covers some metamodels and robotic domain models related to all aspects of mobile assistance in a healthcare setting as indicated in the Pilot Coverage Matrix above (Table 1.1). The pilot provides valuable information on the accessibility and the usability of the RobMoSys technology through the SmartMDSD toolchain[2]. This Integrated Software Development Environment (IDE) for system composition in an enterprise robotics software ecosystem ensures full compliance with the RobMoSys methodology when using this pilot.
The pilot's motivation is that a new robot has been commissioned for a new apartment where the person that needs assistance lives. The System Builder wants to provide the best combination of components that meets the requirements for the specific apartment. This pilot is based on three different tiers for the composition of the system:

- Tier 1. Basic structures independent of the robotic domain. Mainly, the concept of composition of services, the concept of component, and the composition workflow.
- Tier 2. Defines the structures within service robotics. In this scenario, the people recognition, the localization and the mapping are used.
- Tier 3. This is where the PAL contribution is made. Specific services that provide recognition of people, arm movements and robot patrol in an apartment. Here, as a Component Supplier, the user is in charge of providing the software and hardware components to meet the requirements, while as a System Builder, the user selects various components that perform the necessary services and combines the basic components.

The RobMoSys composition relies on the Component-Definition and the System Component Architecture metamodels. Service definitions act as link between roles and activities. The key aspect here is that the services decouple interactions so it is possible to compose components free of interference. The TIAGo healthcare application depends on several Robotic Domain Models. In our particular scenario, the appropriate domains consist of the Perception and Flexible Navigation domains. However, in this pilot the existing domains were not extended.
A bridge to the TIAGo base was implemented in order to access the low-level functionalities of the RobMoSys tools. Where possible, the bridges were replaced with the RobMoSys conformant ROS Mixed Port to communicate with the existing legacy TIAGo robot system which is 100% ROS compliant. The pilot is open to the possibility of adding the role of Performance Designer to add the management of non-functional properties analysis and to the role of Behaviour Developer to advance with the coordination of tasks, selecting and composing the skills necessary for the task.

3.4 Technical Details

The platform involved in the pilot is a TIAGo robot equipped with a mobile base, a lifting torso, a head and an articulated arm with a dexterous hand, 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 humanoid hand, a PAL parallel gripper or a third party gripper. The torso includes a prismatic column, a pan-tilt head with a RGB-D camera and a thermal camera mounted on top

[2] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:start

of the head. TIAGo features the following characteristics: 70 kg weight, 15 degrees of freedom, 110 up to 140 cm of height, a footprint of 54 cm of diameter. The mobile base has a differential drive with max speed of 1 m/s and operates in indoor environments. The laser range-finder on the mobile base of TIAGo has 3 versions: 5.6 m, 10 m and 25 m range. Several human robot interfaces are integrated like a speaker and a stereo microphone. Also a GPU can be attached to the robot if needed to enhance the computational power.

From the control software point of view, PAL provides several off-the-self functionalities. The available controllers allow to command the wheels in velocity, the head and torso in position and the arm in position and effort mode (sensorless torque control). If provided with the force/torque sensor it can be controlled with admittance control as well.

For the navigation stack, TIAGo provides a path planning with self-collision avoidance, mapping and self-localisation. From the Human-Robot Interaction point of view there are several functionalities available such as arm and gestures motions creation, people and face recognition.

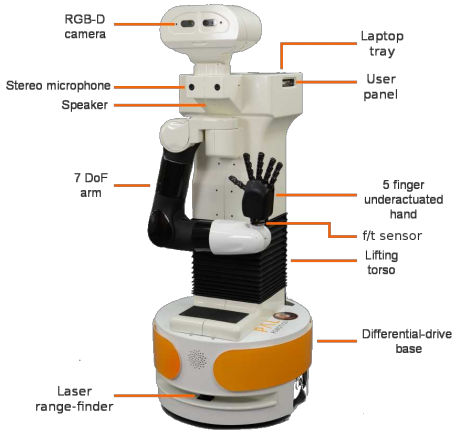


Figure 3.3: TIAGo main components.

Application development and deployment can be easily done from a standalone Docker virtual machine built specifically by PAL that contains everything needed:

- Ubuntu Xenial 16.04
- Gazebo 7 simulator
- SmartMDSD toolchain updated to v3.12
- ROS Kinetic
- PAL TIAGo packages and configurations
- machine learning libraries and datasets (TensorFlow models)

In the following sub-sections relevant technical details about the use cases of the healthcare pilot development are presented.

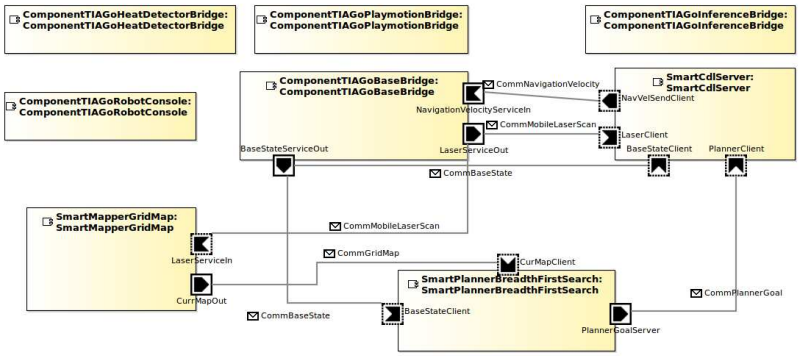


Figure 3.4: System Component Architecture view for the first use case.

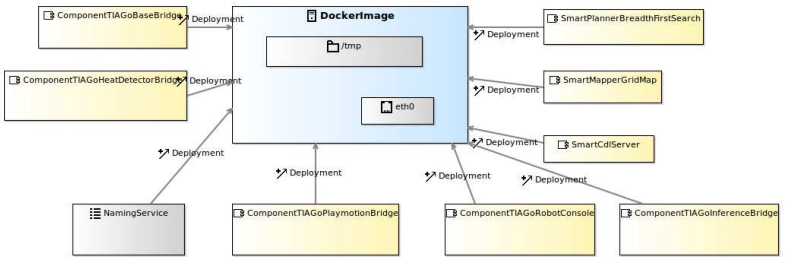


Figure 3.5: System Deployment view for the first use case.

Use case 1: Assistive robotic system composition

Figures 3.4 and 3.5 depict the System Component Architecture and the System Deployment diagrams, respectively, corresponding to use case 1, as modelled in the SmartMDSD toolchain. In the first use case PAL has developed several components that model the bridge with the legacy system:

- RGB-D camera + Deep learning people detection
- Thermal camera + OpenCV people detection
- Motions and gestures creation specifying the arm joints goals to be reached

To speed up development time, components are bridges made by PAL like ROS interfaces, to access existing legacy code for the TIAGo robot. It also shows how RobMoSys works finely with legacy code that has been previously implemented, even if the benefits of RobMoSys are diminished by the capabilities and design of existing code. However, some components can be replaced with native RobMoSys components as in our architecture where we are using RobMoSys flexible navigation components[3], replacing our navigation stack that is developed in ROS within the TIAGo framework. The ComponentTIAGoBaseBridge allows communication with the TIAGo base to obtain the information from the laser sensor, the base position and the velocity and send the velocity commands to the robot.

[3] https://robmosys.eu/wiki-sn-01/domain_models:navigation-stack:start

These bridge 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.
In the current structure, the management of the world model data is carried out within the ROS system. As, for example, the association of the people 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[4], the tf ROS package and the controllers developed in TIAGo with the ros control package[5].
Summarizing, these are the functionalities developed for the use case:

- ComponentTIAGoBaseBridge 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.
- ComponentTIAGoInferenceBridge does the people detection given a compressed image, using the deep learning algorithms based on the TensorFlow models, as shown in the Figure 3.6.
- ComponentTIAGoHeatDetectorBridge does the people detection using the image from the thermal camera and the OpenCV algorithms. The output of the people detection algorithm indicates the color of the temperature gradient of the person detected, as shown in the Figure 3.6.
- ComponentTIAGoRobotConsole calls the coordination interfaces to run the navigation demo, play motion demo and people detection demos.
- ComponentTIAGoPlaymotionBridge executes the motions expressed inside the robot by joints configurations. It can be used, for example, to orientate the TIAGo head to detect people or to greet the person with an arm motion.

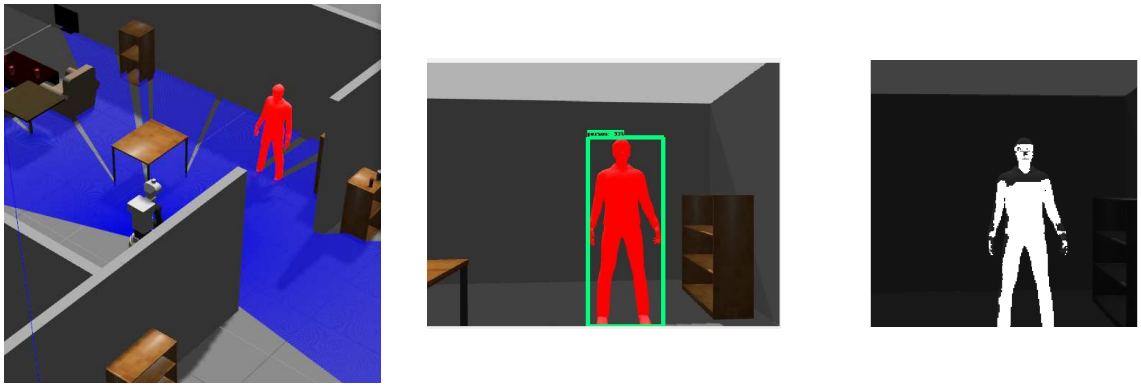
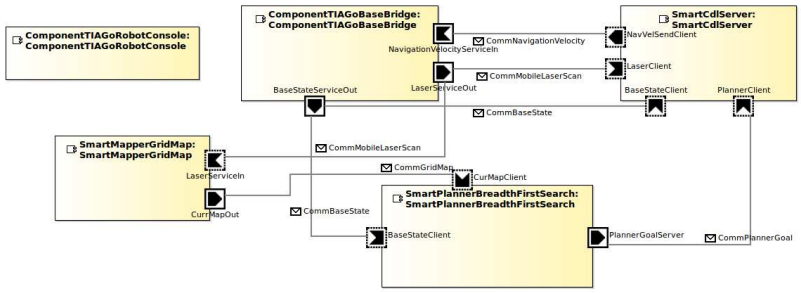


Figure 3.6: Simulation example of people detection with the deep learning component (in the center) and the thermal detection component (on the right), respectively.

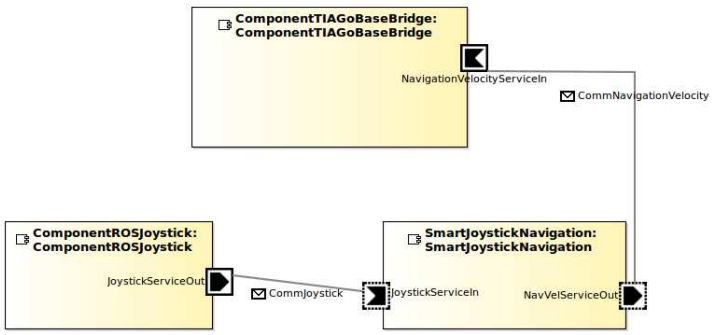
[4] http://wiki.ros.org/play_motion
[5] http://wiki.ros.org/ros_control

Use case 2: Replacement of components

The demonstrator prepared for the second use case consists of an application where the robot navigates in the apartment using the RobMoSys flexible navigation stack, already introduced in the first use case, for autonomous navigation. In the following scenario there is the need to manual navigate the robot in the apartment for different purposes, for example for the map creation or to make the robot reach a particular location in the apartment. Then the autonomous navigation components are exchanged by the manual navigation components as is without the need to make complex adaptations of the configuration and the composition of the robot by the System Builder. Figure 3.7 shows the System Component Architecture diagrams for the two scenarios, the autonomous navigation and the manual navigation respectively, as modelled in the SmartMDSD toolchain.



(a) Autonomous navigation



(b) Manual navigation

Figure 3.7: System Component Architecture views for the two scenarios of the second use case.

