SAARLAND UNIVERSITY

Faculty for Mathematics and Computer Science
Department of Computer Science
Masters's Thesis

Development of Language and Task Models for Skill-based Selection and Operation of Robots using Speech Dialogue

Reviewers:

Prof. Dr. Antonio Krüger
Prof. Dr. Peter Fettke

submitted by

Julian Wolter
Masters's Program in Computer Science
6th July, 2020

Supervisor:
Prof. Dr. Antonio Krüger, German Research Center for Artificial Intelligence,
Saarland Informatics Campus,
Saarbrücken, Germany

Advisor:
Dr.-Ing. Michael Feld, German Research Center for Artificial Intelligence,
Saarland Informatics Campus,
Saarbrücken, Germany

Reviewers:
Prof. Dr. Antonio Krüger, German Research Center for Artificial Intelligence,
Saarland Informatics Campus,
Saarbrücken, Germany

Prof. Dr. Peter Fettke, German Research Center for Artificial Intelligence,
Saarland Informatics Campus,
Saarbrücken, Germany

Submitted
6th July, 2020

Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Campus - Building E1.1
66123 Saarbrücken
Germany

Eidesstattliche Erklärung:

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath:

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, 6th July, 2020 __________________________

Einverständniserklärung:

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent:

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 6th July, 2020 __________________________

Acknowledgements

I sincerely thank Prof. Antonio Krüger for the opportunity to realize this thesis at DFKI. Many thanks also go out to my advisor Dr. Michael Feld, who has always supported me a lot and was always available with advice and action. Without him, this work would not have been possible. I would also like to thank Prof. Peter Fettke, for reviewing my work.

Moreover, I want to thank my friends and family who have supported me throughout the entire process. I want to especially say thank you to my girlfriend for correcting my linguistic mistakes. Furthermore, I thank the team of the MRK laboratory for the support on-site.

Also, I want to thank everyone at the DFKI who helped me at some point during my work. Be it giving practical advice, helping to evaluate the study or providing rooms to conduct it.

Last but not least, I thank every participant of my study.

Abstract

Industry 4.0 is happening. Robots are taking over more and more tasks, even in industries in which they have not yet been present. Products are becoming increasingly individual and are no longer mass products off the shelf, leading to a more customized production process. To produce high-quality products efficiently, human-machine collaboration is often essential, in particular the transfer of control from the human to the machine. Therefore, there is a continually growing need for intuitive and flexible control of robots without extensive training of the worker.

In my thesis, I develop an intuitive system for robot control with the help of natural language understanding (NLU) tools. For this purpose, I first abstract and summarize the many different abilities of various robots by creating a skill model. In addition, a task model defines which tasks can be solved and how. There are often multiple different ways to approach a task, and the most suitable alternative has to be chosen depending on the capabilities of the available robots. The task model can later be mapped to real robots using my skill model. Besides, objects in the environment are modelled and given properties to allow for interaction with them. Once the tasks, robots and objects are defined, the system creates a training corpus to train the NLU system and make it ready for use. An appropriate dialogue model now makes it possible to trigger a task intuitively by voice. To put it all together, I show in my work how to assign the generated task to the available robots with the use of the skill model. Finally, my system sends out high-level robot commands to fulfil the request of the user.

For evaluation purposes, I modelled the MRK4.0 laboratory in my work and showed some possible applications. The model allows representing all scenarios successfully, and the system distributed the tasks sensibly. I evaluated the intuitiveness of the language component with an exploratory user study. For this purpose, I created a simulated environment in which participants in the study should solve some predefined jobs. There was no training in advance. Most participants could solve all tasks after a small explorative phase in the beginning. In both cases, the system fully met expectations.

Contents

Chapter 1
Introduction

1.1 Motivation

The industry is under constant pressure to improve, optimise and develop. Be it the introduction of steam power, or from steam power through electrification to the present state of the art. The working conditions for the employees have improved unmistakable. Today, the shift towards *Industry 4.0* is unmistakable; digitalisation has arrived in the industry. Many jobs, as we know them today, will change in the future. This thesis focuses on one field of particular importance: the human-robot collaboration.

Nowadays, many industrial processes cannot be carried out without the help of robots, for example because of the weight of the components, the required force, the necessary precision or because it would merely be very tiring for workers. Robots are, therefore, already part of everyday life in many industrial environments. Today, however, most robots work on their own and not collaboratively with other employees. Thus, robots are usually separated from the workers by cages and do not directly interact with them. It would be more desirable to have humans transfer tasks to available robots or distribute subtasks among a group of them. This allows for more dynamic processes and quick adjustments to changing production requirements.

A major setback is that most robots are specially designed and programmed to do one task repeatedly. This is suitable for mass production without individualisation, but the market is developing towards individual products according to customer wishes. Besides, product cycles are becoming shorter and shorter, and innovations have to be incorporated more quickly into products, making more dynamic handling of production processes necessary. As a result, robots must also be ready to be used more and more flexibly.

Usually, only specially trained staff can reprogram robots and assign new tasks to them. Often, these interfaces are complex and It is not possible for workers in the factory to reprogram robots on the fly. In the future, however, it will most probably be necessary for a larger group of people to be able to repurpose a robot quickly, without highly specialized training. Only this will make it possible to satisfy the demands of a customer-oriented and highly individual production.

Figure 1: Human-Robot Collaboration with an UR Arm at a workbench.[1]

Also, it would already be helpful today to quickly assign new tasks to a robot, for example, if a worker is temporarily unavailable and the robot has to take over the job. This so-called rapid teach-in could already optimize many processes in factories today and adapt workflows more quickly to new requirements. The necessary prerequisite for this, however, is that the assignment of a task is possible for virtually every employee in the factory, and not only for specially trained staff.

Summarizing, the following current and future problems stand in the way of large-scale use of robots, especially in the context of Industry 4.0. Currently, robots do not work cooperatively with humans and are therefore locked in a cage. Since only specially trained staff can program the robots, the flexible assignment of new tasks is unfortunately also not possible at present.

If we look at the industrial sector and look at how the user-machine interaction has changed in recent years, we can observe significant progress. Particular attention should be paid to chatbot systems, such as *Siri* or *Alexa*, which have managed to create an intuitive, natural language interface that can be used by anyone without training. These chatbots can be extended via apps and allow anyone to perform tasks such as turning on the light, playing music or placing an order on the Internet. However, these chatbots are not aimed at the industry sector yet.

Since these chatbots solve precisely the problems that currently exist in industry 4.0 regarding robots, it would be promising to merge these two worlds, practically an *Alexa*

[1] Source: `https://fullcircle.asu.edu/faculty/exploring-new-frontiers-in-human-robot-collaborations/`

for the industry. I will present such a high-level solution in my thesis: A dialogue-controlled robot control system that collaborates with the worker via a task model. The worker can control the system using natural language, so he does not have to memorize exact commands to control the system.

Furthermore, the system should be able to adapt to new circumstances like the currently available robots quickly. For this reason, I am introducing a skill model, which is used by the system to assign the tasks to different robots. These skills also allow robots that have similar capabilities but are, for example, from different manufacturers, to be used equivalent to carry out the same task. Besides, alternative solutions with different skills can be represented using the task model.

The skill matching significantly increases operability, as the worker no longer has to worry about which robots are available or how to assign the task to them. The decomposition of the task into subtasks is done automatically and can be optimized by taking an overall view of the whole factory into account. In addition, the vendor lock-in is avoided, since thanks to the abstraction, any robot can be addressed.

A robust model in the knowledge base for the objects and robots allows for efficient commands and simple extensions on the current system so that a worker can quickly extend it with new task or robots. This allows a quick reaction to changes in the production requirements.

Figure 2: Human-Robot Collaboration wiht KUKA KMR iiwa robots in an industrial hall.[1]

[1] Source: `https://www.kuka.com/-/media/kuka-corporate/images/about-kuka/kmr_iiwa_kollaboration.jpg`

1.2 Research Questions

The first important aspect of my work is implementing the knowledge representation. What does the system need to know? How detailed should this knowledge or the skills be modelled? Is an ontology a suitable option to represent this knowledge? All other parts of my work are based on this knowledge, so a well-designed knowledge base is an elementary foundation. The first research question that has to be addressed is:

Which high-level abilities of the robot do we need to model, what knowledge about the environment does the system need and how do we represent this data?

After acquiring knowledge, the system needs a way to interact with the user. One central goal of my approach is to over an intuitive interface. As a possible solution, I choose a natural language understanding (NLU) system. But we still need some dialogue models, e.g. for back-questions. The NLU needs to be dynamically created from the knowledge base. Also, the dialogue system should not accept undefined values from the user and requires to verify the input against the knowledge base. Dialogue techniques such as back questions and clarifications must consequently be supported. This requirement leads me to the second research question, which is as follow:

How can we generate an appropriate robot control speech model utilizing NLU which respects the skills of the available robots and which dialogue style and commands should the system support?

As the last step, I will think about a suitable system architecture and about how the system should generate concrete high-level tasks from the user's voice input. It must finally assign the task to a suitable robot (or several robots), which is done by sending out high-level robot commands. So, my final research question is:

How should we design such an architecture that can generate and adjust a robot task based on the dialogue input, assign them to suitable robots and send commands to them?

Another concern in the context of my thesis is transfer of control from machine to human, which could be necessary if, for example, a robot fails for whatever reason. I will provide a interface to allow external software to handle this case. The issue of how high-level commands will be executed in the end by the robot is not part of the focus of my thesis.

1.3 Outline

To begin with, I will present the pieces of work I have investigated to answer my research questions. After discussing and comparing related approaches, I will give you a rough overview of the overall architecture of my system. In the following chapters, I will explain every part of the architecture in detail, beginning with the knowledge base and the skill model. Then we will have a look at how the system interacts with the user and how I utilize an NLU system to improve usability. I will also give you details about the dialogue model. After that, the last major topic is the generation, assignment and execution of actual tasks. I will explain how the system respects the different skills of individual robots and decide which robots could satisfy the user request. Subsequently, I demonstrate my system by modelling the MRK4.0 laboratory and show possible applications. To illustrate that the system is as user-friendly as desired, I will present the results of a user study on the language model. In the end, I summarize my findings and give an outlook for future work.

Chapter 2
Related Work

In this chapter, I will present some work related to my approach. To each research question I have searched for appropriate related work which I will present summarized here. In addition, I will compare the different approaches I presented and finally compare them with the goals of my approach.

2.1 Representation of Knowledge and Robot Skills

To begin with, I will cover the topic of the first research question, namely knowledge representation. The focus is on the representation of the robot, which level of abstraction is necessary and how we can integrate the task knowledge.

2.1.1 Denavit and Hartenberg Parameters

The need for a standardized way to describe kinematic properties and calculate possible positions existed for a long time. In 1955, J. Denavit and R.S. Hartenberg proposed a way to describe kinematic properties in their paper *A kinematic notation for lower-pair mechanisms based on matrices* [1]. Their approach is in use even today, for example in order to calculate forward kinematics.

They define in their paper a chain of pairs which represent the kinematic mechanism. Each pair can only have one angle of freedom. If a kinematic mechanism has a joint with more than one angle of freedom, it needs to be represented by multiple pairs recreating the original joint. For example, a joint with two degrees of freedom will result in two pairs. In order to represent the one degree of freedom a pair has a special transformation matrix is used.

[1] Public domain image from Wikipedia (*Link*)

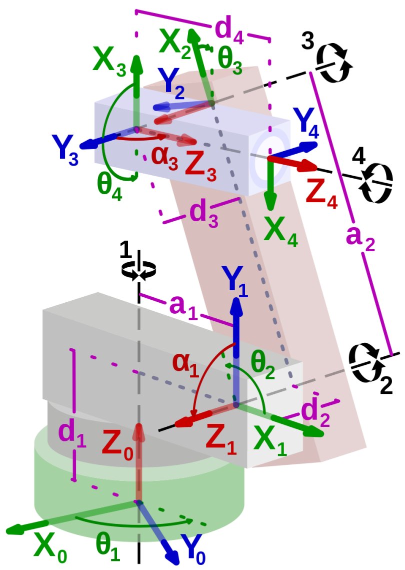


Figure 3: Robot with multiple joints, the corresponding coordinate system and DH parameters[1]

The chain of pairs has a strict ordering: the fixed element is the first link. The correct placement of the 3-dimensional coordinate system is described by precise rules (see Figure 3). All other pairs with their corresponding matrix are added to the chain in the order of their occurrence. Each pair's matrix is the product of two rotation matrices and two translation matrices to which the the Denavit and Hartenberg parameters are added, which results in 4 parameters ($\theta_n, d_n, a_n, \alpha_n$). The matrix is calculated by the following formula:

$$^{n-1}T_n = \mathrm{Rot}(\overrightarrow{z}_{n-1}, \theta_n) \cdot \mathrm{Trans}(0, 0, d_n) \cdot \mathrm{Trans}(a_n, 0, 0) \cdot \mathrm{Rot}(\overrightarrow{x}_n, \alpha_n)$$

Depending on which kind of movement a pair can do (revolution joint or prismatic joint) some parameters are static or variable, e.g. . Now these matrices of all pairs are added up together and form one large equation which describes the possible movements of a kinematic mechanism. The state is determined by the corresponding Denavid and Hartenberg parameters. By finding appropriate parameters, you can describe every possible position the mechanism can reach.