The ComponentTIAGoBaseBridge, also used in the first use case, allows communication with the TIAGo base to obtain the information from the laser sensor, the position and velocity of the base, and send the velocity commands to the robot. This component is used in both the autonomous and manual scenarios, while the navigation components are easily interchanged. In manual navigation, only the input is used to send the robot velocity commands, the other input and output ports are hidden in the diagram. For the autonomous navigation the ComponentTIAGoRobotConsole is also present, to call the coordination interface of the autonomous navigation components to allow the robot to patrol the apartment through a certain number of previously defined waypoints.

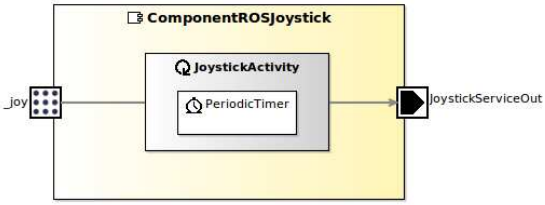


Figure 3.8: Internal view of the ROS Mixed Port component for the joystick.

In the manual navigation, the RobMoSys conformant joystick navigation component uses the ROS Mixed Port component to communicate with the physical joystick that is attached to the robot and that is made accessible by a topic in the ROS legacy system of the robot. Figure 3.8 shows the internal view of the ROS Mixed Port component for the joystick. The port on the left named _joy is the communication point with the ROS ecosystem to access the topic to get the values of the joystick.

This use case was tested both in simulation and on the real robot, even if the real robot tests were executed with a shared ROS master communication between the robot and the computer running the SmartMDSD toolchain due to the difficulties to deploy the RobMoSys framework inside the robot. The difficulties mainly are due to the current status of the SmartMDSD toolchain where the process of deployment of the components inside the real robot is still not well defined, future versions of the toolchain will work in this direction. Figure 3.9 shows two snapshots taken from the execution of the tests in simulation and in the real robot at the PAL office.

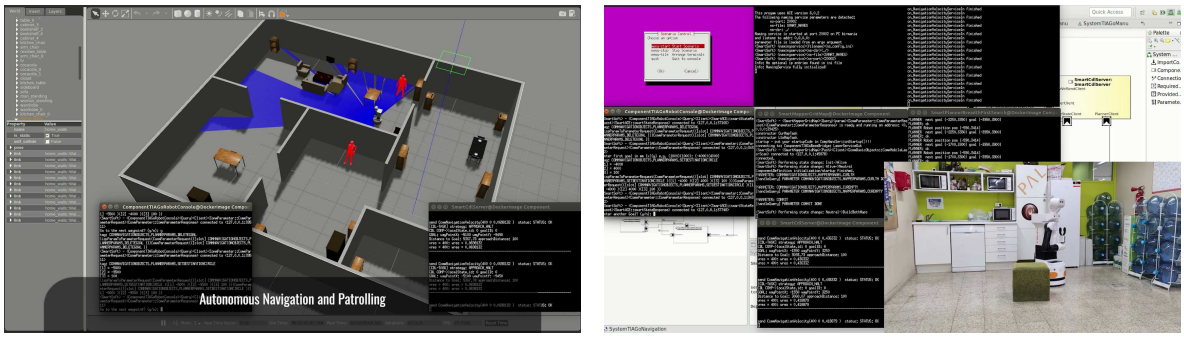


Figure 3.9: Snapshots taken from the execution of the tests in simulation and in the real robot at the PAL office.

3.5 Key Performance Indicators and Benchmarking

The performance indicators for the first use case will measure the effort reduction for the composition and manufacturing of an assistive robotic system that fulfils the functional requirements. This gain is especially highlighted in the second use case where the productivity is measured using the RobMoSys approach for components replacement.

3.5.1 Goals

The overall goal for the TIAGo healthcare Pilot is:

PAL-G1 To demonstrate that a component in a robotic system can be replaced by another component as-is by increasing the interoperability and the composability of the system functionalities.

The reduction of deployment and in general system engineering efforts is, for PAL, a big expected benefit of the RobMoSys approach due to the fact that we have different robotic platforms with different sensors, hardware devices and different software modules that can be used. This high variability in building the systems leads to a huge effort every time that a component has to be replaced by another one.

3.5.2 Questions

The questions are used to characterise the way in which a goal can be assessed, and correspond to issues that are expected to have major impacts on the achievement of the goal presented in the previous section. In the context of this Pilot, the questions are formulated from the point of view of the user in the System Builder role, where the Component Supplier is implicitly involved to provide the components for the assistive robotic system.

- Questions related to `PAL-G1: Components replacement & system composability`

  PAL-Q1 How can the effort for deployment of an assistive robotic system based on a new component be reduced?
  One important and time-consuming step during the deployment of a robotic system is putting together the appropriate components needed for realizing the desired robot duties. Enabling and supporting the user in the System Building to integrate the required modules, provided by the Component Supplier, would lead to a reduction of the overall deployment effort. Related questions would be, for example: How can the effort in designing an assistive robotic system supporting different configurations be reduced?

  PAL-Q2 How can the use of mixed-ports reduce the effort to port an existing system inside the model driven architecture?
  Another important, and also critical step in selecting a new framework of robot deployment is how to port the existing modules and legacy code avoiding to rewrite everything from scratch. This takes into account the benefits of having a framework based on a model-driven architecture like RobMoSys, but trying to reuse as much legacy code as possible, especially at the beginning when the framework is new and few components are available.

3.5.3 Metrics

This section describes the metrics used for answering the questions posed in the previous section. The formulas are intended to better explain what the metric is trying to measure. As with many metrics, without a reference value, they are only to be understood as an indicator. 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 with the complexity of a system.

PAL-M1: Number of RobMoSys functional components

$$M_{\text{PAL-m1}} = \text{Number of available RobMoSys functional components} \tag{3.1}$$

The number of functional components to choose from the RobMoSys framework at the moment of deploying a robot system can be used as an indicator of the associated deployment 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 systems. However, even with the support of such tools, the larger the number of components provided by the Component Supplier, the more numerous the number of choices the user has in the System Builder role.
This metric can be used as an indicator of the associated deployment effort when to port an existing system too. The larger the number of implementations of the several functionalities existing in the RobMoSys framework less effort has to be done to create mixed-ports to reuse the legacy code.

PAL-M2: Number of configuration parameters

$$M_{\text{PAL-m2}} = \text{Average number of configuration parameters per component} \tag{3.2}$$

Similar to the number of functional components $M_{\text{PAL-m1}}$, the number of configuration parameters relevant for the components to build a robotic system can be used as an indicator of the associated programming effort. Each component has its own set of configuration parameters. The larger the number of configuration parameters, the larger the effort related to correctly parameterizing the system.

PAL-M3: Variation between functionally similar components

$$M_{\text{PAL-m3}} = \sum \text{Different configuration parameters} + \sum \text{Different inputs and outputs} \tag{3.3}$$

The same functionality ("navigation" functionality, for example) can be realized by different components. If the configuration parameters and inputs and outputs of the components were identical, the user in the System Builder 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.

PAL-M4: Capability to apply different roles in a transparent way for users

$$M_{\text{PAL-m4}} = \frac{\sum \text{Different configuration parameters} + \sum \text{Different inputs and outputs}}{\text{Number of functional components}} \tag{3.4}$$

This is a measure of the separation of the System Builder role. How many configuration parameters and inputs and outputs the System Builder has to set with respect to the number of functional components available in the system, without knowing the details of the implementation of each component. The larger the number of settings for the components, the more the System Builder is involved in the details of the component implementation, which is the role of the Component Supplier.

PAL-M5: Number of functional modules existing in the legacy code

$$M_{\text{PAL-m5}} = \text{Number of available functional modules existing in the legacy code} \tag{3.5}$$

The number of functional modules available in the legacy code at the moment of porting a robot system deployment to the RobMoSys model driven architecture can be used as an indicator of the associated effort. The larger the number of functional modules existing, the larger is the effort to port them in the new framework. This highlights the importance of appropriate tools like the mixed-ports component for automating the composition and validation of systems communicating between different frameworks. The easier it will be for the user in the System Builder role to deploy a robot system without starting everything from scratch but reusing the legacy code as much as possible.

PAL-M6: Number of steps needed for a mixed-port to reuse legacy code

$$\begin{aligned} M_{\text{PAL-m6}} = &\text{Number of configuration parameters needed for a mixed-port} + \\ &\text{Number of code lines needed to translate the messages between the two framework} \end{aligned} \tag{3.6}$$

The number of configuration parameters relevant for the use of a mixed-port that is needed to reuse legacy code existing in another framework can be used as an indicator of the associated deploying effort in the new model driven framework. The larger the number of configuration parameters and lines of code needed to translate the messages and the communication port logic between the two frameworks, the larger the effort related to correctly reuse as much as possible the legacy code.

3.5.4 Benchmark

The table 3.1 presents the final version of the benchmarking plan to assess the overall benefits of the RobMoSys approach in the context of this Pilot. The Prior column represents the status for our robotics platforms in the beginning of the RobMoSys project. The Current column represents the KPIs that we actually achieved in the Pilot and the Target column represents the target goals that we wanted to reach with the RobMoSys approach. The assessment is done in a qualitative way using three levels: Low, Medium, High, and a combination of them where needed. The reasons behind it are related to the benchmarking challenges described in the next sub-section and also to the fact that it is difficult to come up with a number when we are grouping together components of different functionalities in the metrics and also when we are measuring qualitative aspects like the variation and the capability of a component.
As shown in the table, the RobMoSys approach brings a lot of benefits in this Pilot, especially for the interoperability and the composability of a robotic system starting from several components or replacing existing components to address new configurations. Some weaknesses can be found in the deployment of the system inside the real robot and in the availability of components for the several functions that an assistive robot like TIAGo encounters in a healthcare environment. While for the first issue it is just a matter of maturity of the tools, the second issue will be addressed the more the RobMoSys community get bigger. Such as, with the increase of component developers, more components will be available, avoiding the porting of any legacy code from the existing framework and facilitating the reuse of existing components.

Goal PAL-G1: Components replacement & system composability				
Question	**Metric**	**Prior**	**Target**	**Current**
PAL-Q1	PAL-M1: Number of RobMoSys functional components	None	High	Medium-Low
	PAL-M2: Number of configuration parameters	High	Low	Low
	PAL-M3: Variation between functionally similar components	Medium	Low	Low
	PAL-M4: Capability to apply different roles in a transparent way for users	Low	High	High
PAL-Q2	PAL-M5: Number of functional modules existing in the legacy code	High	Low	High
	PAL-M1: Number of RobMoSys functional components	None	High	Medium-Low
	PAL-M6: Number of steps needed for a mixed-port to reuse legacy code	High	Low	Medium-Low

Table 3.1: Benchmarking plan for goal PAL-G1: Components replacement & system composability.

Benchmarking Challenges

The benchmarking process is generally associated with challenges and difficulties in mapping the benefits of the model-based development approach to quantitative properties. It is quite difficult to come up with expressive key performance indicators for this pilot because we want to measure whether RobMoSys processes lead to better system in terms of effort reduction and productivity that are important key factors for a System Builder like PAL Robotics. Many of the achievements are at a qualitative level more than a quantitative one and they are about whether introducing a model-driven approach to shape a robotics ecosystem based on separation of roles and composition has been populated with convincing showcases. However, it is hard to quantify the prior level of know-how of a developer and estimate a percentage of effort reduction with statistically relevant numbers. One possible solution to this is to look into the overall number of components and roles transparency. Because a high number of them means that the methodology provides benefit.

4. Modular Educational Robot (COMAU)

4.1 Progress Summary

The main focus of this pilot is the development of the setup for a basic educational application through the integration of RobMoSys components and showing the benefits of its overall approach. In particular the pilot is addressed 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 with respect to standard methodologies. During the period M37-M48 the main effort focused on the final implementation of the pilot use case accordingly to of the pilot skeleton defined during the project.

4.2 Pilot Scenario and Use Cases

Scenario

The fast growth of the robotics market in the last decade has been able to increase its popularity also outside of industrial environments, acquiring notoriety within other markets, such as architectural installations and as tools for educational purposes. The availability of low-cost hardware platforms and SW platforms allowed the diffusion within industrial and manufacturing fields able to create interest, culture, and competence in a very fast-growing market. Taking into consideration the educational market can be strategic not only from a prospective point of view but also for the most variety kind of approach it can suggest. Assembly, mechanic, electronic, Information Technology are only few of the possible fields to be taken into consideration. Most important applications where robotics can be applied into educational environments are pedagogic and technical. The first one uses interaction with very simple robots to develop cognitive attitudes. The second one more focused to develop competence in HW and SW. Based on the second point we adapt the development of a modular and scalable platform mainly dedicated for educational scopes starting from the basic to more advanced applications.

Use cases

The pilot case is based on the open architecture of e.DO platform, a new robot developed for educational purpose that uses a ROS node to connect the Smartsoft environment with the robotics framework.
Different uses-cases have been taken into account Developing customized software functionalities 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 algorithms

4.3 Focus and Coverage of RobMoSys Features

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 enables teachers

Figure 4.1: e.DO platform

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 a user interface

In particular, in the pilot A communication trough SmartMDSD Toolchain has been established between an external programing device (such as joystick or inertial sensors) and the e.DO robot

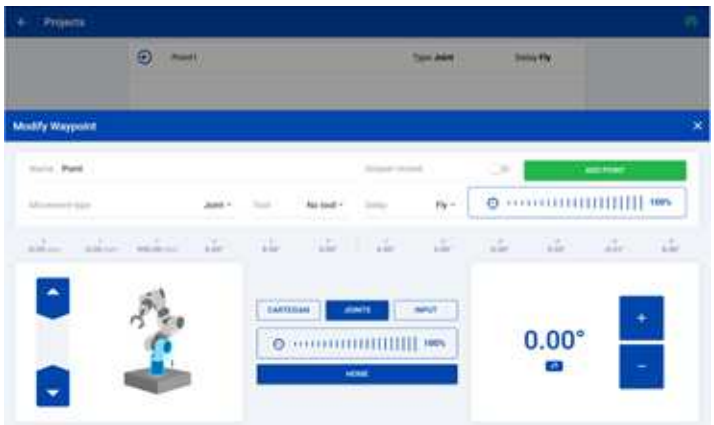


Figure 4.2: User interface for the e.DO platform

permitting easy and fast programming features as well as an high level of reconfigurability of the system. A specific component called edotranslater manages the communication with a client server protocol among e.DO and input device.

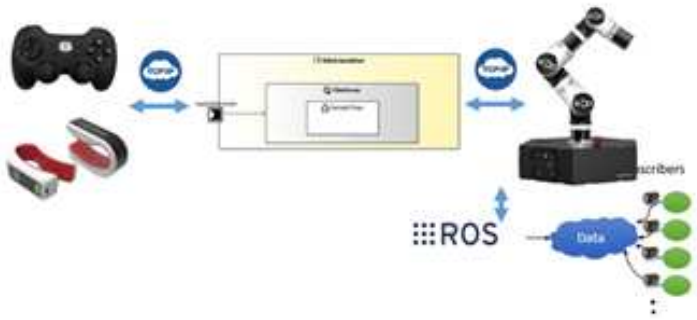


Figure 4.3: System architecture of the Pilot application

4.4 Technical Details

In the pilot the e.DO platform is used: it is a modular manipulator with a payload of 1kg, and composed by 6 motorized axes. The modular design enables the implementation of alternative kinematic structures with the same motorized units. The main control unit of the e.DO is composed by a Raspberry Pi3 mother-board. ROS melodic, GAzebo 7 and SmartMDSD toolchain are used in the pilot as well as OpenCV function for detection tasks.

4.5 Key Performance Indicators and Benchmarking

A set of key performance indicators to evaluate the benefits of the RobMoSys approach in the context of this Pilot have been defined. The main focus of this Pilot is on task-level programming, and the related KPIs are associated to programming level.

4.5.1 Goals

The overall goal for the Modular Educational Pilot is:

COM-G1 Increase the robot programming ease of use and improve the flexibility of e.DO platform by model-driven approach

The reduction of efforts needed for programming and setup of the system is for COMAU, a very important benefits of the RobMoSys approach. This implies a reduction of overall cost of installation of robotics systems.

4.5.2 Questions

The following questions have been formulated in order to evaluate 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

- Questions related to goal `COM-G1: Increase the robot programming ease of use and improve the flexibility of the system`

  COM-Q1 How could a model driven approach by task composition reduce programming efforts?

  The possibility to select the appropriate functionalities to achieve the specific task would be an advantage for the user in terms of time consumption and efforts on programming step. The system would be able to support the user on the task composition phase allowing the definition trough the model driven approach of a set of functionality available on the system

  COM-Q2 How can flexibility using mixed port decrease set-up effort?

  The model driven architecture and the use of mixed port feature will enable a flexible and simple porting of already existing code and modules on a new robot system allowing the user to reuse available resources and reduce the overall set-up effort of the system.

4.5.3 Metrics

COM-M1: Effort of programming and training phase

$$M_{\text{COM-m1}} = \text{Time effort taken on the programming and training phase} \tag{4.1}$$

Time effort on depends and the user profile but assuming that the pilot is addressed mainly to not skilled users the minimal amount of time for programming and training phase will be evaluated considering a comparison with standard programming modality

COM-M2: Number of functional components available and reused

$$M_{\text{COM-m2}} = \text{Number of functional components available and reused} \tag{4.2}$$

This metric indicates how many components could be available and could be reused in different applications and tasks thanks to the use of RobMoSys approach

COM-M3: Configuration steps needed for a mixed-port

$$M_{\text{COM-m3}} = \text{Number of configuration steps needed for a mixed-port} \tag{4.3}$$

The number of configuration steps and lines of code needed for the use of a mixed-port translating messages between different framework can be used as an indicator of the effort for the deployment of a new model driven.

4.5.4 Benchmark

The pilot is able to demonstrate the advantage of Robmosys approach in the scenario of educational platform on task-level programming. The metrics show an overall reduction of time efforts during programming phase and an increased availability of reusing components as showed in the table.
The pilot is able to demonstrated the advantage of Robmosys approach in the scenario of educational platform on task-level programming. The metrics show an overall reduction of time efforts during programming phase and a increased availability of reusing components

Goal COM-G1: Increase the robot programming ease of use and improve the flexibility of the system			
Question	Metric	Current	Target
COM-Q1: How a model driven approach by task composition could reduce programming efforts?	COM-M1: Effort of programming and training phase	Medium-Low	Low
	COM-M2: Number of functional components available and reused	Medium-Low	High
COM-Q2: How flexibility using mixed port could decrease set-up effort?	COM-M3: Configuration steps needed for a mixed-port	Medium-Low	Low
	COM-M2: Number of functional components available and reused	Medium-Low	High

Table 4.1: Benchmarking plan for goal COM-G1: Increase the robot programming ease of use and improve the flexibility of the system.

5. Human Robot Collaboration for Assembly (CEA)

5.1 Progress Summary

In the past period, the focus of the pilot was to:

- Extend the tools functionalities and build more RobMoSys-conformant models based on the pilot through the ITPs contribution
- Extend the pick-and-place task to build an assembly task
- A new release of the virtual machine with an updated version of *Papyrus for Robotics*, ROS2 and the pilot and ROS2 stack for ISybot.
- a benchmarking plan to assess the pilot's objectives

5.2 Collaboration with the ITPs

The pilot has been used as concrete application scenario to foster the collaboration with several RobMoSys Integrated Technical Projects (ITPs).

- Mood2Be[1]. Exporter from *Papyrus for Robotics* behavior tree models to the XML format supported by BehaviorTree.CPP.
- eITUS[2]. Components' faults modeling and code-generation to support safety analysis through simulation-based fault injection
- SafeCC4Robot[3]. Improve the modeling interface to integrate methodological guidance for the development process and supporting contract-based validation approaches.
- SCOPE[4]. Exporter from *Papyrus for Robotics* models of skills, behaviors and system architecture to the SCOPE toolchain for the analysis of correct task execution.
- ForSAMARA[5]. Extension of task-based hazard and risk analysis according to the project needs
- Miranda[6]. Modeling and code-generation for a field trial application of an inspection robotic rover in the context of a nuclear site.
- Planning4Papyrus. Guidelines for improving *Papyrus for Robotics* with PDDL Planning.
- Human-Robot Cooperation (HRC). Guideline for including human-robot coproduction and human skills in RobMoSys.

[1] https://robmosys.eu/mood2be/
[2] https://robmosys.eu/e-itus/
[3] https://robmosys.eu/safecc4robot/
[4] https://robmosys.eu/scope/
[5] https://robmosys.eu/forsamara/
[6] https://robmosys.eu/itp-2-2-2/

5.3 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, easy task definition (to hide low-level details) and to ensure task reusability, safety and more particularly risk assessment are key features that this pilot aims to realize for reducing risk occurence. The collaboration between our pilot and other ITPs helped to extend *Papyrus for Robotics* tools with more advanced features that allow easy orchestration of the task.

Scenario

The pilot demonstrates task and system definition for a human-robot collaboration use case: assembly through RobMoSys tools, mainly *Papyrus for Robotics*. The interaction between the robot and the operator is direct (with no fences) for teaching an assembly task to the robot. The robot then operates automatically. The assembly task relies on the developments made in the last period of RobMoSys for a pick-and-place task (described in the deliverable D4.2). In these previous developments of our pilot, we used ISybot collaborative robot. We applied the same models and approach to Franka Emika robot with more modules to showcase models and components reusability and consolidate the assembly skills defined for the robot.

Use cases

Use case 1: Context-aware robustness for an assembly task

Deliverable D4.2 described a use case focused on a pick-and-place task. That use case demonstrated the interaction between four distinct roles of participants in the RobMoSys ecosystem, as shown in Figure 5.1: *(i)* domain experts, who defined a set of robot skills to realize the application, namely *grasping*, *ungrasping* and *moving*; *(ii)* component developers, who defined a concrete robot component that could realize the skills from the above skill set; *(iii)* behavior developers, who described the pick-and-place robot task as a behavior tree, based on the skills from the same skill set; and *(iv)* system developers, who created the system specification in terms of an instance of the robot component and the pick-and-place task assignment. D4.2 showed how the building blocks provided by the different participants could be composed together using *Papyrus for Robotics* to deploy and run the complete application using the ISybot collaborative robot arm. In this deliverable we extend the pick-and-place use case with more skills to describe an assembly task. We change the robot that performs the task (from ISybot to a Franka Emika one) to demonstrate robustness and reusability enabled by the RobMoSys approach. In this use case, the domain experts define an additional set of robot skills to realize the application, namely *inserting* and *screwing*, and include Human-Machine Interface (HMI) skills, namely *alerting human operators*. Component and behavior developers extend the respective models to realize the new skills in a robot component and use them in the task description in form of behavior tree, respectively. At run-time, the task is executed and its constraints are checked continuously based on the environment actual data. In case of deviation or unexpected behavior, the orchestrator orders the robot to go to a safe position and raises an event through the HMI to call for human intervention. Once the error fixed, the human indicates that the intervention is finished and the orchestrator

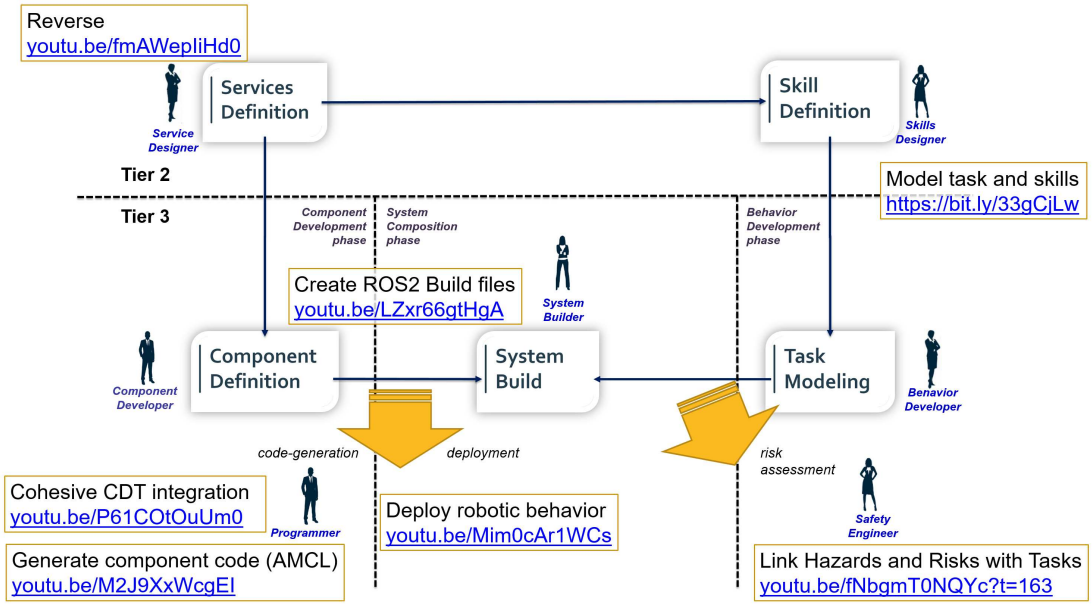


Figure 5.1: Collaboration between roles supported by *Papyrus for Robotics*. Each arrows means that the source role provides models for the target role. The figure includes links to additional documentation resources for each role.

continues with the next planned actions. This use case will be used for the benchmark of this pilot.