A major advantage of the approach is that it can completely describe lower-pair mechanisms by using matrix algebra which can be easily computed. Furthermore, a precise mathematical description of the robot in a standardized way is possible. The downside of this approach is that it is very low-level; there is simply no higher-level abstraction included, just bare, but important, mathematics. For my approach, like for most kind of other applications, this approach is also too low-level; I will need a much higher level of abstraction.

2.1.2 Unified Robot Description Format

One very common way to represent a robot is the Unified Robot Description Format (URDF). This representation was developed and is mainly used by the Robot Operating System (ROS) [2]. The main goal of this representation is to use it in order to simulate and to visualize a robot and to do kinematic calculations. Therefore, it is very detailed regarding the physical properties of the robot, e.g. which size robot have, what type of joints there are, how large the degree of freedom is and so on. But URDF only consider actuators of the robots, there is no information about sensors available in this format [1].

URDF uses Extensible Markup Language (XML) files in order to store the information. The file consists of two main parts: the joint elements and the link elements. The names already reflect very well which tasks these two parts have: every joint of a robot is represented by a joint element and every mechanic connection between joints is represented by a link element in URDF (see Figure 4). Every joint has a child and parent link. Like with the DH parameters, the origin fixed link of the robot is link 1. This definition of links and joints also leads to some limitations: URDF only supports robots with rigid links connected by joints, there is no support for flexible elements. Also, it is only possible to represent tree structures with URDF, so parallel robots like a Hexapod are also not possible to be represented.

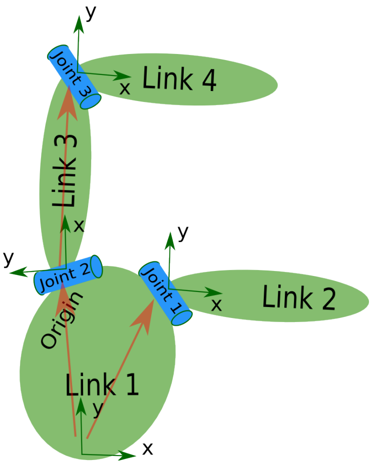


Figure 4: Relationship between links and joints in URDF[2]

All link and joint element have a lot of properties describing their mechanical properties, like size, limits and information to prevent collisions. Furthermore, you can name every element which allows to group them together, e.g. into some functional block like an arm. The problem is that if a robot consists of several identical parts, for example two mechanical identical arms with 3 mechanical identical fingers, each part must be written down explicitly again and again, which leads to a lot of duplication. In order to overcome this issue, URDF uses macro language called XML Macros (Xacro). This allows users to

[1] There is an implementation for sensors available, but is was never really used by ROS and support was dropped. *Source*

[2] Images by Willow Garage from ros.org under Creative Commons Attribution 3.0 license (*Link*)

write simple macros that will automatically generate the mechanical identical parts of the robot in order to avoid writing them down several times. Before passing the file to an URDF parser, the macros will be executed, resulting in a valid URDF file where all links and joints are written down explicitly.

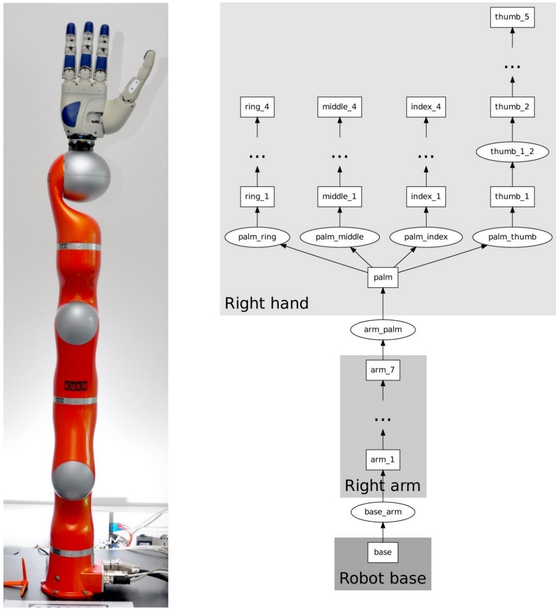


Figure 5: Left: KUKA LWR-4 arm with DLR-HIT hand
Right: URDF representation. Source: [1]

The significant advantage is that this format allows us to determine whether a specific movement is mechanically possible. For example, you can use inverse kinematics to find out how to reach a particular position or whether it is even possible to achieve it. You can furthermore determine whether or not the robot would hit something during the movement. As another advantage, this format can be used to simulate and visualize robots. Additionally, there already exists a feature-rich open-source toolchain. Last but not least, URDF offers some first steps in grouping joints and links together to a “higher-level” component by naming them properly, but there is no semantics of robot parts included. One concrete example of the KUKA LWR-4 arm with DLR-HIT hand in URDF is shown in Figure 5.

The disadvantage for me is that this format is still very low-level. There is no higher-level abstraction of the functions of the robot, just a bare physical description of the robot. It does not include any high-level knowledge about the reasons for any specific link or joint. You may know that the robot has two arms, but you know nothing about what exactly it can do with it. Another point is that there is no real layer of generalization available in URDF, you must rewrite mechanical identical parts every time or use the macro approach which was added later (and it feels a bit like a hack).

Although this approach offers more abstraction than the DH parameters, it is still too low-level for my purposes. I can now group joints and links halfway, but still don’t know what the robot is capable to do and URDF does not offer a good concept for generalization either.

2.1.3 KNOWROB - Knowledge Processing for Autonomous Personal Robots

In contrast to URDF, the next approach by Moritz Tenorth and Michael Beetz presented in their paper KNOWROB - *Knowledge Processing for Autonomous Personal Robots* [3] is a very high-level abstraction. Their goal was to develop a knowledge processing system that tells robots to do the right thing to the right object in the right way. They transfer already existing approaches in knowledge representation and reasoning to the context of robotic control. The problem they try to solve is that tasks are becoming more and more complex. For example, just the task to set up the table is highly complex and involves a lot of questions that need to be answered: Which objects need to stand on the table? Where does the robot have to be to grab a specific object? Which gripper should be used? And many more questions like this. It is not practicable to formulate everything down to the smallest detail for every case. Therefore, such tasks must be solved by general and flexible routines which adjust automatically to the particular situation at hand.

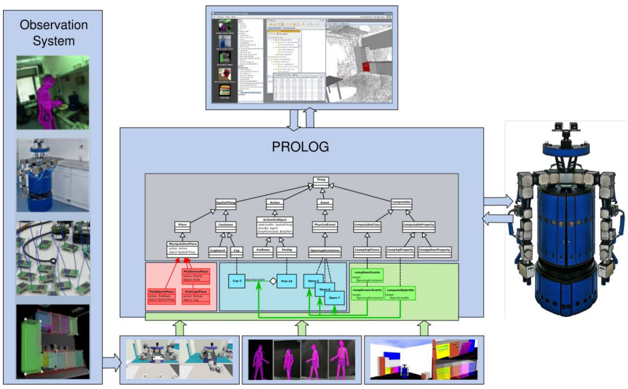


Figure 6: Overview of the architecture. Source: [3]

KNOWROB offers such routines by using the Web Ontology Language (OWL) in combination with Prolog for inference. An observation system consisting of different sensors acquires data which is pre-processed using computable predicates in order to make it accessible to Prolog. An overview of the architecture can be seen in Figure 6. The most interesting part for me is how the ontology is structured which is a first step towards my knowledge model.

The knowledge is represented via the description logic OWL, which leads to the fact that we have two main levels: classes and instances. The classes contain the abstract terminology like types, events or actions. The instances represent, for example, concrete objects, observed events or actions that could actually be performed. A first overview is given in Figure 7. The knowledge can be split into three big parts: action models (red), instances of objects, actions and events (blue) and the computable classes and properties (green). The blue block represents observed events, physical objects or performed actions.

The instances are created directly from the computable classes in the green block. This observed data is used by action models to generate new classes. In the example in the Figure, the subclass "ManipulationPlace" is created.

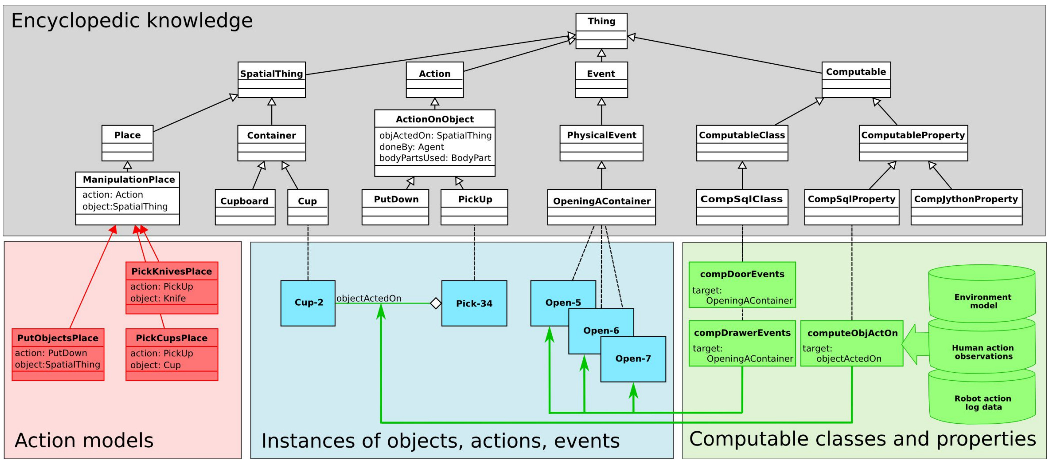


Figure 7: Details of the relations between classes (grey block) and instances (colored blocks) inside the ontology of KNOWROB. Source: [3]

One example is how the robot learns where it has to stand in order to grab a piece of tableware. A visualization of this process is given in Figure 8. The first reasoning step is to load the observed data into the knowledge processing system via the computable properties (see Figure 8 left). In the next step, data mining algorithms are used to aggregate clusters. In the last step, these clusters are used in decision tree learning to extract rules which map the action to a specific place. Of course, I have now simplified the process, but because the specifics are not relevant to my work, I will not go into further detail.

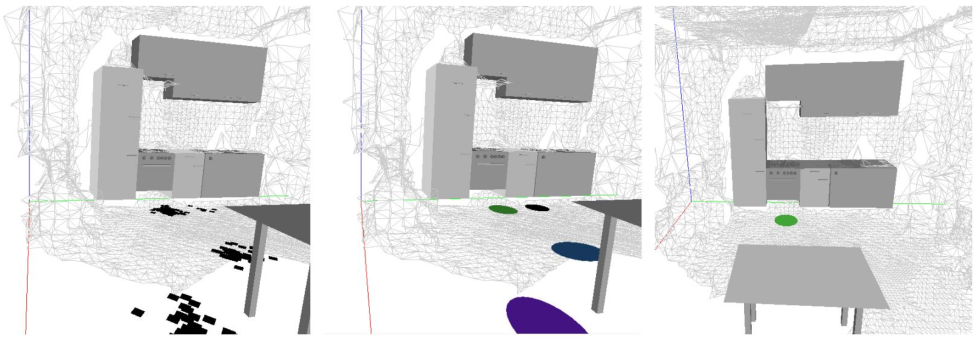


Figure 8: Left: observed positions for picking up pieces of tabelware
Middle: clustered positions into places
Right: result that has been learned. Source: [3]

This approach by KNOWROB shows us that a high-level abstraction is possible and can be very powerful. The focus for the authors was mainly in automatically learning and extending the knowledge and handling uncertainty which is not the focus of my thesis.

Furthermore, the concrete execution or binding of the actions to a robot got a little bit lost in their approach. However, this work gave me great food for thought on how an ontology can be structured.

2.1.4 Semantic Robot Description Language

Another approach for representing robots is the Semantic Robot Description Language (SRDL) proposed by Lars Kunze, Tobias Roehm and Michael Beetz in their paper Towards Semantic Robot Description Languages [4]. Their goal was to close the semantic gap between high-level actions like "Pick up the cup with the right hand" and the low-level robot description. Furthermore, they want to equip robots with some self-knowledge. Therefore, the robot itself can decide whether it is able to perform a given task and, if so, how it performs the task. As a consequence, SRDL can answer questions like "Can you set the table with cups and plates" and if so generate and appropriate lists of tasks that need to be done in order to fulfil the request. The generated plan refers to a concept like thumb or gripper, but not to specific links or joints. In order to achieve this goal, the authors first developed a semantic description language for robots, extended the action representation of KnowRob by linking to robot descriptions, and designed an inference mechanism for matching robots and actions.

The SRDL framework is integrated within the KnowRob knowledge processing system. This results in the fact that SRDL also uses OWL in order to model knowledge about robots, capabilities and actions and that Prolog is used for implementing an inference algorithm. The advantage of this approach is that SRDL can directly utilize the build in inference mechanisms of OWL for reasoning and re-use the KNWOROB's knowledge about actions with just some SRDL specific information about robot, components and capabilities. Furthermore, they established a system that automatically imports URDF into SRDL to save some repetitive work.

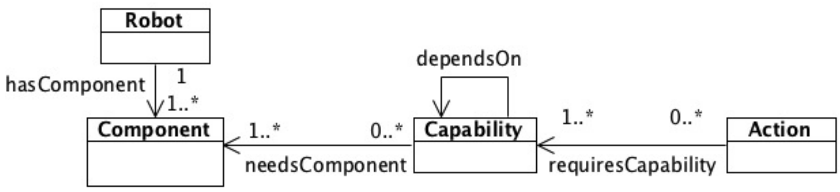


Figure 9: Relationship between the concepts of robot, component, action and capability. Source: [4]

The main concept of SRDL consists of four parts: Robot, Component, Action and Capability. A robot is a physical agent that performs actions, for example Baxter or the MiR platform is a robot. A component is a part of a robot, for example the "left arm" of Baxter. An action is defined following the definition of a PurposefulAction in OpenCyc (consciously, volitionally, and purposefully done by at least one actor). Finally, a capability is the ability to perform a certain action. Capabilities are assigned to components and components are assigned to robots. A graphical representation is given in Figure 9. In contrast to KnowRob, capability matching is not done via inference rules, instead the required capabilities are compared with the capabilities of a robots component. SRDL can match high-level actions to specific robots via the concept of capabilities (see Figure 10).

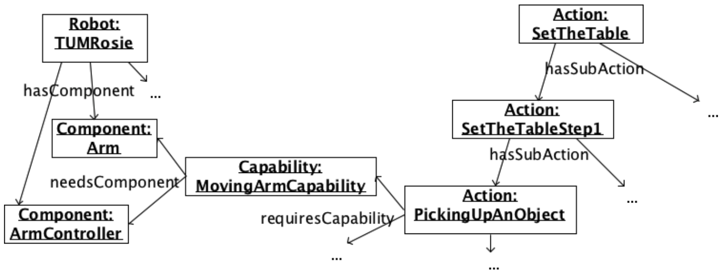


Figure 10: Overview of how components of a robot are matched to capabilities needed for a certain action. Source: [4]

Actions are stored inside a hierarchical action tree. These trees could consist of sub-actions and even these sub-actions could contain sub-actions. One example is given in Figure 11 for the action "SetTheTable" which contains sub-actions like “PickingUpAnObject” in the action tree. If the SRDL framework must prove whether a specific action is executable by a single robot, the algorithm will at first collect all sub-actions resulting in a large list containing all necessary actions. In the next step, the algorithm checks for each action if the robot is capable of doing the action with some component to do the action. If this works for all actions in the tree, including sub-actions, the robot can execute the requested action.

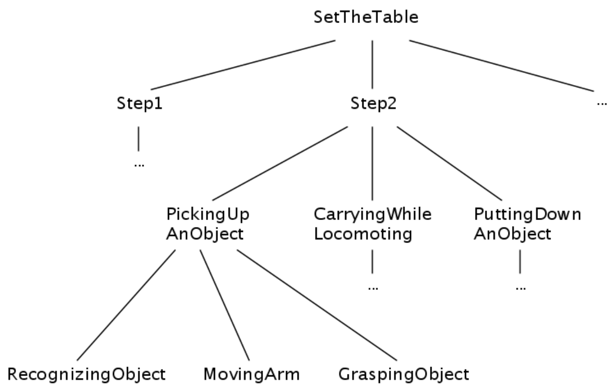


Figure 11: Example action tree for "SetTheTable". Source: [4]

This approach gives first impulses on what a task model for robots can look like, which is based on capabilities. The degree of abstraction is well chosen, so that with this model you can answer questions like "Can the robot lay the table" can be answered and the necessary steps can be generated. The only downside of this approach is that there is no support for a multi-robot scenario. It is not possible to represent a task that can be solved in different ways and human-machine collaboration was also not considered by the authors. But nevertheless, the paper gives some ideas worth thinking about.

2.1.5 RoboEarth Language

The last approach I will present regarding knowledge representation is by the authors Moritz Tenorth et. al. titled *The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments* [5]. Their goal is to make robot knowledge generally accessible, compared to user-generated content on the Internet. They want to build a network of robots that can mutually use each others' learned knowledge, such as tasks, objects and locations. Their work is based on the SRDL framework [4] which I introduced in the previous chapter.

Their idea is that much of the knowledge you gain about robots, objects and the environment is universal. Therefore, they want to reuse this knowledge instead of reprogramming it over and over again. For example, a glass is always a glass, a kitchen looks always similar and the task of serving a drink is usually solved in the same way every time. Therefore, they try to store this knowledge in a general form, so that other robots can access it without having to adapt this knowledge specifically for these robots. The central database that contains this knowledge is called RoboEarth. An overview of the architecture is given in Figure 12. Initially, the basic architecture was taken over from SRLD. As soon as the robot is assigned a task for which it lacks the necessary knowledge, be it knowledge about the task, an object or the environment, the RoboEarth database can be queried. If the knowledge is available there, it is sent back to the robot in general form. The robot can now ground and use this knowledge via inference rules for its specific platform, so the general knowledge becomes specific knowledge for exactly this case.

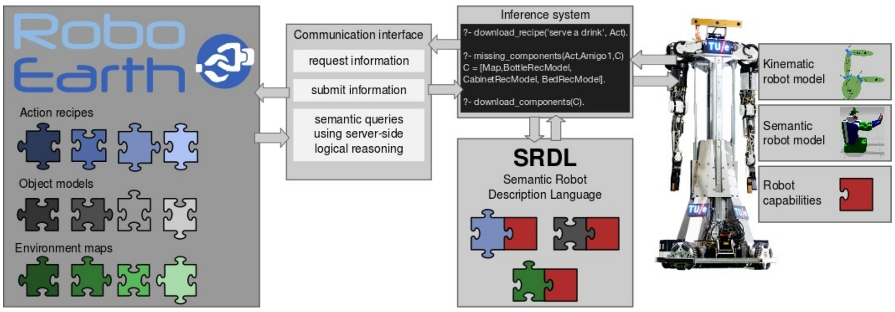


Figure 12: Overview of the architecture. Source: [5]

One big novelty is that unlike SRDL, capabilities are no longer directly determined by only those capabilities that directly connect an action to a component. Instead, abstract requirements are needed to accomplish the task. These requirements can be met directly by the robot's capabilities. However, there may be requirements that the robot has never heard of. Consequently, it cannot provide the ability to do this. However, the RoboEarth language allows requirements to be met not only by capabilities but also abstractly by, e.g. sensors. An example of this is given in Figure 13. The robot requests an action tree from RoboEarth for the task "FetchingWater". This takes 3 requirements: "EnvironmentMap", "Navigation" and "ObjectRecognition". The last two can be fulfilled directly by the robot through its capabilities, but the first one cannot. However, the robot also has several sensors. By means of an ontology, it can be determined that a sensor of the robot is able to provide the ability "EnvironmentMap". This means that the robot is ultimately able to perform the task based on the capabilities and requirements.

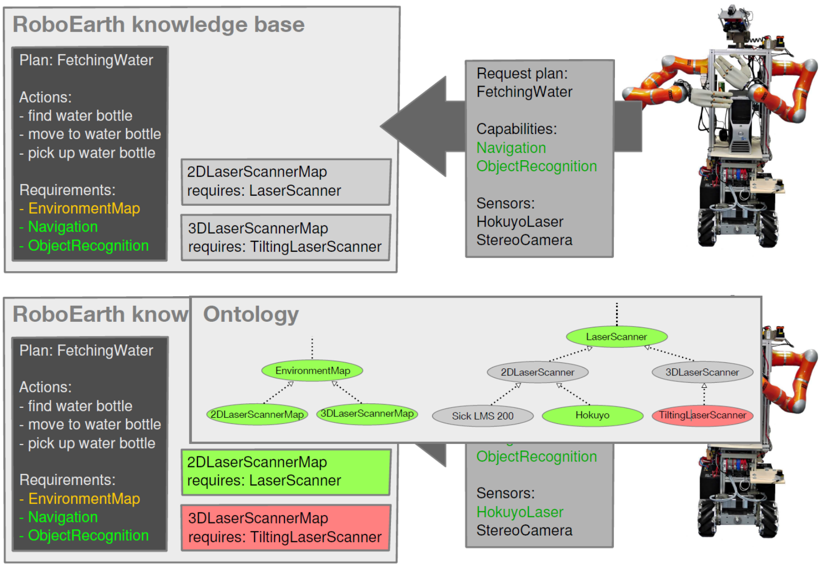


Figure 13: Top: Requested plan for "FetchingWater" from RoboEarth.
Bottom: Fullfilling the requirment of EnvironmentMap. Source: [5]

The great advantage of RoboEarth is the increased abstraction between the requirements needed to perform an action and the capabilities of the robots. These requirements come quite close to my model of skills, only that with my model, all skills are fixed from the outset and cannot be dynamically extended. A significant disadvantage is still that RoboEarth does not offer support for solving a certain task differently, e.g. because robots have different abilities which result in different solutions. Besides, human-machine collaboration or multi-robot scenarios were not considered here either. This work has made me confident that abstraction at the level of skills makes sense and that a meaningful match between robots and tasks is possible based on those skills.

2.1.6 Conclusion & Comparison

The papers presented cover a wide range of possible models for modelling robots and tasks. Some of them directly describe the real hardware, while others operate on a very abstract level. In the following Table 1 I have compared all works with each other and my own approach.

Paper	General				Represented Knowledge								Features				
	Representation	based on	Domain	Level of Abstraction	Detailed Robot Descr.	Abstract Robot Descr.	Single Robot Tasks	Multi Robot Task	Concepts for Assign. of Robot to Task	Objects in Env.	Places in Env. / Map	Computable Expressions	Determine if Task is executable	Assign Task to Robot	Transfer of Task Knwl. to oth. Robots	Execution Alternatives for Tasks	Prep. for integrating a Dialogue System
DH Param. [1]	Matrix		Inverse Kinematics	Low	✓								(✓)				
URDF [2]	XML		Inverse Kinematics	Mid.	✓								(✓)				
KNOWROB [3]	OWL		Knowledge Representation	High		✓	✓			✓	✓	✓	✓				
SRDL [4]	OWL	[2] [3]	Knwl. Rep. & Task Asgmt.	High		✓	✓		✓[1]	(✓)	(✓)		✓	✓			
RoboEarth [5]		[4]	Knwl. Rep. & Task Asgmt.	High		✓	✓		(✓)	✓	✓	✓	✓	✓	✓		
My approach	YAML		Knwl. Rep. & Task Asgmt.	High		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

[1] Knowledge about tasks is tightly bound to a specific robot and objects.

Table 1: Comparison of Related Work regarding Representation of Knowledge

2.2 User Interaction

In this chapter I will deal with the subject of human-machine interaction, mainly in the form of a natural language dialogue. First of all, I will show two related works that show how to create a dialogue model with the help of knowledge models. Then I will show which interaction models are well suited to make a robot accessible to a human via natural language by looking at already implemented architectures. I am particularly interested in what possibilities are offered to the user and how these are implemented.

2.2.1 SpeechPA: An Ontology-Based Speech Interface for Personal Assistants

Emerson Cabrera Paraiso and Jean-Paul A. Barthès built an ontology-based speech interface for personal assistants which they presented in their paper *SpeechPA: An Ontology-Based Speech Interface for Personal Assistants* [6]. Their personal assistant, which they call SpechPA, is suitable for a specific domain. For this they use a set of tasks and the domain ontologies in order to handle knowledge and separate the models for reasoning. They can also use their ontology in order to improve the user interaction. In their example, they implement a Personal Assistant that helps them to fill out official forms.

Figure 14: Left: Excerpt of the domain ontology.
Middle: The concept of RMI and its attributes.
Right: Possible actions. Source: [6]

First of all, I want to introduce the knowledge model of SpeechPA. An ontology separated into two parts is used for the representation: a set of task and a set of domain ontologies for reasoning. The main use of the ontologies is to interpret the context of utterances by the user and to keep a computational representation of knowledge, e.g. for inference. On example for knowledge representation is given in Figure 14. In this example the concept allowance is shown with 3 sub-concepts (left). These sub-concepts, such as RMI, can have attributes (middle). Concepts can also be equipped with actions that can be triggered by the user (right). Furthermore, there is a task model in the ontology, as you can see in Figure 15.