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

In the second use case we demonstrate the interaction between system developers and safety engineers (Figure 5.1). System developers provide information about the operational scenario for the task, the agents and the involved objects. Safety designers use *Papyrus for Robotics* to apply Hazard Analysis and Risk Assessment (HARA) techniques to the task specification, following guidelines conforming to safety standards. For each action in the task, safety designers list all the relevant hazards and compute the risk indices. After having computed the risk criticality, safety designers provide risk reduction measures for each hazard associated to an action. Finally system developers validate and put in place the necessary safety measures and deploy the system.

5.4 Focus and Coverage of RobMoSys Features

The pilot was initially intended for the use with ISybot collaborative robot[7] and we extended its usage with another collaborative robot with the same capabilities: Franka Emika[8]
This pilot focuses on task specification, reusability and monitoring, and safety functions. We built upon the simple pick-and-place task developed in the last period to develop an assembly use case. We use a new gripper for manipulating objects like gears, screws, washers, etc., while in the first use story for pick-and-place we used another gripper meant for manipulating paper 5.2. The

[7] https://www.isybot.com/
[8] https://www.franka.de/

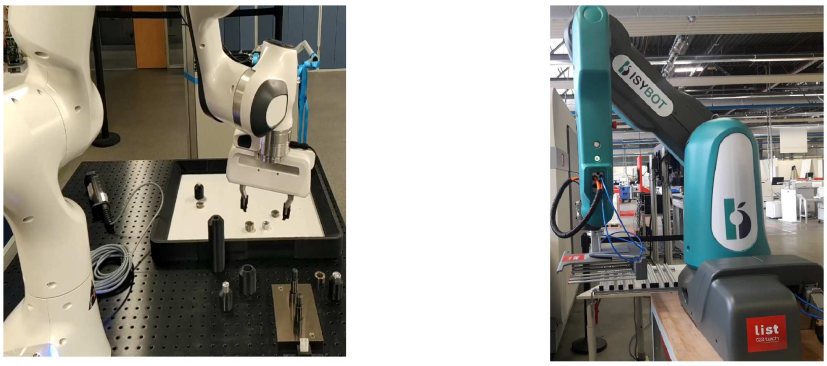


Figure 5.2: Franka Emika and ISybot grippers

operator interacts with the robot for teaching the task to the robot. Then the robot executes the assembly task. *Papyrus for Robotics* tool was also enriched with new functionalities, through new developments fed by the collaboration with ITPs.

5.5 Technical Details

The pilot is fully supported by *Papyrus for Robotics* toolchain[9], a Papyrus[10]-based domain specific modeling language for robotics. *Papyrus for Robotics* features a modeling front-end which conforms to RobMoSys foundational principles of separation of roles and concerns.
For more information, see the following resources:

- Pilot description in the RobMoSys Wiki[11]
- A wiki page[12] and a video[13] that show some of the mentioned previous models with skills modeling for robotic behavior featuring the pilot and agile risk assessment
- A description of the robotic task and its execution with the behavior tree in simulation[14].

5.5.1 Technical description of system and task

Figure 5.3 shows the setup of the assembly task. The robot has a number of different objects to assemble on a fixed axis on a plate. The robot has to apply the relevant strategy for the manipulated object. For instance, we consider here 2 objects that are nuts and metal washers. For each object, a specific skill has been developed for grasping, insertion and screwing. However, the way the skills are implemented is decoupled from the behavior tree (BT) definition.
Figure 5.4 shows the behavior logics for the assembly task.
The task demands the execution of specific procedures to initialize and prepare the robot. The actual assembly task description prescribes a set of skills calls to perform an assembly task (these

[9] https://www.eclipse.org/papyrus/components/robotics/
[10] Papyrus is an industrial-grade open-source Model-Based Engineering tool, see https://www.eclipse.org/papyrus/
[11] https://robmosys.eu/wiki/pilots:hr-collaboration
[12] https://wiki.eclipse.org/Papyrus/customizations/robotics/hara
[13] https://youtu.be/fNbgmTONQYc
[14] https://wiki.eclipse.org/Papyrus/customizations/robotics/demos/pick-and-place

Figure 5.3: Assembly task setup

spaces are known and assigned as input parameters through programming by demonstration functionality to the BT leaves representing concrete actions). These skills are implemented inside the module Robot Skills Service that communicated directly with the robot controller. The manner that the skills are implemented is not of interest here, we only need to know that the skills are available and that they propose the necessary strategies for the objects present in the task. So the objects positions are known by the robot. The orchestrator triggers the first action of the robot for inserting a metal washer on the axis with the adapted strategy. Once the insertion is done, the next action for screwing a nut on the axis is launched. If a problem occurs (e.g object lacking or action failure), the orchestrator asks the robot to go to a safe position and a message is displayed for the operator via an HMI so that he can intervene to fix the problem.

5.5.2 Models

The first use case considered for our pilot builds upon the pick-and-place application with the ISybot collaborative robot arm described in D4.2.
For the pick-and-place we modeled 3 skill definitions, namely `Grasp`, `Ungrasp` and `MoveTo`, with the corresponding (3) coordination service models and 4 data types, representing the coordination information between the task and the service level[15]. In addition, we defined three Tier 3 models: 1 component definition representing the ISybot, with 3 coordination ports, each one providing

[15] https://robmosys.eu/wiki/general_principles:separation_of_levels_and_separation_of_concerns

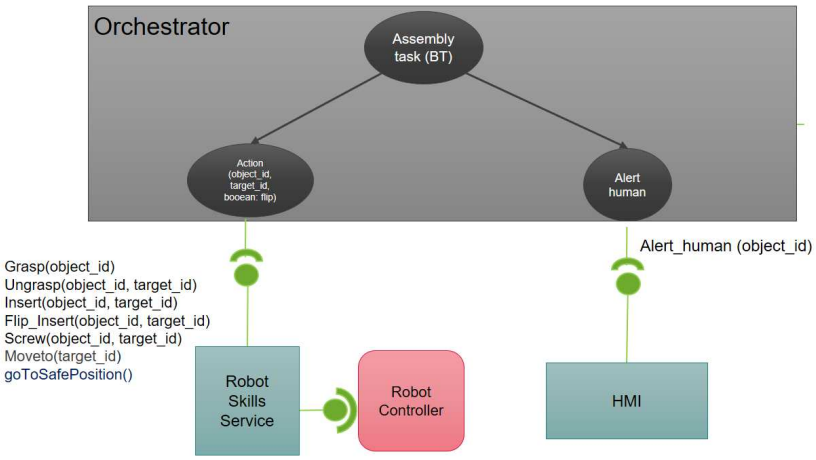


Figure 5.4: Assembly task orchestrator

a coordination service to realize the corresponding skill from the skill set above; 1 behavior tree model describing the pick-and-place task as a sequence of skills from the same skill set; and finally, 1 system model where the ISybot component is instantiated and the task assignment is specified. From this base, we created 3 new skill models (including their coordination services) representing the additional skills for the assembly task; 1 new robot component model (`RobotSkillServer`) representing the Franka Emika arm and 1 HMI component (`AdaptiveHMI`), each one providing the right coordination interfaces to realize all the skills (the 3 from the pick-and-place task, plus the 3 new skills for the assembly task); 1 behavior tree model describing the assembly task including the new skills and reusing a number of pick-and-place sequences from the initial base model. Finally, the system model was now composed of the 2 component instances and the assembly task assignment. Figure 5.5 shows some of the described models.

5.6 Key Performance Indicators and Benchmarking

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 demonstrate how the RobMoSys approach increases the quality and robustness of robotics applications and enables fast adaptation and reuse of the deployed solutions in different contexts (different robots, tool upgrades, etc.).

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 ensure the reusability of the system?
- `CEA-Q2`: How can we reduce the programming efforts?

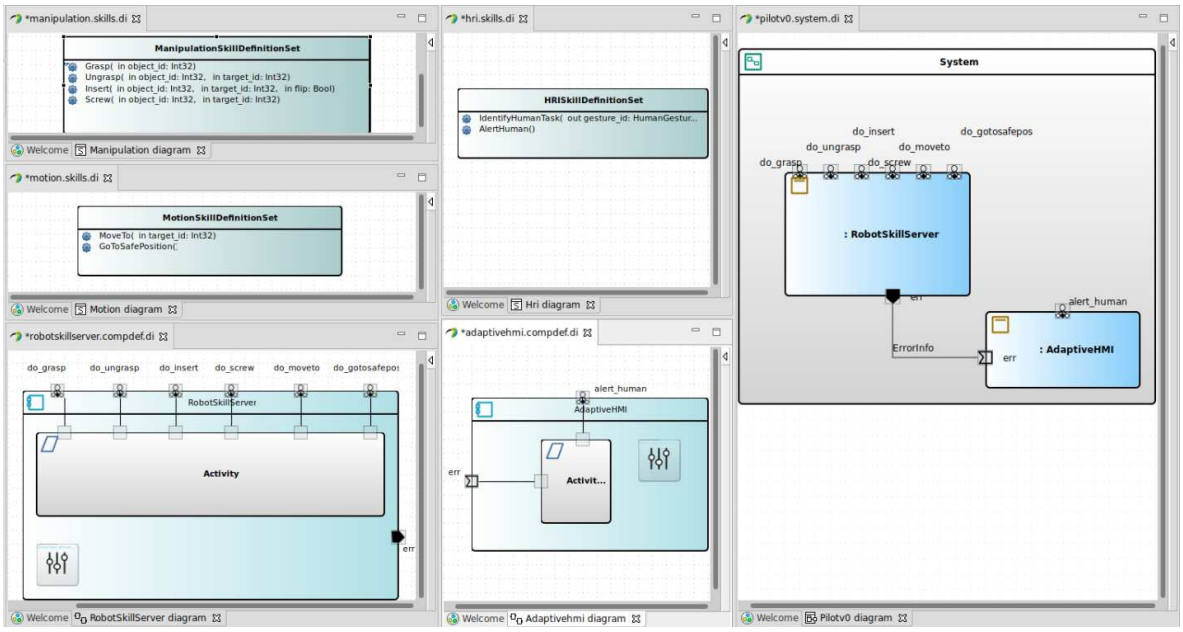


Figure 5.5: (Part of) Models of assembly task

- 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 reusability, we can consider the following answer:

- CEA-Q1-A1: Ensuring the reusability of the system by defining easily-configurable components and generic interfaces

Regarding CEA-Q2 about the reduction of programming errors, code generation will ease the system programming by reducing the programming effort:

- CEA-Q2-A1: The automatic generation of component codes from models with systematic application of best-practice naming/programming conventions reduces erros, programming effort and improves code readability and maintenance.