Since SpeechPA uses natural speech, a commercial speech recognition is used first. How-

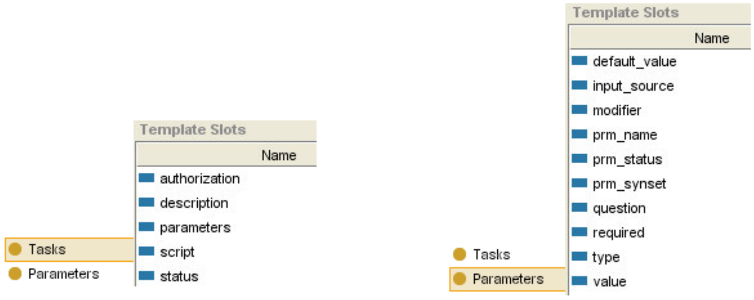


Figure 15: Left: Parameters of a task.
Right: Fields of the parameters. Source: [6]

ever, the results are not perfect, for example due to noise or poor pronunciation. Therefore, the utterances may contain lexical or syntactical errors. First, such errors are ignored and a list of disfluencies is extracted. The resulting utterances are processed in two steps: first parsing and syntactic analysis and in the second step the application of the ontology. First, each utterance is assigned to a syntactic category (verb, noun, etc.). Afterwards, they try to assign the statement to one of the three following categories: order, question or answer. To determine the domain more precisely, on the one hand the ontology is used, but also a lexicon which has been enriched with WordNet [7].

After the domain has been determined, the parser returns a matrix with the tokens and their syntactic category. These tokens are looked up in an ontology, where further analysis procedures are applied, like Trigrams, to evaluate word similarities [8]. Then, for example, a task is found that matches the request of the user, the transferred tokens are assigned to the corresponding parameters and the task is initialized. If a task is started and not all parameters are provided, the system automatically requests the missing parameters from the user. If a request cannot be answered because it is outside the domain or the system is unable to do so, be it because knowledge is missing or because no suitable task is available, the user is prompted to make another request.

An example dialogue is given in Figure 16.The first question of the user has been answered by utilizing the knowledge inside the ontologies. The next two questions could not be answered by the system, the first problem is that sending an email is not inside the domain of the personal assistant, the second question could not be answered because the system did not have any knowledge about foreigner receiving RMI. The last request of the user triggers the task to fill out an RMI application. The system queries the required parameters from the user.

An exciting aspect of the system is the separation of knowledge into task and domain knowledge. Besides, the use of lexicons and syntactic analyses allows the user a certain degree of freedom in formulating their request. However, this processing is very complex and the paper also mentions that semantic interpretation is also a challenging part. Taking these factors into account, a modern NLU system is probably more comfortable to use and should deliver at least equivalent results which are why I'm going to use one of these for my work. But the approach works very well in their specific domain, the separation

1	PA: Welcome to Voice Assistant.
2	USER: List all conditions to receive an RMI.
3	PA: The candidate must: live in France, be older than 25, have income not higher than 417,88 € per month.
4	USER: Send an email to John
5	PA: I cannot do what you want. Do you want to know what I can do?
6	USER: May a foreigner receive an RMI?
7	PA: I cannot do what you want. However I can: List all conditions to receive an RMI. These are tasks related to RMI.
8	USER: Start an RMI application.
9	PA: What is your age?
10	USER: 26.
11	PA: What is your monthly income?
12	USER: 300 euros.
13	PA: Where do you live?
14	USER: ...

Figure 16: Example dialogue with the SpeechPA. Source: [6]

of knowledge and tasks seems reasonable; the interaction as a personal assistant with the classification of request into the three categories is very intuitive. In summary, it can be said that the system provides useful functional approaches.

2.2.2 Ontology based Chatbot (For E-commerce Website)

The next approach was made by Anusha Vegesna, Pranjal Jain and Dhruv Porwal in 2018 and published by them in the paper *Ontology based Chatbot (For E-commerce Website)* [9]. They build a chatbot for a specific domain, in that case for an e-commerce site, that can answer questions about products of users. The approach is special in that they use many already existing services and integrate their system into already widespread platforms.

Your system consists of three parts: the knowledge base, the ontology and the dialogue manager. For the knowledge base, free APIs are used in the area of e-commerce, such as eBay. This data is processed and finally stored in a MongoDB where it is available online. Ontology was implemented using Protégé [10] as a platform. It gives an initial overview of basic vocabulary and expressions along with their relationships. Last but not least, they use api.ai as a dialogue manager. This platform supports NLP in order to offer an intuitive humane machine interaction. Using annotated example sentences, intents are trained that can later be recognized and interpreted by the platform. An overview of api.ai is given in Figure 17 on the left side.

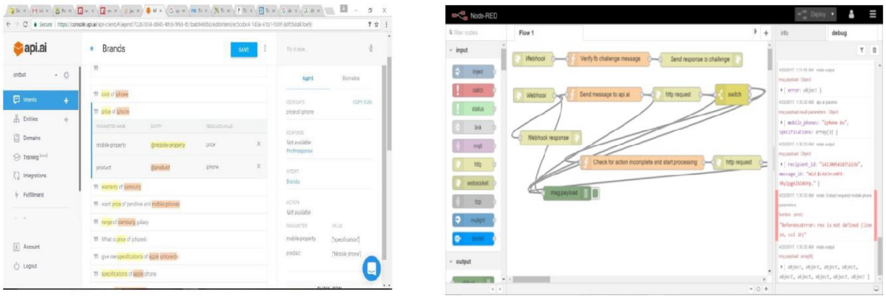


Figure 17: Left: api.ai for NLP, annoted sentences
Right: NodeRED used for dataflow control. Source: [9]

The different components of the system are connected via a NodeRED instance, see Figure 17 on the left side. The Facebook chat, for example, is connected via this instance, so that the system can be reached via Facebook. If the system is now asked a question, it first tries to recognize the intent of the user via api.ai. Then the ontology is used to establish the connection between entities of the NLU and properties of the knowledge base. If the requested information is contained in the knowledge base, it is communicated to the user. An example dialog is shown in Figure 18.

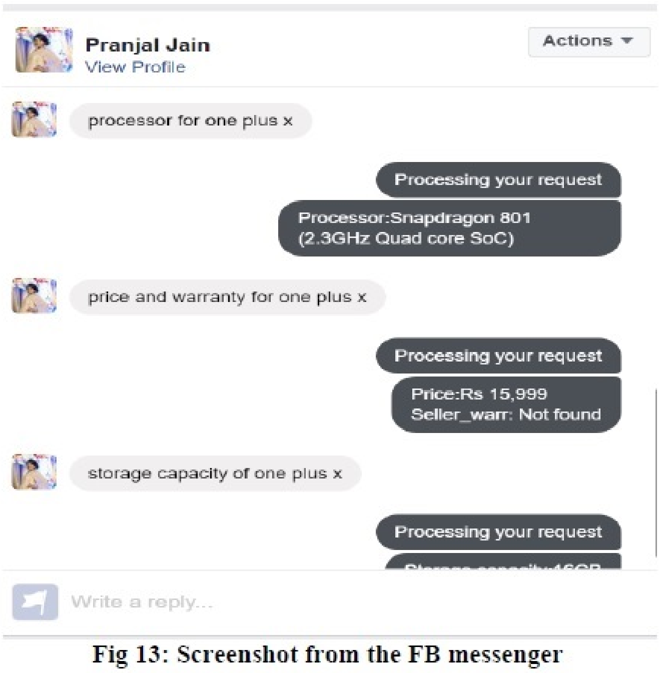


Figure 18: Example dialogue with the Ontology based Chatbot. Source: [9]

The important benefit of this approach is that it uses many existing technologies to build a fully functional assistant. By utilising an NLU system, it also allows processing of natural language without extensive analysis, but rather by just specifying annotated example sentences that have to be created once. Through studies, these sample sentences can be extended and the system benefits directly. The separation of knowledge and relationships also allows the knowledge to change dynamically during runtime without having to adapt anything in the dialogue model.

For my purpose, I will also use an already existing NLU platform so I can focus on what's most important in my work. Additionally, I will also introduce a separation between knowledge and dependencies, because especially in the industrial domain, the environment can change very quickly, for example, a robot breaks down or there are now twice as many special tools. My system should be able to absorb these changes without having to throw the complete dialogue model overboard.

2.2.3 The Robot Voice-Control System with Interactive Learning

The next approach ranges in the domain of robotics and is developed by Miroslav Holada and Martin Pelc in their paper *The robot voice-control system with interactive learning* [11]. Their goal is to make a robot not only intuitively accessible via speech, but also to make it possible to program the robot via speech. In addition, their approach includes basic image processing so that the system can detect basic objects and colours. I will focus more on user interaction in my summary and not go very deeply into the architecture as this chapter is not so relevant to my work and it is kept relatively straightforward.

The scenario consists of a table on which different geometric objects in different colours are placed. A robot arm can manipulate these objects. A picture of the scenario is given in Figure 19. Via speech, a user can now control the robot and even program new commands into it.

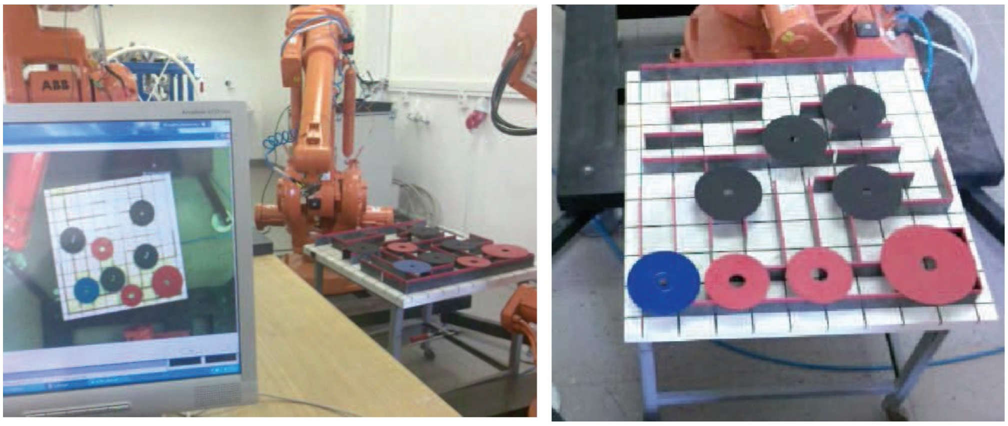


Figure 19: Scenario of the Robot Voice-control System. Source: [11]

The interesting part is how the dialogue is modelled. In principle there are four vocabularies which are divided as follows:

- Simple basic control commands like "take it", "move up" or "stop" that are necessary for basic robot control. There may be synonyms.
- Unused words and short phrases used for new actions like "search" or "disks". This is the biggest vocabulary.
- Name of the robot actions taught in by the user.
- Build in static activities like "robot calibration" or "learn new command".

This simple subdivision of vocabulary allows the user to control the robot very easily and teach new commands to it. The subdivision of the tasks into small tasks allows a fine-grained control. If the user now wants to program a new command, he first says the signal word for creating a new one. After that, every control command of the user is recorded until the user finishes the recording. Last but not least, the command is given a name. The command sequence can then be called again under this name at any time. An example dialogue is given in Figure 20, where a user teach-in the command that searches for disks and moves them on the table.

"Start recording new command." ...this is operator's command (italic text)...
"I'm recording" ...the system says (bold text)...
"Search black disks"
"I'm searching ... Four disks were found"
"Move on first"
"I'm moving ... Done"
"Take it."
"Ok"
"Move on position alpha."
"I'm moving ... Done"
"Put it"
"Ok"
"Stop recording."
"I stop the recording. Please, say new command"
"Search" "Disks" "Done"
"New command is entered and named: Search disks. Is it right?"
"Yes"
...now, the newly defined command may be used...
"Repeat command"
"Enter command"
"Search disks"
"OK"
...now the system repeats the command until no more disks are found...
"No object found. Repeating done."
...now all the disks on the scene are transported into position alpha...

Figure 20: Example dialogue utilizing interactive learning. Source: [11]

The authors say that their system is very accessible, even to people who have no experience with robots or programming which is a great achievement. The breakdown of large actions and commands into small atomic actions works very well and allows for great usability. Therefore, I will choose a similar approach and, among other things, offer the user this smallest unit of action. However, there are also drawbacks, for example, the flexibility of a command is very limited, more complex commands cannot be created. In addition, there is only one category of tasks, multi-robots or human-machine scenarios are challenging to implement, and there is almost no error handling.

2.2.4 Human-Robot Dialogue for Joint Construction Tasks

Another interesting approach is created by Foster et al. in their paper *Human-Robot Dialogue for Joint Construction Tasks* [12]. The new idea here is to consider robots and humans completely equal. The resulting symmetry leads to new challenges regarding the dialogue system and opens up new ways to create a construction plan. Their scenario is that a human collaborates with a robot to process a construction plan. An overview is given in Figure 21, where you can see the robot used called *JAST construction robot*(left) and the Baufix airplane to be built (right).

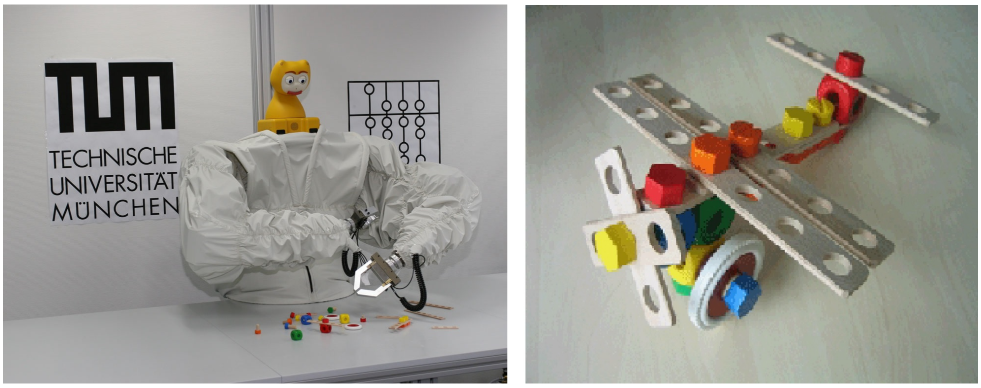


Figure 21: Picture of the *JAST construction robot* and the airplane. Source: [12]

Traditionally, humans stand above robots and are not equal to them. Therefore, the most common form of interaction is teleoperation which means that the human gives commands to the robot. Since robots become more and more autonomous and can work for the most part without human intervention, the human only acts as a supervisor in the event of an error. In almost all cases these two forms of interaction are sufficient, but robots and their tasks are becoming more and more complex, which is why a radically different approach is chosen, in which man and machine are completely equal, e.g. the robot could also ask the user to hold an object or assemble some components. An example of such an interaction is given in Figure 22. First, JAST takes the initiative and asks the user to build something. JAST supports the user by providing him with the required components.

The main focus of the dialogue is not on what should be done, since both man and machine know the goal. Rather, it must be clarified who takes on which sub-task. Here, too, natural language is used as the interaction means of choice. The OpenCCG [13] natural-language-processing library is used for this purpose. In addition, OpenCV is

JAST: Welcome to JAST. Would you like to build a tail section?
User: Okay.
JAST: Can you take care of the bolt and slat?
User: Sure. *[Picks up a red bolt and a five-hole slat]*
JAST: *[Picks up a red cube]* Tell me when you are done.
User: *[Puts bolt through slat]* I'm done.
JAST: *[Gives user the cube]* Here is the cube.
User: *[Takes the cube]* Thanks! *[Screws cube onto bolt]*

Figure 22: Example Interaction. Source: [12]

used for object and face recognition. Feedback is also given in the form of speech, but the robot can move its arm instead of speech, for example, which gives the system a certain multimodality. The robot is controlled by action primitives [14]. These are parametrised, chainable motion specifications that divide large actions into small subtasks, similar to atomic tasks. For example, the task to grasp an object will be divided into a sequence of subtasks like detect the object, open gripper, move arm towards the object and so on.

The paper gave a lot of new insights, especially on which role the robot in my system should have. This helped me to determine the position of the robot within my system and also to decide on what kind of interaction between man and machine is most effective and intuitive. In addition, an NLU system was successfully used and larger tasks were broken down into smaller atomic tasks, which encourages me to follow this approach further.

2.2.5 Learning to Interpret Natural Language Commands through Human-Robot Dialog

Since NLU has obviously become an essential part of the work in order to provide an intuitive voice interface, the question on how to best train this system arises. An answer to this question can be found in Jesse Thomason et al.'s paper *Learning to Interpret Natural Language Commands through Human-Robot Dialog* [15] from 2015. They built a dialogue agent who understands human instructions through semantic parsing, who can actively solve ambiguities through a dialogue manager, and who can learn new words from conversation in order to improve performance.

Many existing natural language understanding approaches plainly use simple keyword search or large training sets of hand-annotated data, so with traditional NLU the detection is often weak or a lot of work is necessary to craft good training data. Jesse Thomason et al. chose a different approach, his dialogue agent integrates a semantic parser that generates a logical state from the utterance of the user. This logical state is handled by a dialogue manager which maintains a belief-state of what the user wants the system to do. An overview of the architecture is given in Figure 23 (left). At the beginning, the system will be given only a few examples. As soon as the agent understands a user, previous utterances of the user that were not understood are linked to this goal, so that new training sentences are generated by errors for the semantic parser.

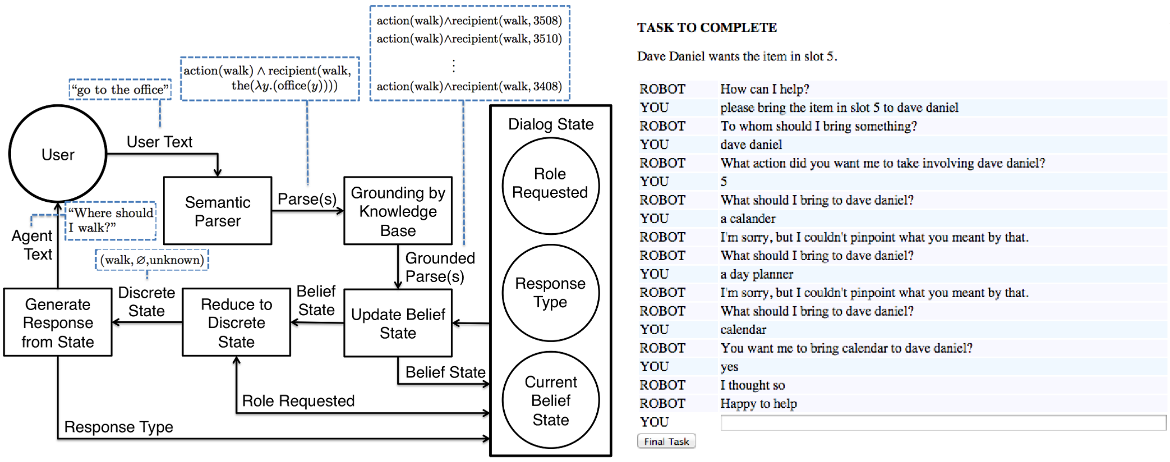


Figure 23: Left: Architecture while processing "go to the office" (dashed boxes)
Right: Example dialouge learning new synonyms. Source: [15]

An example of such a process is given in Figure 23 (right). In the beginning, the system does not know that *"day planner"* and *"calandar"* both just mean "calendar". After understanding the user, the first two utterances will be handled by the semantic parser as synonyms for *"calendar"*. This approach of incremental learning is promoted by the authors as more robust than the keyword search and requires only minimal initial data.

The remaining parts of their architecture is quite similar to already presented works, also they use CCG and the λ-calculus in their semantic parser. Furthermore, a knowledge base is used to ground references. Since I will not adopt the learning architecture shown here, I will not go into further details.

Finally, a study was conducted to evaluate the system. Users were asked via Mechanical Turk[1] to complete three tasks (navigation, delivery and validation) with the system and then to participate in a survey (rating 0-4). The tasks were split into train and test sets (20%/80% train/test split). In every batch, 48 persons conducted the experiment. Some of the results are given in Figure 24. After each iteration the system became better, which the

[1] Amazon Mechanical Turk (MTurk) is an Internet crowdsourcing marketplace to assign tasks to individual persons. So called *Human Intelligence Tasks* (e.g. identify objects in a picture) are posted there and crowdworkers will complete them in exchange for a rate

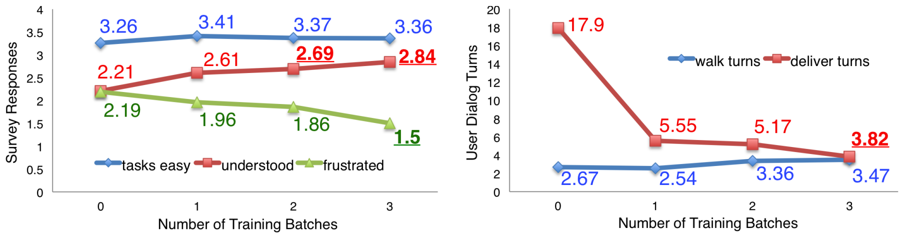


Figure 24: Mechanical Turk survey results (underlined bold differ significantly ($p < 0.05$) from batch 0)
Left: Mechanical Turk survey average responses.
Right: Mean user turns with correct goal reached. Source: [15]

surveys confirm. The satisfaction with the simplicity (*task easy*) and the understanding increased and the frustration decreased. In addition, one can see very clearly that the number of turns needed to understand the user's goal decreased significantly.

This work makes clear how important a well-trained NLU is, as it saves the user many unnecessary turns in dialogue and frustration. They furthermore show that the turn rate can drop significantly after only one iteration of learning. This is one of the reasons why I will evaluate the user voice interface of my work later. All in all, this is a very nice approach that can possibly also be adapted to classical and commercially available NLU systems.

2.2.6 Conclusion & Comparison

For this research question I have studied a range of different papers, some of them from different areas of research. Finally, Table 2 compares all the papers considered.

Paper	General			Grounding				Accessible Robot Tasks					Dialogue Features		
	Scenario	Modality	utl. Knowledge to gen. Diag. Model	Location	Directions	Objects	Discourse Memory	Navigation	Localisation	Manipulation	User-defined Tasks	Complex Tasks	Utilizing NLU	Further Inquiries	Status Updates
SpeechPA [6]	Personal Assistance	Written Lang.	✓											✓	
OntBot [9]	Answering Questions	Written Lang.	✓			✓							✓		✓
Robo Voice [11]	Robot Ctrl. & Task Teach-In	Speech	✓	✓	✓	✓	?	✓	✓	✓	✓			(✓)	✓
Joint Tasks [12]	Solving Tasks Cooperatively	Speech	✓		(✓)	(✓)			✓	✓	✓		✓		✓
Learn NLU [15]	Learning nat. Lang. Cmds.	Speech	✓					✓	✓	✓	✓		✓	✓	
My Approach	Robot Control in Industry	Speech	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Comparison of Related Work regarding User Interaction

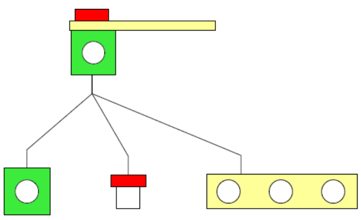
2.3 Task Representation, Generation and Assignment

In the last section, I present the related work regarding the task model, organization and generation. I also look at how tasks can be assigned to individual robots and how autonomously robots should perform such tasks.

2.3.1 Following Assembly Plans in Cooperative, Task-Based Human-Robot Dialogue

First of all, I look at another paper by Foster and Matheson with the title *Following Assembly Plans in Cooperative, Task-Based Human-Robot Dialogue* [16] which is based on [12] I presented earlier. In contrast to the previous paper, this one focuses mainly on the architecture, on how such a system is structured, and on how building plans can be stored and processed.

For their approach they chose AND/OR graphs [17] as the representation for the assembly plans. This graph is a directed acyclic graph consisting of two kinds of nodes: AND node and OR node. For an AND node all children must be fulfilled, for an OR node only one child must be fulfilled. An assembly plan consists of several sequences of assembly steps that are represented in such a graph. An example for an assembly step is given in Figure 25. In order to be able to divide the steps meaningfully, each step must fulfil the following 3 criteria: only one screw may be used, only one threaded fastener may be screwed onto the screw and any number of unthreaded pieces may be used. Some additional information is also provided, e.g. in which hole the screw should be inserted or the alignment.



Bolt b_1(bolt, small, red)
Insert list $[i_1$(slat, three-hole)$]$
Fastener f_1(cube, green)
Details $[$hole$(i_1) =$ **Middle**, direction$(f_1) =$ **South**,
direction$(i_1) =$ **South**$]$

Figure 25: Visual and symbolic description of an assembly step. Source: [16]

Since there are several possibilities for building a finished object, the assembly steps are finally arranged in an AND/OR graph, as you can see in Figure 26. The system can use two different ways to explain the assembly plan: depth-first or top-down. An important architectural building block is the object inventory. It stores the currently available objects and the construction progress at any time. As soon as an assembly step has been successfully executed, the state is updated: objects required for the step are deleted and the built object is added.