For CEA-Q3 about system robustness at run-time, 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-Q2-A1, CEA-Q3-A1 and CEA-Q3-A2, the following metrics have been identified. The metrics described in D4.2 were adjusted to the new pilots needs and objectives. Of course, the metrics corresponding to CEA-Q1-A1 only consider the parts of the two tasks, pick-and-place and assembly, which are functionally similar, i.e., robot motion, grasp and ungrasp capabilities. The metrics corresponding to CEA-Q2-A1, CEA-Q3-A1 and CEA-Q3-A2 consider all the components.

CEA-M1: Percentage number of reusable skill and task models

This metric will allow determining how many behavior models can be reused despite the changes that the deployment of the different use cases require. Models considered in this metric include the robot skills and the BT specifications of pick-and-place and assembly tasks.

$$M_{\text{CEA-M1}} = \frac{\text{Reused skill models} + \text{Reused BT models}}{\text{Total behavior models}} \tag{5.1}$$

CEA-M2: Variation between similar robot component models

This metric allow us to measure the effort done by a System Builder in replacing a robot component with a functionally similar one. It accounts the number of component differences, in terms of input and output ports, including port types, such as coordination interfaces for coordination ports and component services for service ports, and configurable parameters (if any). The larger is the computed number, the higher is the effort of the System Builder.

$$M_{\text{CEA-M2}} = \sum \text{Different configurable parameters} + \sum \text{Different inputs and outputs} \tag{5.2}$$

CEA-M3: Reduction of the effort spent in programming

The effort depends on the user profile. It is difficult to standardize as it depends on the user background and expertise in programming. We decided then to choose a metric that, for each component, accounts for the number of generated lines of code (LoC) of C++ header/body files, build files and message, service and action definition files against the total number of LoC (including the code written manually by the component developer). We consider simple heuristics to account for the additional modeling effort implied by the use of an MDE approach: we consider that the effort of modeling a component port, a parameter and an activity corresponds to the effort of writing 2-5 LoC, depending on the programmer experience. The metric is as follow, where C is the set of all the components. The larger is the computed number, the higher is the reduction of the effort spent in programming, which also implies less maintenance efforts.

$$M_{\text{CEA-M3}} = \frac{1}{|C|} \sum_{C} \frac{\text{Generated LoC} - \text{LoC due to modeling effort}}{\text{Total LoC}} \tag{5.3}$$

CEA-M4: System ability to report the state of task execution

This metric allows us to measure the system ability for following the task execution and reporting the components'states at runtime.

$$M_{\text{CEA-M4}} = \frac{\text{Reported states by components}}{\text{Total reportable states of components}} \tag{5.4}$$

CEA-M5: System ability to report an error to the operator

This metric allows us to measure the system ability to communicate an error to the operator at run-time with RobMoSys tools. The metric is as follow, where *Low* indicates the system ability to at least display a text message via the HMI; *Medium* indicates the system ability to display a text message through the HMI and performs a basic behavior adaptation during task execution; and, finally, *High* indicates a full integration of advanced notification mechanisms about the system

Goal CEA-G1: Increase the robustness, reuse and fast adaptation of robotics software systems				
Question	Metric	Prior	Target	Current
CEA-Q1: How can we ensure the reusability of the system?	CEA-M1: Reusable skill and task models	Low	80%	100%
	CEA-M2: Variation between similar component models	High	≤ 2	0
CEA-Q2: How can we reduce the programming errors?	CEA-M3: Reduced effort spent in programming	Low	Medium	[35%,67.5%]
CEA-Q3: How can we increase the system robustness at runtime?	CEA-M4: Ability to report the state of task execution	80%	$> 80\%$	100%
	CEA-M5: Ability to report an error to the operator	Low	Medium	Medium

Table 5.1: Benchmarking plan for goal CEA-G1: Increase the robustness, reuse and fast adaptation of robotics software systems.

state to the operator (via a set of dashboards, etc.) and real-time, seamless adaptation of task behavior.

$$M_{\text{CEA-M5}} = \text{Low} \ / \ \text{Medium} \ / \ \text{High} \tag{5.5}$$

5.6.4 Benchmark

Benchmarking results

The benchmarking results according to the Goal-Question-Answer-Metric approach are displayed in Table 5.1. The *Prior* column represents the status of our robotic software platforms in the beginning of the RobMoSys project. The *Current* column represents the KPIs that we actually achieved in the pilot and the *Target* column represents the target goals that we wanted to reach with the obmosys approach.
For what concern system reusability, prior to RobMoSys the lack of clearly identified architectural patterns for task-plot[16] and component[17] coordination caused little reuse and high variation in our software models, even for functionally similar use cases. The RobMoSys approach enabled the *complete* reuse of skill and task models for this pilot (CEA-M1). Also, the variation index CEA-M2 resulted to be the *lowest possible* (0), because the only different parameters and ports in the Franka Emika robot component with respect to the ISybot component are those that enable the realization of unique skills in the assembly task (i.e., those skills that couldn't be reused, because not available in the pick-and-place task).
For what concerns the reduction of programming errors in robotic software, the quantitative analysis for this pilot follows from the values in Table 5.2. For the two components AdaptiveHMI and RobotSkillServer, the 1st column represents the number of *generated* LoC in the C++ body and the 2nd column represents the number of *manual* LoC added by component developers. The latter realize the (in-component) binding between the "outside" interface of the component

[16] https://robmosys.eu/wiki/general_principles:architectural_patterns:robotic_behavior
[17] https://robmosys.eu/wiki/general_principles:architectural_patterns:component-coordination

	Body files		Header files	Service files	Build files	
	Generated	*Manual*	*Generated*	*Generated*	*Generated*	*Manual*
HMI	166	28	225	10	190	7
Skill Server	422	192	381	45	199	7

Table 5.2: Generated and manually added LoC for the C++ header/body files, service and build files of components `AdaptiveHMI` and `RobotSkillServer`.

(coordination/data-flow interface) and the component-specific internals (algorithms). The number of *generated* LoC of C++ header files and that of *generated* service definition files (ROS2 message, service and action files) are in columns 3 and 4, respectively. No manual modifications have been needed for these files. Finally, column 5 represents the number of *generated* LoC in the build files and column 6 the number of *manual* LoC added by component developers to compile and link the component code against the functional layer provided by algorithm developers in binary form. According to `CEA-M3`, the achieved index of reduction of programming errors and efforts is between 35% and 67.5%, depending on the user background and expertise in programming.
Finally, for what concerns increasing the system robustness at run-time, prior to RobMoSys our robotic software systems were already able to report information about the state of task execution, e.g., running states and state transitions for tasks expressed as finite state machines (`CEA-M4`). The missing information (estimated to 20% of the total information that could be available for the system) was about the component states, due to a lack of clearly identified patterns for managed components[18]. The RobMoSys approach enabled the *complete* tracing of the component states execution. In addition, when a part was not found during the execution of the assembly task, the system was able to report an error to the operator through a textual message displayed in an HMI and performing a basic behavior adaptation by moving to a safe position and waiting for human intervention (`CEA-M5`).

Benchmarking challenges

The benchmarking process is generally associated with difficulties in mapping the benefits of the model-based development approach to quantitative properties. The assembly task, for instance, is a more complex version of pick-and-place one. It is then hard to quantify the prior level of each of the KPIs. It is also hard to quantify the know-how of a user and measure some of the KPIs because some of them depend on the use expertise and best practices application (like programming effort, modeling effort, and number of models and components). It is also difficult to estimate a percentage of effort reduction without a statistically relevant number of users. In order to have significant results, an empirical study would have been interesting.

[18] https://robmosys.eu/wiki/composition:component-activities:start

6. Intralogistics Industry 4.0 Robot Fleet (THU)

6.1 Progress Summary

In this reporting period, a particular focus was put on the uptake of this pilot by projects outside of RobMoSys. Thereby, the scenario not only received a lot of extensions, but it also served as a testbed for the accessability and the applicability of the RobMoSys approach via the SmartMDSD toolchain. This gave a lot of insights into the maturity level of systems built with the SmartMDSD toolchain, the maturity level of the SmartMDSD toolchain itself, and in particular insights into how well the envisioned benefits and advantages of the RobMoSys approach already show a real positive effect for the users. A major effort was to include the feedback from real-world scenarios with high technology-readiness levels into the latest release of the pilot and the SmartMDSD toolchain and to come up with further use cases of RobMoSys in the context of this pilot.

6.2 Pilot Scenario and Use Cases

The Intralogistics Industry 4.0 Robot Fleet Pilot is about goods transport in a company, such as factory intralogistics. It features the delivery of a set of orders by a fleet of collaborating robots. This pilot is not just a single and specific application. Rather, it is a complete testbed for manifold aspects of RobMoSys as the individual hardware assets and software assets can be used to build a number of different systems and applications. At first, the pilot was meant to showcase RobMoSys benefits in particular by the example of robotics navigation (e.g. performance of goods delivery and according non-functional requirements). Meanwhile, it comes with a manipulation stack and an object detection stack that are also fully RobMoSys compliant.

Figure 6.1: The Intralogistics Industry 4.0 Robot Fleet Pilot.

The pilot has been described in detail in the previous version of this deliverable, D4.1 - First Report on Pilot Cases. Examples and videos of the pilot in action are accessible via the RobMoSys wiki[1]

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

(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.

and can also be found on the YouTube channel of THU[2]. The pilot includes a set of different robots and stations that interact:

- transfer stations at which small load carriers (boxes, trays) are picked up/delivered to
- transfer stations at which items/goods are picked up/delivered to
- robots to pickup, transport, and deliver small load carriers
- robots for mobile manipulation to do order picking of goods into small load carriers

The pilot is physically located at THU and may be used on site or remotely. It consists of hardware (e.g. fleet of robots, robot platforms from different vendors, different laser rangers, different RGBD-sensors, UR5 manipulator, and transfer stations with conveyor belts for handling small load carriers) and models (e.g. domain-specific models of services, skills, tasks, software components).
In the previous reporting period, the models were already fully conformant to RobMoSys and most of them already came with executable software (i.e. software components, task plots, deployments provided via the RobMoSys conforming SmartMDSD toolchain).
Now there is a full set of models, corresponding executable software and real-world scenarios available that are composed out of these building blocks and that all are based on the latest SmartMDSD toolchain release. An excerpt of the pilot is available in simulation for off-site use.
The pilot has been directly adopted by 7 out of 28 RobMoSys Integrated Technical Projects (ITP) as is partially documented in the RobMoSys wiki [3]:

- 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)
- Advanced Robot Simulations for RobMoSys (AROSYS ITP)
- QoS Metrics-in-the-loop for better Robot Navigation (MIRoN ITP)

[2] https://www.youtube.com/user/RoboticsAtHsUlm/videos
[3] https://robmosys.eu/wiki/community:start

- Guidelines for improving SmartMDSD with DDS and QoS attributes for communication (SmartDDS ITP)
- OPC UA for RobMoSys (OPC UA ITP)

RobMoSys compliant contributions to this pilot were also done (and are still being done) by the following projects that are independent from RobMoSys:

- BMBF LogiRob - Multi-Robot-Transportsystem im mit Menschen geteilten Arbeitsraum (National Project, Germany)
- BMWi PAiCE SeRoNet - Plattform zur Entwicklung von Serviceroboter-Lösungen (National Project, Germany)
- ZAFH Intralogistik - Kollaborative Systeme zur Flexibilisierung der Intralogistik (Federal State Project, Germany)

Since every FESTO Robotino robot is delivered and operated with RobMoSys compliant SmartSoft components, all Robotino robot owners worldwide are users of at least parts of this pilot and rely on the flexible navigation stack[4]

Scenario

Figure 6.3 shows examples of scenarios in which the Intralogistics Industry 4.0 Robot Fleet Pilot has been used. These range from simulation over lab environments to tests at sites of end-users.