A big advantage of their approach is that the building plan is not fixed in a certain way, but offers several possibilities to reach the goal. Such a concept is also important for my work, because I have to solve the same tasks with different robots, where different

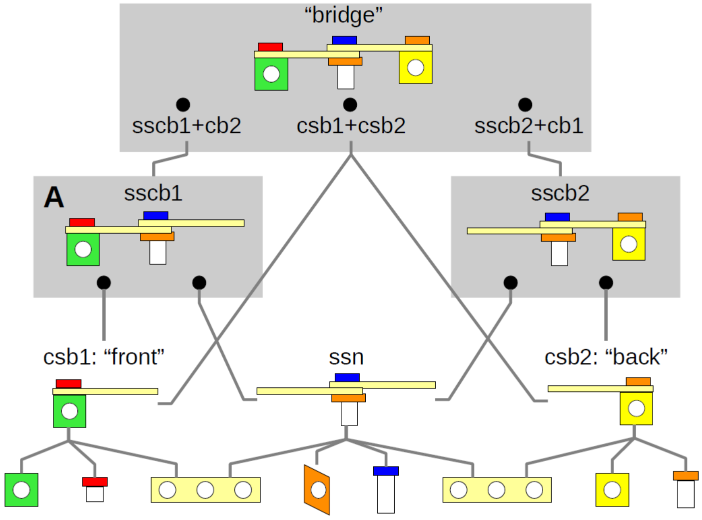


Figure 26: AND/OR graph example. Source: [16]

robots solve the same task in a different way. In addition, the representation as an acyclic graph is very useful and allows it to be easily processed later, e.g., for speech interaction. Therefore, I will also choose a graph representation for my approach and offer a possibility similar to AND/OR graphs to map alternative solutions for a given task.

2.3.2 Interruptable Autonomy: Towards Dialog-Based Robot Task Management

Another publication where I would like to focus more on architecture is the paper *Interruptable Autonomy: Towards Dialog-Based Robot Task Management* [18] by Yichao Sun et al. The authors' goal was to build a system of mobile service robots that would bring or pick up objects in an office complex. Tasks can be assigned to the robots by voice, as shown in Figure 27, for example. It turned out that the robots were too autonomous and therefore the architecture had to be adapted so that the robots could react more flexibly to the users.

USER:	Go to the meeting room.
COBOT:	I am going to "the meeting room", Room 7405, is that correct?
USER:	Yes.
COBOT:	Going to "the meeting room", Room 7405.

Figure 27: Example dialogue. Source: [18]

After a command has been transferred to the system via voice or web interface, it is planned by a central task management system, the so-called scheduling agent. However, it often happened that a task did not have to be executed completely, e.g. because the person was already met in the floor and the robot no longer had to drive to the office. It could also be that the user wants to give a passing robot a new task. As another potential problem, it can happen that due to obstacles the robot cannot reach its destination, e.g. because a suitcase blocks the passage. All these events require an architecture that dynamically reallocates the tasks. Therefore, the authors used the architecture shown in Figure 28, with a central planning component.

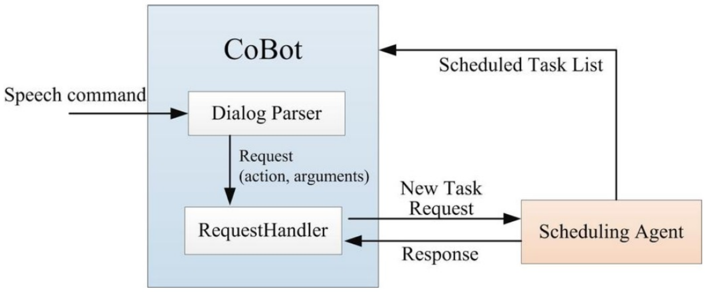


Figure 28: Architecture overview. Source: [18]

This central component allows you to re-evaluate the assignment of the tasks at any time and redistribute the tasks if necessary. The main focus is on time-constraints so that users do not have to wait too long. In order to calculate the times, the current position of the robot and the information from the navigation system are essential. In addition, care is taken to ensure that a user is informed in the event of an error, e.g. if a robot has got stuck. As an example, it can happen that a robot, which actually wants to drive to room A, gets a new task on the way and, due to time-constraints, first drives to room B to complete this new task and only then continues its way to room A.

Figure 29: CoBot-2, an indoor mobile robot. Source: [19]

The work in this paper is based on an earlier paper by Brian Coltin, Manuela Veloso and Rodrigo Ventura titled *Dynamic User Task Scheduling for Mobile Robots* [19]. Both papers used the robot shown in Figure 29.

2.3.3 SiAM-dp / step-dp Task Model

DFKI also has several solutions to represent task models and connect them with speech. One solution is SiAM-dp [20] which was initially developed by Robert Neßelrath. In the course of time, SiAM-dp was extended with many functions, including a powerful task model. Magdalena Kaiser, for example, has provided such a task model which was extended by Vanessa Hahn with user backchannels [21].

SiAM-dp is based on the Eclipse OSGi framework and is a platform for multimodal dialogue systems. Most activities can be carried out via a graphical interface. In the following I will focus more on the task model functionality.

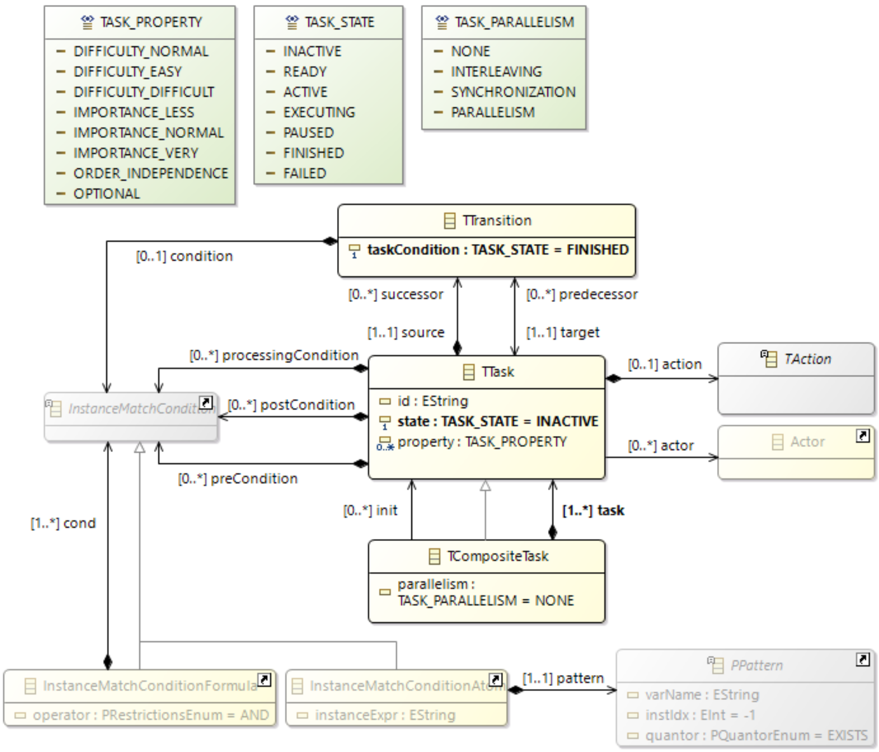


Figure 30: Task model used by Vanessa Hahn. Source: [21]

Vanessa Hahn has based her work on the concepts of the MBUI [22] task model. The resulting model is shown in Figure 30. Among other things, attention was paid to the correct modelling of the parallel execution of tasks. A clever hierarchy allows individual tasks to be combined into larger entities. This makes it possible to represent even complex tasks and hierarchies. In order to be able to react dynamically to different events, there are also pre- and post events.

Another DFKI dialogue software, which also has task model functionalities, is step-dp. The task is defined directly in JAVA code for which various pre-built function blocks are available. This model can then interact with the dialogue, for example.

What all these task models have in common is that they were developed in the context of dialogue. Therefore, the focus of the development was on how to best integrate these models into a dialogue and not on how to control and manage different robots with them. Furthermore, these models do not have any planning or assignment functionality regarding robots. In addition, these task models can only be broken down in a certain way and cannot be dynamically adapted to different robots.

2.3.4 A Comprehensive Taxonomy for Multi-Robot Task Allocation

Now that I have taken a look at the representation and practical implementation of task models, I will give some details on the planning and allocation. A suitable paper on the subject is *A comprehensive taxonomy for multi-robot task allocation* [23] by G. Ayorkor Korsah, Anthony Stentz and M. Bernardine Dias. In it, they present a taxonomy for *Multi-Robot Task Allocation* (MRTA) problems. The task allocation specifies which robot should do which task in which order at which time in order to achieve the overall system goal, which can involve a lot of requirements. In addition, they give a small overview of existing concepts and finally provide possible solutions for the planning.

The paper starts with a short introduction to the Gerkey and Matarić categorization of *Multi-Robot Task Allocation* (MRTA) problems. This now widespread taxonomy consists of three axes, which are structured as follows: the task type, the robot type and the allocation type. A graphical representation is given in Figure 31. What is important to know is that only ST-SR-IA problem is solvable in polynomial time, all other problems are NP-hard.

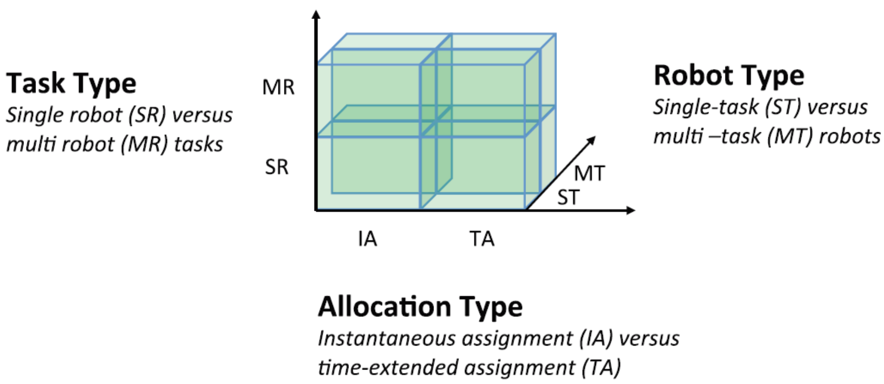


Figure 31: Three axes of Gerkey and Matarić's taxonomy. Source: [23]

Next, the Zlot's task types [24] are introduced as shown in Figure 32. This terminology is adopted by the paper. There are some important definitions, which I will summarize in the following:

- A task t is called **decomposable** if it can be represented by a set of subtasks σ_t. These subtask satisfy some specific combination ρ_t of σ_t which satisfies t.
- A task is **multiply decomposable** if there exists more than one possible decomposition.
- An **elemental task** is a task that cannot be decomposed.
- A **decomposable simple task** is a task that can only be decomposed in elemental task. There exist no multi-agent allocatable.
- A **simple task** is either an elemental task or a decomposable simple task.
- A **compound task** can be decomposed into a set of simple or compound tasks. There is only **one** full decomposition.
- A **complex task** is like a compound task which, in contrast, has several possibilities of decomposition. There can be multi-agent allocatable subtasks.

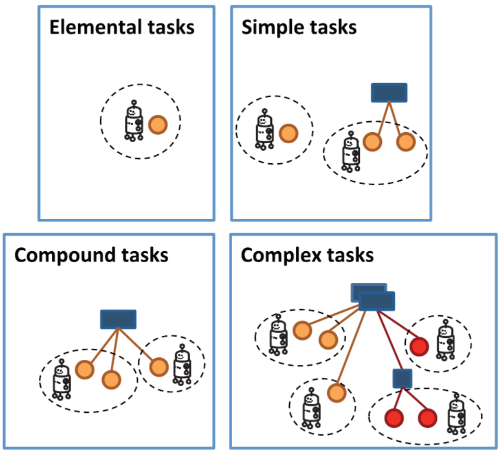


Figure 32: Illustration of Zlot's task types. Source: [23]

These task types allow to describe variations of single or multi robot tasks with several variants in execution and synchronization. For example, if you have only one simple task, there is only one solution with one robot involved. However, if it is a complex task, there are several solutions which can involve different robots. In my later work I will also describe a task model, which has similarities to the one presented here.

The paper then introduces the *iTax* taxonomy, which is based on the different dependencies within the task assignment. The four high-level classes are shown in Figure 33 and are defined as follows (simplified):

- **No Dependencies (ND)**
  Just simple or compound tasks that have independent agent-task utilities.
- **In-schedule Dependencies (ID)**
  Like ND, but the agent-task have intra-schedule dependencies. So one agent have to perform tasks in a given order.
- **Cross-schedule Dependencies (XD)**
  Simple or compound tasks that also depend on the schedules of others.
- **Complex Dependencies (CD)**
  Complex task which have inter-schedule dependencies.