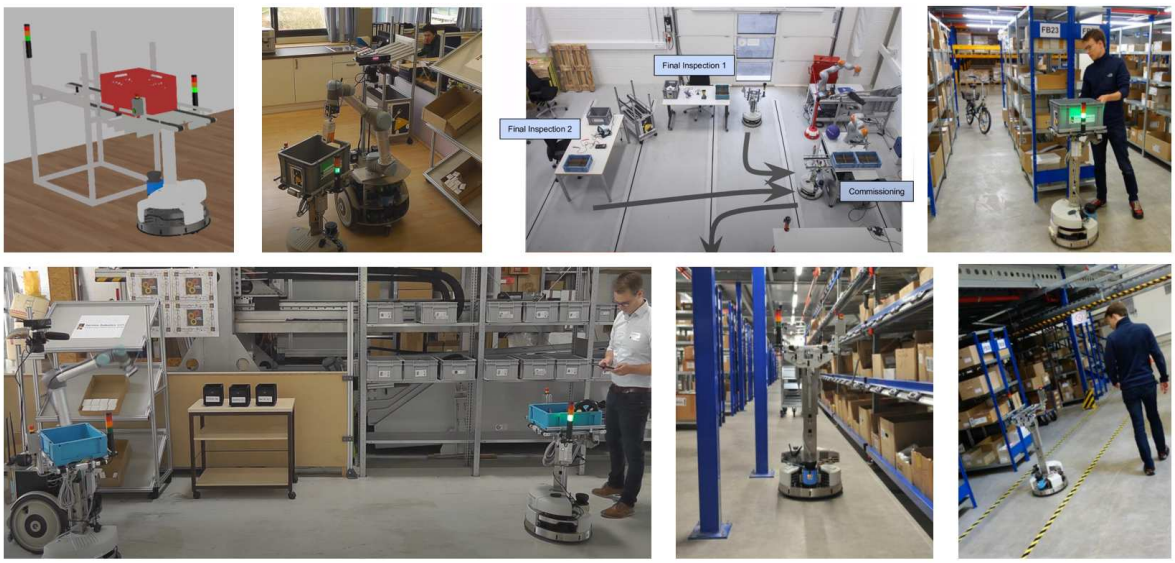


Figure 6.3: Examples of scenarios in which the pilot has been used.

[4] https://robmosys.eu/wiki/domain_models:navigation-stack:start

User stories

The pilot skeleton and the pilot scenarios are built by using the SmartMDSD toolchain that conforms to the RobMoSys composition structures. This pilot demonstrates the application of RobMoSys composition structures and of separation-of-roles along various axes ranging

- from domain-specific models at Tier-2 (services, skills, tasks) to concrete assets at Tier-3 (software components, system architectures, task plots, deployments)
- from simulation and lab-deployments to on-site real-world tests and even worldwide shipping of the flexible navigation stack
- from composing new pilot scenarios by system builders to modifying and configuring existing ones by the end-user
- from collaboration of different roles in the same organization to different roles spread across different organizations and different locations and thus selecting components via data sheets on platforms[5]

Thus, the pilot covers many of the technical user stories that are outlined in the RobMoSys Wiki[6] but also user stories related to ecosystem infrastructure (brokerage platforms for services and assets), interactions along separation-of-roles and user stories related to the toolchains (in this pilot the SmartMDSD toolchain). We highlight some unique system-level user stories that are showcased via this pilot:

Hardware / software component replacement: As system builder, I received an order for a new mobile robot application. The new application is quite similar to an already existing one. Thus, I want to reuse as much as possible of the already existing solution. In order to make a perfect match to the new requirements (e.g. reuse navigation skills, reduce costs, add person following skills), already fitting skeletons and components shall be kept and the effort shall go into targeted replacements, removals and additions of components and into rearrangements of task plots. I expect that the size of changes (and thus the effort) is in relation to the similarity between the already existing use case and the new one.

Middleware-agnostic systems and mixed middleware systems: Depending on the application environment, there are different requirements for the underlying middleware (performance, throughput, security, customer specifications, e.g. ACE, DDS, OPC UA). As system builder, I want to be able to fulfill these requirements by just configuring the assignment of middlewares to system parts at any time during deployment (without having to go into the system again or even interact with developers of the used components for changing source code and recompiling).

Mastering system level properties: As system builder of a transport robot with an onboard camera for navigation, I need to convincingly explain to the hospital that privacy requirements are fulfilled and cannot get violated (e.g. that raw camera images never leave the robot system). Furthermore, I must check whether the robot is able to react in time to dynamic obstacles such as persons (i.e. the data processing chains and corresponding response times from laser data to motion control commands fulfill the specified requirements).

[5] https://www.robot.one/, https://www.xito.one/
[6] https://robmosys.eu/wiki/general_principles:user_stories

Digital data sheets for component selection: As system builder, I got a contract to provide a mobile robot for goods transport in a company. I need to select from offers of different navigation skills those components that adequately match the given requirements, that is technically (such as being able to handle 3D sensing of the surroundings for the navigation) as well as economically. I want to use these components directly, without a need for interaction with the component developer, and thus a data sheet shall provide all the information I need to know in order to correctly configure and use the component in my setting.

Migration path and connecting to brown-field installations: As system builder, I got a contract to provide a flexible transport solution based on a mobile robot that is already available at the customer. The mobile robot just talks ROS and shall be reused although all the new skills shall be composed out of RobMoSys technology in order to endup with economically feasible efforts. The new application has to seamlessly interact with the customer's existing infrastructure (transfer stations for small load carriers talking Industry 4.0 standard OPC UA).

Semantic configuration by the end-user: The reason for the purchase of a fleet of robots was their promise for being easily adjustable to different flow of goods and to different layouts. Of course, I want to do these configurations by myself in order to most flexible in responding to my shortly changing needs. I do not want to make an IT project out of these reconfigurations.

It is also important to clearly point out for which kind of user stories this pilot has not been used yet - no user stories related to safety have been applied to this pilot yet.

6.3 Focus and Coverage of RobMoSys Features

The pilot is fully supported by the SmartMDSD Toolchain[7], an Integrated Software Development Environment (IDE) for system composition in a robotics software business ecosystem. This ensures full conformance to the RobMoSys methodology when using this pilot.
The pilot covers most ecosystem roles (despite the safety engineer), most metamodels, robotic domain models related to all aspects of mobile manipulation with a fleet of robots, and composition ranging from software components over task level coordination to systems-of-systems.
The pilot is intended to showcase the ease of system building via composition of software components to complete robotics applications. This also includes the deployment in real-world scenarios and on-site configurations by the end user. It is important to understand that the pilot is not only thought for internal use by RobMoSys core partners and by RobMoSys ITPs, but also for use in other projects that want to exploit RobMoSys technology. Thereby, the pilot gives valuable insights into the accessability and the usability of the RobMoSys technology via the SmartMDSD toolchain and the already achieved technology-readiness-level in the processes and the achieved technology-readiness-level in the outcoming systems.
The following are implemented use cases to illustrate the benefit of RobMoSys. More information on the following material is available within 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:

[7] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:start

- see the excange of a hardware and software component demonstrated using the industry 4.0 pilot[8]
- see how PAL Robotics modifies the pilot to use the software with its TIAGo robot[9]
- there are different combinations of simulators (Gazebo, Webots), robots (Pioneer P3DX, PAL Robotics TIAGo, FESTO Robotino, Segway), distance sensors (SICK, Hokuyo, Intel Realsense) and different navigation components such as AMCL, GMapping, Cartographer

Middleware-agnostic systems and mixed middleware systems:

- see the change of the middleware on a per-servicce basis (ACE and DDS mixed middleware system)[10]

Ecosystem collaboration including the different roles that participants can take:

- see how the system builder and the behavior developer can interact[11]

Mastering system level properties: (extra-functional) properties, dependency graphs for composed components:

- see how the RoQME ITP monitors non-functional properties on this pilot[12]
- see by the example of this pilot how the SmartMDSD Toolchain assists in understanding data-trigger-chains and how it assists in finding proper cycle times[13]
- see how dependency graphs manage privacy requirements[14]

Digital data sheets for component selection:

- see how digital data sheets are integrated into the SmartMDSD toolchain[15]
- see where the digital data sheet comes to a use[16]

Task level coordination, skills, robotic behavior:

- see Robotic Behavior in RobMoSys using Behavior Trees and the SmartMDSD Toolchain (MOOD2BE ITP)[17]
- see Support of Skills for Robotic Behavior[18]

[8] https://www.youtube.com/watch?v=e4uC0mEWxCk
[9] https://www.youtube.com/watch?v=FCvK9dAZXPo
[10] https://youtu.be/Tvms5MK0m-k
[11] https://robmosys.eu/wiki/community:behavior-tree-demo: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://wiki.servicerobotik-ulm.de/tutorials:start#lesson_8dependency-graph_extensions_for_smartmdsd_toolchain_smartdg
[15] https://wiki.servicerobotik-ulm.de/how-tos:documentation-datasheet:start
[16] https://www.robot.one/search and https://www.xito.one/marketplace.html
[17] https://robmosys.eu/wiki/community:behavior-tree-demo:start
[18] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:skills:start and https://robmosys.eu/wiki/composition:skills:start

- see the SmartTCL Hierarchical Task Nets with the Sequencer (*ComponentTCLSequencer*) and the Knowledge Base (*ComponentKB*)[19]

Migration path and connecting to brown-field installations:

- see how to connect to ROS nodes[20]
- see how to connect to OPC UA devices[21]
- see how to access components from the YARP middleware (CARVE ITP)[22]

Semantic configuration by the end-user:

- see how the end-user can rearrange shelves by himself and how he can modify the workflow[23]

Uptake in other projects:

- see the BMWi PAiCE SeRoNet production logistics show case[24]
- see the BMBF LogiRob show case[25]
- see the ZAFH Intralogistik show case[26]

Maturity level of the pilot:

- used by Festo and shipped worldwide[27]
- see scenarios at Transpharm[28] and at DaimlerTSS [29] that are now fully RobMoSys compliant

6.4 Technical Details

For detailed technical information on the pilot skeleton and the pilot applications, see the following resources:

- Pilot description in the RobMoSys Wiki[30]
- Flexible Navigation Stack[31]
- Mobile Manipulation Stack[32]

[19] https://github.com/Servicerobotics-Ulm/ComponentRepository
[20] https://youtu.be/N1oMMIBIx5k
[21] https://youtu.be/Xi7Irjk8Kyw
[22] https://robmosys.eu/wiki/community:yarp-with-robmosys:start
[23] https://youtu.be/5116bGhXBr8 and https://youtu.be/Nzwjb8BaQns
[24] https://youtu.be/Nzwjb8BaQns
[25] https://youtu.be/ML_BtZsiPHo
[26] https://youtu.be/oGKjK7K76Ck
[27] https://wiki.openrobotino.org/index.php?title=Smartsoft
[28] https://youtu.be/r4mgPgyYISQ
[29] https://youtu.be/cggCY-cvdJ8
[30] https://robmosys.eu/wiki/pilots:intralogistics
[31] https://robmosys.eu/wiki/domain_models:navigation-stack:start
[32] https://robmosys.eu/wiki/domain_models:mobile-manipulation-stack:start

- Software Components for use with the pilot[33]
- Skills for robotic behavior featuring the pilot[34]
- Tutorials for the SmartMDSD Toolchain featuring excerpts of the pilot[35]
- Tutorial for using software components of the pilot with the TIAGo robot base[36]

Software components are available[37] for use with the SmartMDSD Toolchain for immediate composition:

- hardware abstraction for several robot platforms and different sensors/actuators
- navigation: mapping, planning, obstacle avoidance, SLAM
- object detection for mobile manipulation
- manipulation: workspace scanning, path planning, trajectory control
- task level coordination: sequencer for hierarchical task nets, knowledge base for world model coordination
- human-machine-interaction: joystick control, person following, tools for semantic configuration
- components for interfacing to industry 4.0 OPC UA devices and to MES (manufacturing execution system)
- simulation

Figures 6.4, 6.5, 6.6 and 6.7 are to give an overview on some selected details that are showcased in this pilot.