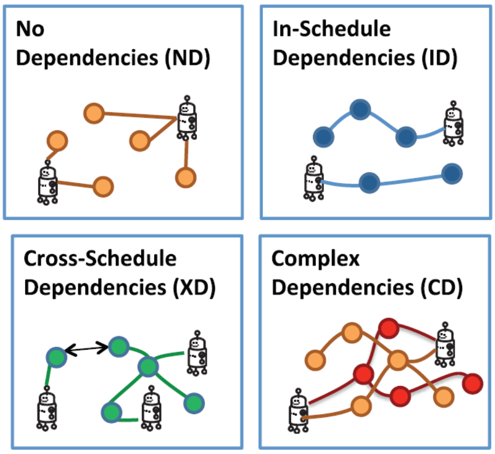


Figure 33: Examples illustrating the four high-level categories of the new taxonomy. Circles represent tasks, lines agent routes and arrows constraints. Source: [23]

These four high level classes are further broken down to a second level, which is based on the Gerkey and Mataric's taxonomy but is not relevant in this context. The paper then proposes one or more solutions to the different variants of MRTA problems.

The paper gives a good overview of the MRTA problems and introduces some useful taxonomies, some concepts of which can be found again in my work. For example, I also have a distinction between elemental tasks and complex tasks. It also gives a good insight into the problems and possible solutions that can occur with MRTA.

2.3.5 CHIMP - Online Task Merging with a Hierarchical Hybrid Task Planner for Mobile Service Robots

Last but not least I look at a concrete implementation of a task planner. In the paper of Sebastian Stock et al. the planner CHIMP [25] is introduced, which is a hybrid task planner with online merging capabilities. Hybrid planning is one strategy to treat multiple aspects of plan-based robot control like task dependency, time, space and so on. The problem is that this results in a gigantic search space. Therefore, hierarchical planning is used as a strategy to efficiently cut through this search space.

CHIMP is based on a meta-CSP planning. A backtracking search is done in which the HTN task is decomposed into a (meta-)constraint in order to ensure causal feasibility. The special feature that sets Chimp apart from other planners is that plans can be merged online. In the scenario shown in Figure 34, the robot initially only gets the task of moving *coffee1* to *table2*. After the task has been planned and the execution has started, a second task arrives. The *milk1* is now also to be brought to the *table2*. This unplanned task is now unified online with the already planned task.

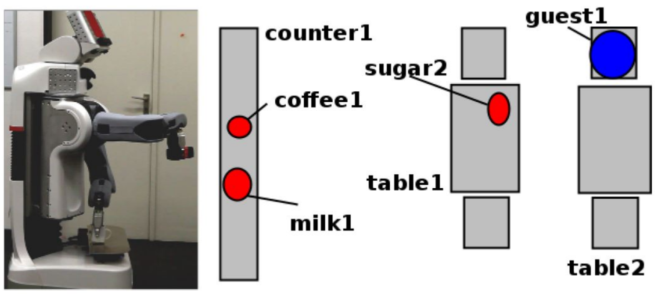


Figure 34: Illustration of the demo scenario. Source: [25]

The runtime of CHIMP increases exponentially with the number of operations, as you can see in Figure 35. This is, however, to be expected. The disadvantage is that CHIMP only deals with actuator planning and does not consider any human-machine interaction.

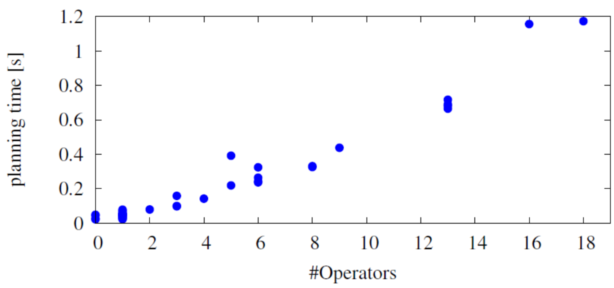


Figure 35: Relation between plan length and generation time. Source: [25]

2.3.6 Process-oriented Task Assignment for Assembly Processes with Human-Robot Interaction

Müller et al. published the paper *Process-oriented Task Assignment for Assembly Processes with Human-Robot Interaction* [26] in which they noticed that in factories tasks are often performed alternately by humans or machines, depending on whether, for example, appropriately qualified personnel is available. Also, some tasks can only be carried out by machines, one reason being the precision required. However, some large components are still manufactured entirely by humans. Moreover, it would often be necessary for humans and machines to work together cooperatively and made comparisons between them.

For this reason, the authors first took a closer look at the skills of humans and machines, compared them with each other. They have assigned a specific task to each skill, which represents the smallest unit in their system. These particular tasks were then combined into basic tasks, as can be seen in Figure 36. Finally, they have defined tasks and parameterized them accordingly so that they can be divided into corresponding specific tasks.

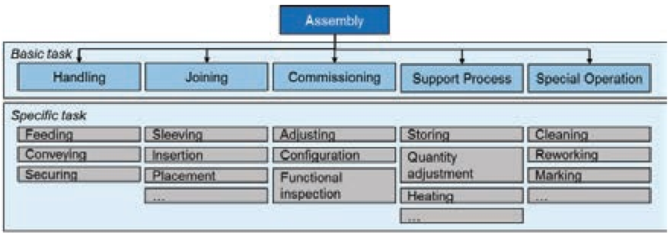


Figure 36: Model for basic (top) and specific (bottom) tasks. Source: [26]

If a task has to be performed, the abilities of the robots are compared step by step to those of a human being. Each time an assignment needs to be made, the system decides whether it is better to have the task performed by a robot or a human. Because it is difficult to express and compare a human with a robot being by using alphanumeric values, it is more like an estimation. Many factors can be taken into account in the assignment; for example, the focus could be not only on efficient production but also on the most cost-effective one.

Finally, they evaluated their results at the ZeMa, with the setup shown in Figure 37. They were able to distribute the tasks between human and machine successfully. However, it was also essential to consider things like ergonomics and the layout of the workplace. In my approach, I will also consider the layout, precisely the place of execution, when assigning the tasks.

Figure 37: Model for basic (top) and specific (bottom) tasks. Source: [26]

2.3.7 A Skill-based Robot Co-worker for Industrial Maintenance Tasks

In the paper *A Skill-based Robot Co-worker for Industrial Maintenance Tasks* [27] by Koch et al. the authors examine the concept of a robotic co-worker in an industrial environment. For this purpose, they developed the robot shown in Figure 38, which is intended to support the user in screwing tasks. This robot is controlled by a software called Skill Based System (SBS), which controls the assignment of tasks using skills.

Figure 38: KUKA LWR IV+

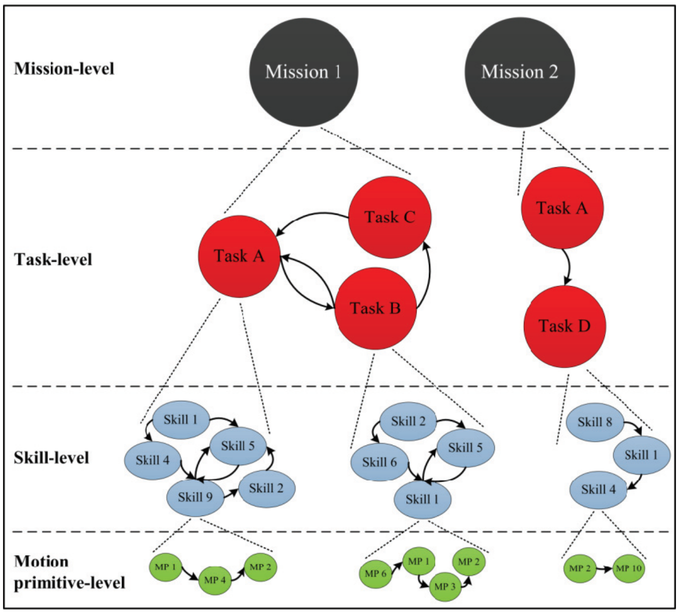


Figure 39: Structure of the skill model. Source: [27]

They defined so-called missions, which consist of individual tasks. These tasks map to skills, which again map to robot movements. This hierarchy is also shown in Figure 39.

In order to start a mission, a graphical user interface is available. When a mission is started, the preconditions and parameters are first checked to ensure that the mission can be executed. If this check is successful, the mission is executed. In addition, the final state after execution is predicted, so that it can be compared with the real state at the end to determine if the execution was correct. The process is shown in Figure 40.

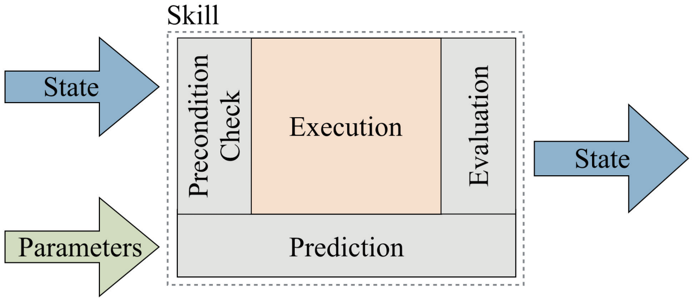


Figure 40: Execution of the skill model. Source: [27]

They then carried out a small study and found that the system could be used intuitively even by people from outside the field. My approach will also include a preliminary check of the parameters. However, I will not only predict the parameters but simulate a task before execution to make sure that all individual actions are possible.

2.3.8 Conclusion & Comparison

The research field of task generation, planning and assignment is enormous. Thus, the presentation was limited to a selection of relevant papers that cover interesting aspects of. task processing. There are many other approaches, e.g. that of Vaquero et al. in the paper *The Implementation of a Planning and Scheduling Architecture for Multiple Robots Assisting Multiple Users in a Retirement Home Setting* [28]. There are also many other tools and frameworks, such as the Planning Domain Definition Language (PDDL) [29] and its extension for multiple robots called MA-PDDL [30].

To conclude, I have summarized the papers explained previously in detail in a table and compared them with each other in order to draw comparisons among them and my approach. This gave me a good impression of which approaches have already been successfully implemented and what a system should look like that can meet my requirements.

Paper	General			Task Features						Task Execution					
	Goal	Field	Mutli Robot	Task Hierarchy	Parametrizable	Parallel Execution	Error Handling	Exec. Alternative	Situation depended Exec.	Preconditions Checked	Task Simulated	Skill Matching	Weighted Options	Task Assignment	Flex. Robot Iface.
Assembly Plans [16]	Following Assembly Plans	Industry		✓	✓			✓							
Inter. Autonomy [18]	Fulfilling simple Transport Tasks	Assistance Robots			✓		✓		✓				✓	✓	
SiAM-dp [20]	Dialogical Repr. of Tasks	Dialogue		✓	✓	✓	✓								
iTax [23]	Classifying diff. Task Types	Taxonomy	✓	✓		✓									
CHIMP [25]	Planning Tasks with mult. Robots	Planner	✓		(✓)	(✓)			✓	✓			(✓)	✓	
Task Assign. [26]	Task distribution btw. Hum. & Rbt.	Industry			✓				✓	✓		✓	✓	✓	
Skill-based [27]	Human-Machine Collaboration	Industry		✓	✓		(✓)			✓	(✓)	✓	✓	✓	
My Approach	Operate Robots by Voice	Industry	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3: Comparison of Related Work regarding Task Representation, Generation and Assignment

Chapter 3
Architecture

After I have presented the used related works, I will now give my approach in a broad overview. First of all, I will go into more detail about the architecture of the system to get an overview of the interrelationships and requirements of the different components. I will then present a theoretical background on the skill matching used, which had a decisive influence on all components. In the following chapters, I will go into detail about the individual components in the architecture.

3.1 Implementation Goals

As mentioned at the beginning, I want to construct a system which allows transforming commands given by voice into robot commands to meet the request of the user. For language processing, a natural language understanding system should be used to avoid rigid grammars. In my work, I will use an already existing NLU system called Cerence Mix, which already has a built-in speech recognition engine.

To transform the gained user intention into meaningful robot commands, some steps are necessary which are represented in individual components. All these are covered in my work, from the knowledge base, the modelling of tasks and the actual assignment of tasks to robots. These tasks a robot receives from my system are high-level abstractions. For example, a final command that is sent to the robot could be "grab the object". The conversion of these high-level commands into actual actuator movements must be handled by an external platform that knows the specifics of the different robots. This way, my system can be kept flexible and can be used with many different robots, because an external, specialised component will handle the low-level robot conversation. Besides, my work presents a task assignment rather than complex task planning. If complex planning is necessary, it could be taken over by an external planner in the future.

The system should later be able to use different robots with different capabilities to solve the same problem. For example, a single robot can solve the problem or several cooperative together. Therefore there can be several completely different solutions for a challenge. In the end, the system decides which solution is the best in the current situation.

3.2 Architecture Overview

For an initial introduction to the architecture, I have created a summarized overview in Figure 41. It consists of 4 large segments: the knowledge base, dialogue management, the task generation and assignment and the task execution. All of them interact and depend on each other. Therefore I will give a short introduction here so that one does not lose sight of the bigger picture in the later chapters.