6.5 Key Performance Indicators and Benchmarking

The pilot skeleton and the pilot applications based on it are meant to provide easy access to complex and relevant scenarios such that the benefits and advancements of the RobMoSys approach and of the corresponding technologies can be showcased.
This pilot is a complete testbed for manifold aspects of RobMoSys. At first, the pilot was meant to showcase RobMoSys benefits in particular by the example of robotics navigation (e.g. performance of goods delivery and according non-functional requirements). Meanwhile, it comes with a manipulation stack and an object detection stack that are also fully RobMoSys compliant. The pilot now comprises several real-world scenarios related to goods transport by a robot fleet comprising robots with different skills. These are fully built by using the RobMoSys conformant SmartMDSD toolchain. As the pilot covers a broad set of robotic scenarios (from navigation

[33] https://robmosys.eu/wiki/model-directory:start
[34] https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:smartmdsd-toolchain:skills:start
[35] https://wiki.servicerobotik-ulm.de/tutorials:start
[36] https://robmosys.eu/wiki/baseline:scenarios:tiago_smartsoft
[37] https://robmosys.eu/wiki/model-directory:start

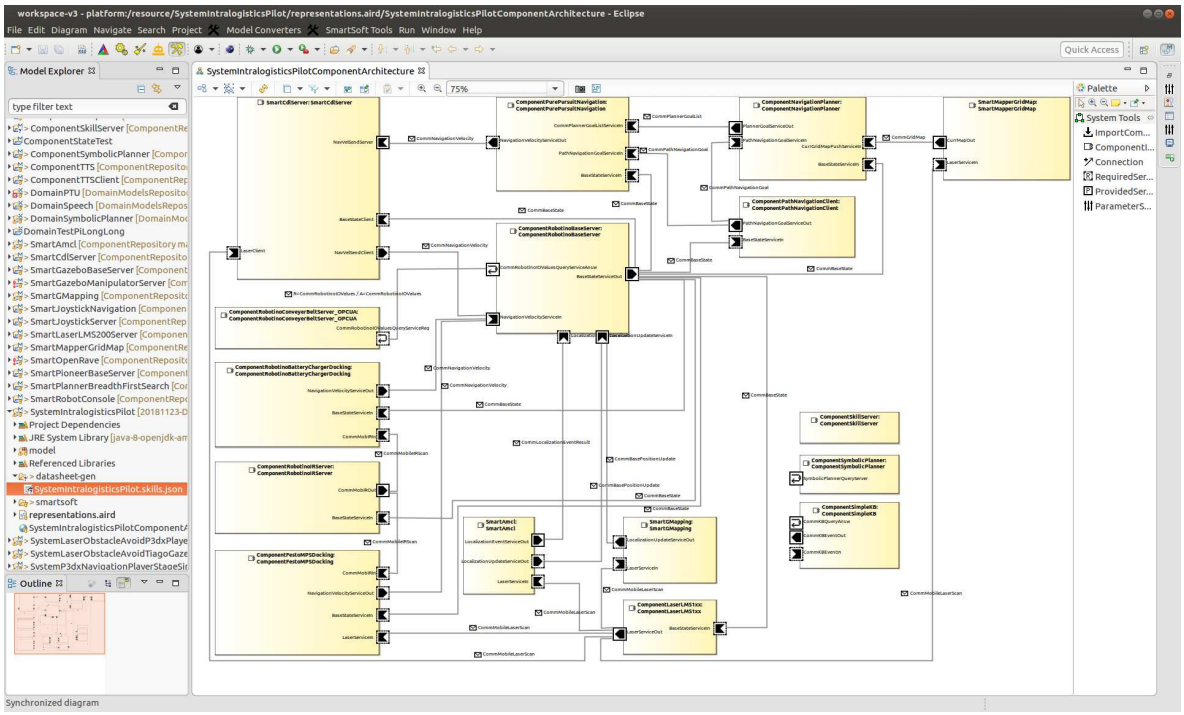


Figure 6.4: The system composition view of a transportation robot of the Intralogistics Industry 4.0 Robot Fleet Pilot in the SmartMDSD Toolchain for composing a pilot application

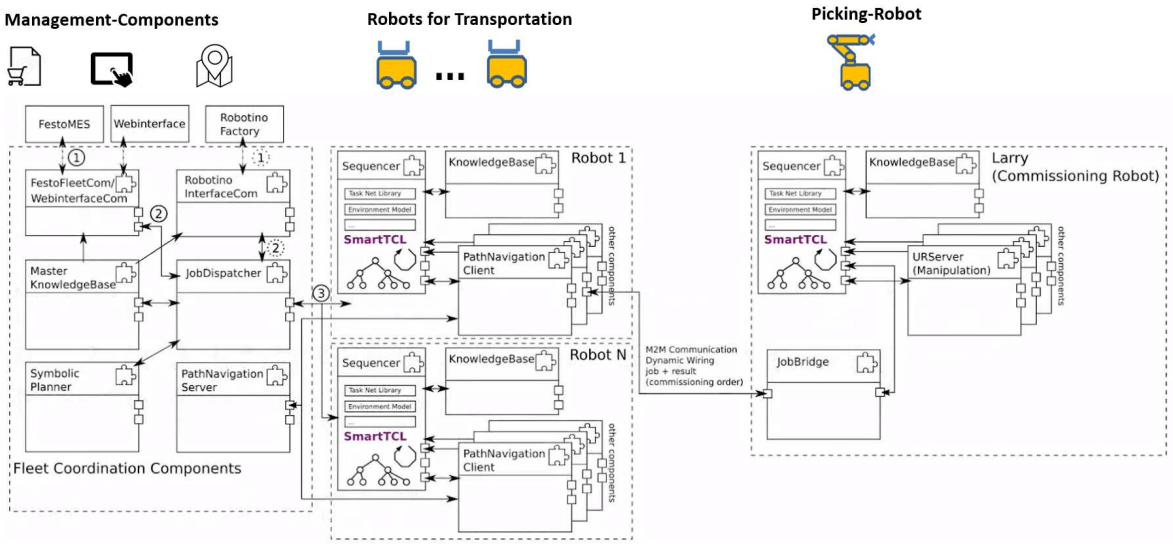


Figure 6.5: Overview on robot fleet with transportation robots and a picking robot.

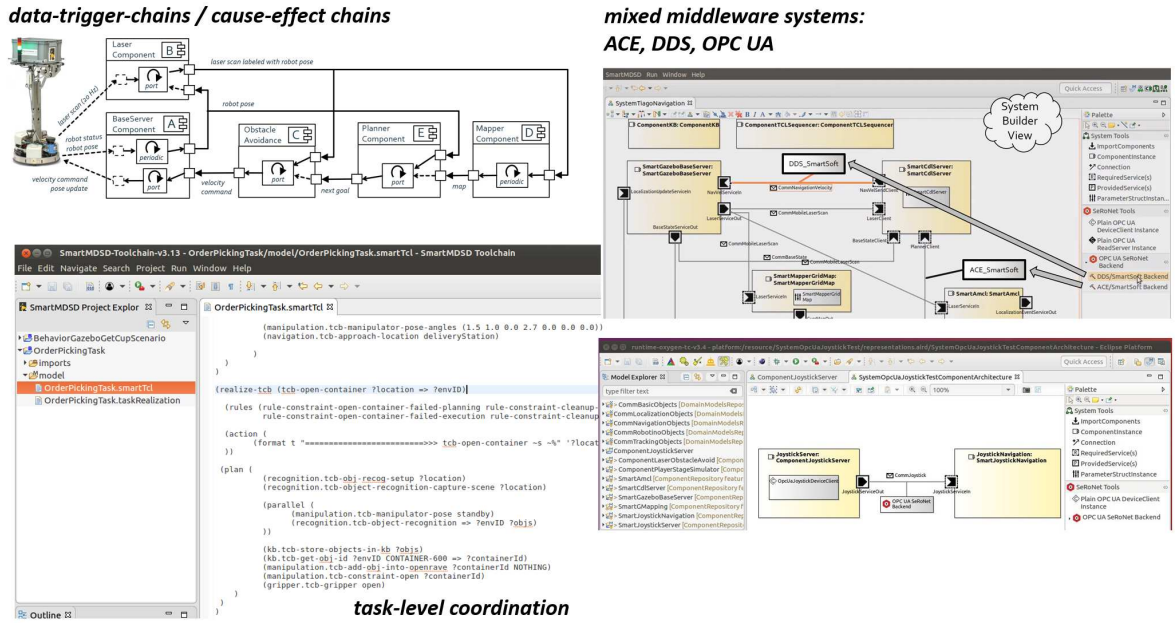


Figure 6.6: Addressing extra-functional properties, mixed middleware systems and task level coordination with this pilot.

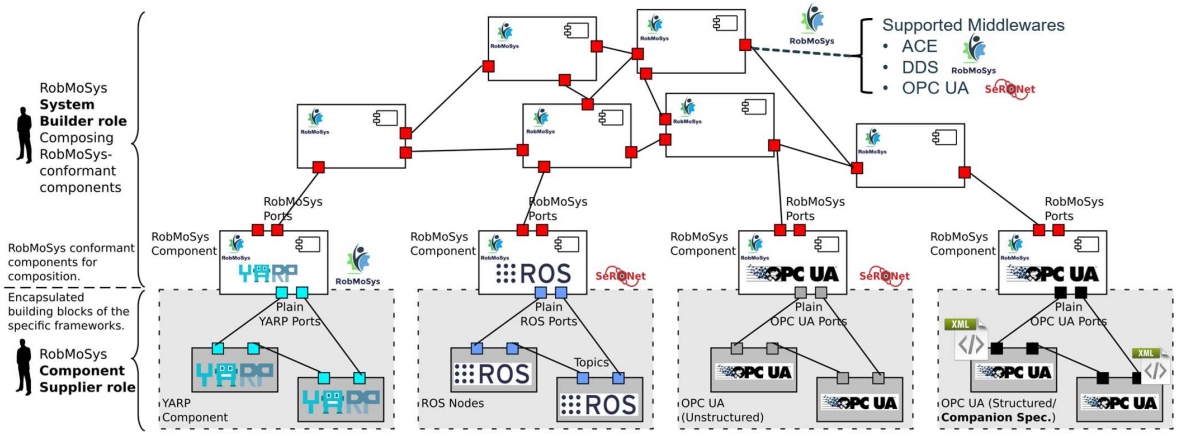


Figure 6.7: Mixed-port components: the pilot talks OPC UA to the transfer stations to handle small load carriers

to manipulation, from single robot task control to robot fleet management) and as these have been successfully implemented and operated in real-world scenarios, this shows the high level of maturity and usability of the RobMoSys compliant SmartMDSD toolchain and of its support for the RobMoSys workflow.
It is quite difficult to come up with expressive key performance indicators for this pilot that addresses so many facets of RobMoSys. Many of the achievements are at a *qualitative* level (introducing a model-driven approach to shape a robotics ecosystem based on separation of roles and composition) and it is about whether the baseline for such an ecosystem has been put into place, has been populated with convincing showcases, and has been taken up by others:

- we do not want to measure output but we want to measure outcome (which in case of this pilot is basically about qualitative features)
- we do not want to measure that we just have the pilot up and running and built it somehow. It is important that we achieved this by applying RobMoSys processes
- we want to measure whether RobMoSys processes lead to better systems (in terms of different criteria such as effort,reliability, etc.)
- as the pilot is built with the SmartMDSD toolchain, benchmarking the pilot quality is also always related to benchmarking the SmartMDSD toolchain

6.5.1 Goals

THU-G1 The goal of this pilot is to demonstrate the benefits of RobMoSys.

6.5.2 Questions

Questions related to goal **THU-G1: Demonstrate the benefits of RobMoSys**:

THU-Q1: What is the coverage of RobMoSys concepts of this pilot?
The achieved results are only convincing in case these have been achieved by applying RobMoSys concepts. Of course, it does not make any sense to just apply as many concepts as possible when this is done only in artificial setups via weak use cases. However, just applying only few RobMoSys concepts would not showcase how well the entirety of RobMoSys concepts finally really fit together.

THU-Q2: Is the pilot both, focused enough for fast take up and broad enough to showcase RobMoSys?
If the pilot is too broad, then contributions can take place in well-separated niches and there is no pressure for testing composition. If the pilot is too narrow, then it might not face challenges of complex systems-of-systems.

THU-Q3: Are the systems built with this pilot mature enough for significant conclusions?
Just running toy examples is not good enough for bold conclusions. Much more convincing are use-cases in real-world testbeds outside of your own lab and performed by others.

THU-Q4: How well do the assets provided by independent roles in the ecosystem finally fit together?
Key for a business ecosystem is that different stakeholders can work independently and

concurrently. For example, a system builder should not need to get in contact with the component builders when using their components.

THU-Q5: Has the effort of addressing a new application been lowered by RobMoSys?
The effort needed to address a new application should be in proper relation to its similarity to an already existing application.

THU-Q6: Do we get better insights into system level properties and can we better manage them?
Basically, the biggest challenge is to master properties of systems-of-systems and RobMoSys is expected to make a big step ahead here.

THU-Q7: How well does the decoupling of solutions and technologies work?
The pace of technologies is much higher than is the pace of solution principles. The latter come with high engineering efforts and costs. Thus, exchange of technologies should be possible without reworking the solution.

THU-Q8: Can the enduser perform reconfigurations without programming?
This is an ultimate test whether the model-driven approach of RobMoSys can bring complex robotic system-of-systems with all their cross-linking (and often hidden) parameters closer to the user.

THU-Q9: What is the learning curve of the toolchain?
If the toolchain is too complex and difficult to use, it is too expensive until one can gain from the advantages of separation-of-roles and of composition. This would be an entry barrier for many highly motivated experts waiting to get access to a robotics ecosystem.

6.5.3 Metrics

Metric 1: Coverage of

THU-M1.1: RobMoSys concepts in terms of roles, Tier-2 meta-models, Tier-3 assets

THU-M1.2: expected combinatorics: number of alternative realizations per capability (e.g. 3 different kinds of navigation), number of different capabilities (e.g. navigation, localization, robot, etc.)

THU-M1.3 user groups (beginner, expert, academia, industry, robotics, software, application, etc.)

Metric 2: Number of

THU-M2.1 user stories demonstrated using the pilot skeleton or the pilot scenario

THU-M2.2 ITPs directly using / contributing to the pilot (internal uptake)

THU-M2.3 projects directly using / contributing to the pilot (external uptake)

THU-M2.4 contacts of the system builder to component providers

THU-M2.5 RobMoSys components and of other components

THU-M2.6 system level properties addressed and successfully showcased

THU-M2.7 exchangeable technologies

THU-M2.8 different successful use cases

Metric 3: Quality

THU-M3.1 Technology-readiness levels of applications using the pilot skeleton.

6.5.4 Benchmark

In the Flexible Intralogistics Scenario[38], each Robotino robot runs 22 RobMoSys compliant software components. The mobile manipulation robot runs 20 RobMoSys compliant software components. The navigation components are the same on all mobile platforms. In addition, there are 8 more RobMoSys compliant software components for the fleet management.

Answer THU-A1 to THU-Q1: This pilot applies the roles component supplier, system builder, behavior developer, service designer, system architect. It does not have a focus on the function developer and it does not apply the role of a safety engineer. The pilot provides domain-specific content at Tier-2 (definition meta-models) such as digital data representations, services, skills and tasks mainly for the navigation domain. It also provides domain-specific content at Tier-2 for (mobile) manipulation and object detection (not as matured as for the navigation domain). It uses the Tier-2 definition meta-models for Tier-3 realization meta-models that finally cover the concrete assets at Tier-3 out of which the pilot skeletion and the related pilot applications are composed. This pilot also covers the workflow along separation-of-roles and composition (digital data sheet for assets and dependency graphs for managing system level properties).

Thus, we rate this pilot to have a very high coverage of RobMoSys concepts.

Answer THU-A2 to THU-Q2: Navigation is at the core of robotics and most aspects of robotics can be illustrated by use cases of navigation. The pilot allows to focus on single robot navigation but also supports a fleet of diverse robots even in workspaces shared with humans. As can be seen by the various scenarios implemented on top of the pilot skeleton, it offers a focused but still highly expandable setting. As can be seen in answer **THU-A4**, the pilot also produced a set of components that can be arranged in different combinations in order to showcase reuse and composition. Of course, there are still plenty of use cases that can be illustrated by this pilot but have not yet done so far.

Thus, we consider this pilot to be a sound and fitting testbed for RobMoSys use cases.

Answer THU-A3 to THU-Q3: The pilot skeleton is being used in at least three different projects outside of RobMoSys (BMBF LogiRob, BMWi PAiCE SeRoNet, ZAFH Intralogistik). The real-world testing covers production logistics as well as order picking. These scenarios do not just run once in a lab, but many rounds in a realistic setting. Furthermore, each FESTO Robotino robot comes with a navigation stack onboard that is compliant to this pilot. These robots are shipped worldwide and are operated worldwide. Since quite some time, we do not get any negative feedback with respect to the navigation.

Thus, we rate the systems built with this pilot to be mature enough for significant conclusions.

[38] https://youtu.be/oGKjK7K76Ck

type of capability	alternatives for a capability	number of alternatives
simulation	Gazebo, Player/Stage, Webots, FESTO simulator	4
real robots	Robotino, PAL Robotics TIAGo, Segway, Pioneer P3DX	4
sensors	SICK, Hokuyo, Intel Realsense	4
navigation	[mapper, planner, CDL motion control] [Corridor Navigation, CDL motion control] [person following, CDL motion control]	3
localization	AMCL, GMapping, Cartographer	3

Table 6.1: Combinatorics by the example of navigation.

Answer THU-A4 to THU Q4: One option to look at this is from the role of the system builder. It is not just about composition of components but also about composition of task plots and the reuse of task plots with alternative implementations of skills. The pilot skeleton led to different alternatives that have been combined in different ways (see table 6.1). Components are not just a 1:1 replacement, but e.g. the navigation components even differ in their granularity and assignment of services. As can be seen from the scenario videos on our YouTube channel[39], the navigation stack is used in quite different task plots (even changing between different navigation modes including person following). All the assets have been built using the Tier-2 domain-specific structures for the navigation stack and they have been combined to different deployments just "as is".

As the above combinatorics ("plug festival") was successful just at the level of the system builder (and without a need for coordination meetings with component developers), we rate the guidance of Tier-2 structures by the SmartMDSD toolchain as success.

Answer THU-A5 to THU-Q5: The Robotino robot normally uses a SICK laser ranger for navigation. However, a new application required a less expensive alternative and the robot had to be equipped with a person following skill. We replaced the SICK laser ranger by an Intel Realsense RGBD camera which is used for navigation as well as for person following. All the navigation components had not been modified. The navigation services proved to be designed such that all the needed sensor data characteristics (variance, viewing angle, etc.) are self-contained in the communication objects and thus get handled automatically (no hidden dependencies and magic numbers).

See also **THU-A2**, **THU-A6** and **THU-A7** for further aspects of reduced effort in composing new and in modifying existing applications.

For the first time, we see a significant reduction of effort for coming up with a solution for an application similar to an existing one (as made possible by composition).

Answer THU-A6 to THU-Q6: We introduced and implemented the concept of *dependency graphs* to manage system level properties in a traceable and analyzable way. We applied that concept to extra-functional properties such as *privacy* (can I get an image of the camera onboard the robot from outside the robot?)[40], to configure *trigger-chains* and to decide on *cycle rates* of periodic activities. All these configurations are done by the system builder

[39] https://www.youtube.com/user/RoboticsAtHsUlm/videos
[40] https://youtu.be/tXEsvgSH9bU

without recompiling software components and without interaction with the providers of the used assets.

To the best of our knowledge, this is the first time that there is such a consistent approach available for system level properties under separation-of-roles and composition underpinned by concrete real-world scenarios. We rate that as a significant leap forward in managing system level properties in composable robotic systems.

Answer THU-A7 to THU-Q7: This has been showcased by mixed-middleware systems being middleware-agnostic. For the first time, the middleware does not need to be specified at component development time. Instead, it is the system builder who assigns middlewares according to application requirements. For this, components are not recompiled but they are just used "as is" and in binary form. This showcased that *early binding of semantics - late binding of technology* is achievable as is required for a composition workflow. As working solutions have been demonstrated by a very demanding use case (middleware agnostic in conjunction with RobMoSys composition workflows), technology decoupling can be put in place also for other parts.

We rate that as a huge leap forward in decoupling solutions and technologies in robotics.

Answer THU-A8 to THU-Q8: This has been showcased by *rearranging navigation corridors* in factory layouts, *rearranging shelves* and *adding new steps* in a workflow[41]. This all has been done just by graphical interfaces by the enduser. These tools then properly distributed all the configuration updates into the world models for immediate impact on the robots. The robots have not been rebooted but just stopped during the user modifications. This shows that the RobMoSys approach helps in consistently managing configurations and presenting them in a user-friendly way. It also shows that there are no hidden parameters somewhere as otherwise those complex on-the-fly reconfigurations would not have been successful.

RobMoSys significantly helps in enabling users to exploit the promise of adaptability of robotic systems.

Answer THU-A9 to THU-Q9: We regularly get support requests from students, doctoral candidates or post-docs in other labs. Very often, we are surprised not yet having heard before that they are using the SmartMDSD toolchain. The context of the questions clearly indicate that they already successfully mastered the role of a system builder without support from us. They are now trying to build their own first component (wrapping their unique libraries) and thereby bumped into challenging questions of how to best address specific needs. This gives a lot of feedback about which kinds of tutorials are still missing (e.g. updated tutorials for advanced topics related to component internals such as functional composition inside a component, advanced tutorials for skill modeling), but also shows that there is no hurdle with respect to standard use cases and with respect to the Eclipse world (SmartMDSD is fully based on Eclipse).

We also use the SmartMDSD toolchain in robotics courses in our bachelor / master programs and of course with PhD level research. The currently educated next generation of students is quite familiar with workbenches like Eclipse. For them, the new aspect is applying model-driven approaches to robotics. That generation is fluent at the user level in handling workbenches like Eclipse.

[41] https://youtu.be/5116bGhXBr8 and https://youtu.be/Nzwjb8BaQns

The BMWi PAiCE SeRoNet project rolls out the SmartMDSD toolchain in its cascade funding calls which already led to industrial components[42]. These lead users from industry also managed their role with only very few support.

Thus, we consider the learning curve to be adequate and to be manageable, even expecting that it becomes even easier in the future.

[42] https://www.xito.one/marketplace.html/

Goal THU-G1: The goal of this pilot is to demonstrate the benefits of RobMoSys.					
Quest.	**Metric**	**Prior**	**Target**	**Current**	**Grade**
THU-Q1	THU-M1.1	basic coverage of component builder, system builder	all	almost all	++
THU-Q2	THU-M1.2	one implementation per type of capability	$\geq$ 2 per type of capability	at least 3 per capability (see table 6.1)	++
	THU-M2.1	—	$\geq$ 3	at least the 6 ones outlined in this report	
	THU-M2.2	—	$\geq$ 1	7	
	THU-M2.3	—	$\geq$ 1	3	
THU-Q3	THU-M3.1	—	$\geq$ 4	mobile manipulation (order picking): TRL 4, navigation stack (Robotino): TRL 9	++
THU-Q4	THU-M1.2	one implementation per type of capability	$\geq$ 2 per type of capability	5 different capabilities each with at least 3 alternative implementations allows combinations by system builder (see table 6.1)	++
	THU-M2.4	many (even at source code level)	few (not at source code level)	none (just reuse and configuration of binary)	
	THU-M2.5	—	$\geq$ 10	22 RobMoSys components on Robotino, 20 RobMoSys components on Larry (mobile manipulation), 8 RobMoSys components for fleet coordination, core navigation components identical on all robots	
THU-Q5	THU-M1.2	one implementation per type of capability	$\geq$ 2 per type of capability	combinatorics available	++
	THU-M2.4	basically always restart with component builder	start at system builder	no need to go back to component builder (system composition/modification in less than 1 hour)	
THU-Q6	THU-M2.6	not systematically	$\geq$ 1	3 by THU + 2 by ITPs (RoQME, MIRoN)	++
THU-Q7	THU-M2.7	not demonstrated yet	$\geq$ 2	3 (ACE, DDS, OPC UA)	++
THU-Q8	THU-M2.8	—	$\geq$ 1	3 (rearranging navigation corridors, rearranging shelves, adding new steps	++
THU-Q9	THU-M1.3	homogeneous	diverse	diverse	++

Table 6.2: Benchmarking plan for goal THU-G1: The goal of this pilot is to demonstrate the benefits of RobMoSys.