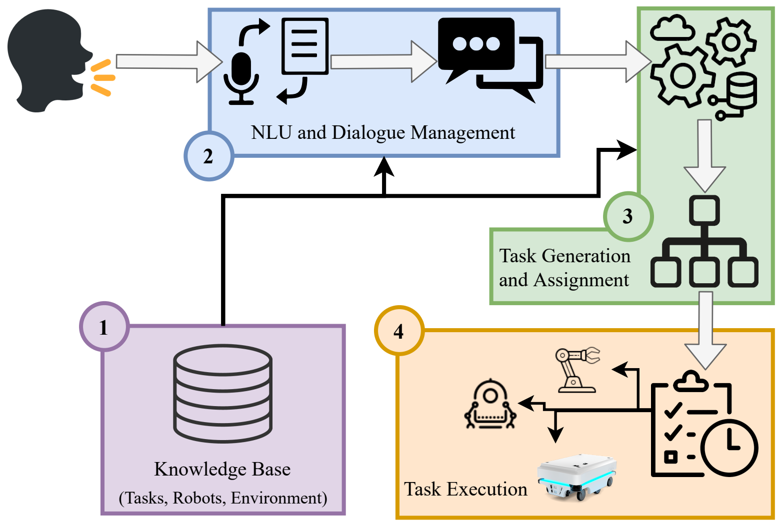


Figure 41: Graphical representation of the overall Architecture

The first principal component is the knowledge base, shown in purple in the Figure. In the beginning, the system needs the knowledge to perform the dialogue with the user. First, the NLU system needs to be trained. Besides, the dialogue needs to know more about the tasks to be able to ask meaningful follow-up questions. The task generation component is utterly dependent on the knowledge base since it operates entirely on it. The subsequent assignment of tasks to the robots also depends primarily on this. Therefore the knowledge base is a central component, which I will discuss in more detail in the following chapter 4.

The next important component is the NLU system and dialogue management, shown in blue in the Figure. It handles the complete communication with the user and is, therefore, the central interface between human and machine. Their task is to detect and validate the intention of the user. Should there be any uncertainties or missing information, it will solve them through quick back questions. In addition, the components can be used to provide feedback to the user during the execution of the tasks, e.g. on the current status. More details can be found in the chapter 5.

The user's request is then processed in the task generation and assignment component, shown in green in the Figure. This part is a significant part of my work. First, the component maps the user's request to a corresponding task. Then this task is broken down into its specific subtask; often, there are several possibilities of decomposition. Then the component determines which options are currently feasible, among other things through the presently available robots. After the algorithm has decided which solution is the best, the components assign the individual tasks to the robots. In chapter 6, this part of my work is described in detail.

Finally, the tasks have to be executed, which is what the task execution component is for, shown here in orange. The previous part passes a complete task plan with assigned robots. This component is now responsible for ensuring that the commands are executed correctly, taking dependencies into account. For this purpose, the component communicates with external software that translates the high-level commands into low-level robot movements. If problems occur, these are reported to the task planning component. I will go into the details in chapter 6.

The appendix contains a selection of screenshots of the graphical interface of the system.

Chapter 4
Representation of Knowledge and Skills

In this chapter, I will explain the representation of all knowledge of my system. First, I will explain in general how my knowledge base stores data. Afterwards, a theoretical section follows, in which the connection between robot skills and tasks is defined precisely. These definitions are fundamental because all other components are based on them. The theoretical part is followed by the practical part, in which the modelling of robots, the environment and finally, the tasks are explained.

4.1 Structure of the Knowledge Base

The presence of a well-structured knowledge base is fundamental to my work, as all decisions are derived from it. For this reason, I will rely on approaches that have already been successfully tested. I have decided to use the TBox and ABox approach [31]. General concepts, so-called TBox, and concrete instances, called ABox, are strictly separated. Thus the knowledge about concepts of the domain, the terminological knowledge, is exclusively stored in the TBox. This includes, among other things, the knowledge about the relationship between different concepts and their properties. This knowledge rarely changes. In contrast to this is the ABox, which reflects the current state of the world through its concrete instances and is therefore subject to frequent changes.

Therefore I use a well-structured object hierarchy for as my TBox model. Since many things are very similar, powerful inheritance in the knowledge base is an advantage. Hence, the characteristics of the parents' concepts are inherited by their children. It is also possible that a concept inherits from several concepts, and thus has several parent concepts. The inherited properties can already have a value assigned or be unassigned. If properties of different concepts have identical names, they are not overwritten, but kept separately according to inheritance structure, so no information is lost. But it is possible to overwrite values that have already been assigned. Thus, the knowledge base offers the highest possible flexibility in order to map all possible entities efficiently.

An elementary property for this work is the location. This is very important later on in order to plan the task correctly. Therefore, there is the basic concept of LocationEntity, from which virtually all other concepts inherit. This LocationEntity contains only a placeholder property for the location. This ensures that later all instances in the knowledge base have information about the location. There are also other fixed predefined concepts that are later used for planning and indicate whether, for example, an object can be gripped. More on this later.

Concrete instances (ABox) are represented by objects. These objects inherit all the properties that their concepts give them. Properties that have not yet been assigned a value now have a concrete assignment. Again, it is possible to overwrite properties already assigned. In addition, further properties can be added as desired, which were not previously noted in any concept. This allows a simple modelling of the environment in the further course of the work, which consists of many objects, which however have much in common.

A small example of a knowledge base is given in Figure 42. In this example concepts for screws are modelled and finally two concrete instances are shown. First of all there is the predefined LocationEntity. From this the general concept of a screw is created, whereby this concept is equipped with additional placeholder properties. Then the concept of the wood and metal screw is created. These two concepts inherit from the general concept of the screw, but one of them also from a new concept called ScrewableEntity. This concept of the ScrewableEntity is important later so that the system knows which operations are possible with which objects. In this specific case it would mean that only the metal screw can be screwed by a robot, but not the wood screw. Finally, two concrete instances are shown in which the still blank properties are assigned a value.

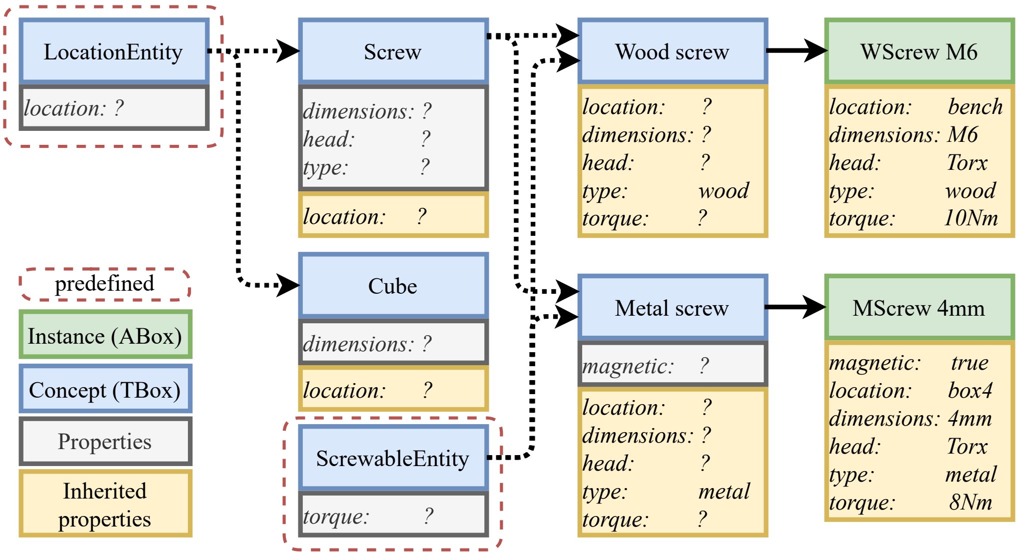


Figure 42: Example knowldge base with Concepts and concrete Instances for two screws

With the help of this ontology, all required knowledge can now be written down in a structured and efficient way. The ontology can change during runtime, e.g. if objects change their location because a robot moves them. Furthermore, the initial ontology must be loaded before starting. Therefore all persistent concepts and instances are stored in YAML format on disk. A distinction is made here between concepts and concrete instances. The reason for this is that the concepts are usually rather static and rarely change, whereas concrete instances, especially in the industrial environment, can change very quickly.

An example of such a file is given in Figure 43. On the left side you see the encoding of the concepts, on the right side the encoding of the instances. The great advantage of YAML, in contrast to XML for example, is that it is very easy to read and write, thanks to its simple yet powerful structure. Since the example is self-explanatory, and further details about the file specification are not very relevant for the work, I will skip further details here.

```yaml
# concepts.yaml
Screw:
  parents: [Object]
  properties:
   dimension: null
   head: null
   type: null

Cube:
  parents: [Object]
  properties:
   dimension: null

Wood_screw:
  parents: [Screw]
  override:
   type: wood

Metal_screw:
  parents: [Screw, ScrewableEntity]
  properties:
   magnetic: null
  override:
   type: metal
```

```yaml
# instances.yaml
WScrew_M6:
  concepts: [Wood_screw]
  properties:
   location: bench
   dimension: M6
   head: Torx
   torque: 10Nm

MScrew_4mm:
  concepts: [Metal_screw]
  properties:
   magnetic: true
   location: box4
   dimension: 4mm
   head: Torx
   torque: 8Nm
```

Figure 43: YAML representation of the KB shown in Figure 42

Since there are already predefined concepts in the knowledge base, not all concepts need to be stored in the YAML file. In the example given here, the concepts *Object* and

ScrewableEntity are not included in the file because they are integral parts of the system. How the knowledge base works in detail and how it is structured, I will explain in the following chapters.

4.2 Theory of Skills and Requirements

Before we can work with skills, we have to define them. What is a skill? Which skill needs a task? What happens if several robots are involved? These questions and more will be answered in the following chapter.

4.2.1 Robot Skills

As a novel approach, I define which robots can perform which tasks using skills. A skill is a two-dimensional parameter, where the first parameter is an abstract skill, and the second parameter gives the locations where the skill can be used to carry out a action. The possible skills are pre-defined by the system. So each skill represents one capability of robots, but it is possible that the capability is applicable in multiple locations. The location is elementary for the assignment of tasks. While stationary robots can usually only use their capabilities in one location, mobile robots can do so in many or even all locations.

Having this, a robot can be defined by a so-called skillset, which is a list of all skills it possesses. It is important to note that in a skillset, each skill may occur at most once, but since the locations are specified via lists, this is no problem. The formal definition of skill and skillset is given in the following.

Definition 1: Skill

$$\begin{aligned} skill &= (c, l) \\ c &\in \{capability_1, capability_2, capability_3, ...\} \\ l &\subseteq \{location_1, location_2, location_3\, ...\} \end{aligned}$$

Definition 2: Skillset

$$\begin{aligned} skillset &= \{skill_1, skill_2, skill_3, ...\} \\ \forall x, y \in skillset &: x \neq y \end{aligned}$$

An illustrative workshop could look like the one shown in Figure 44: The workshop has a total of three workplaces at which objects can be processed. There are three robots, two of which are stationary and the other is is freely movable. First of all, there is PaintRobo which can paint and grab objects. Next to it is MobiPick which can grab objects, store them and move freely on the floor. Last but not least, there is Baxter which can screw

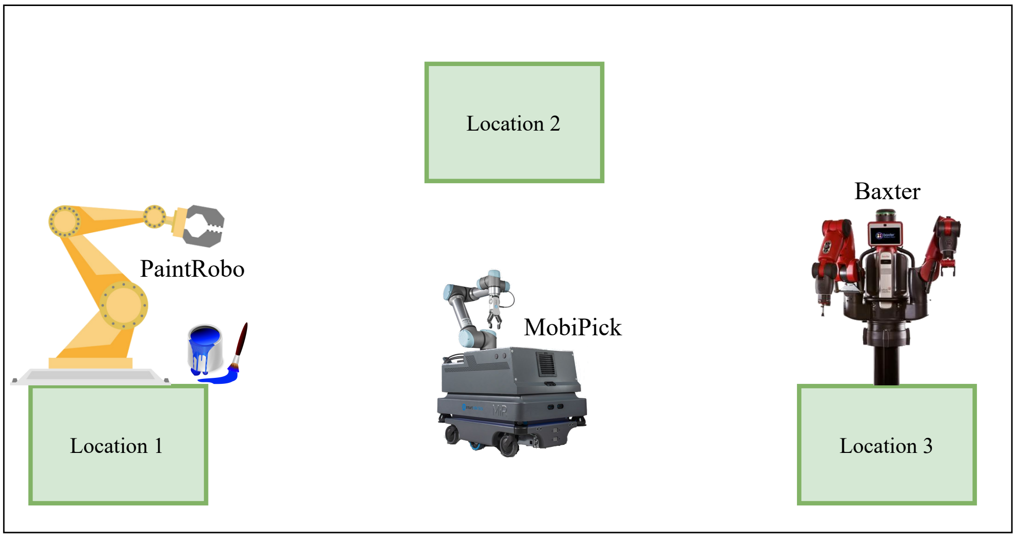


Figure 44: Workshop example with 3 locations and 3 robots

and inspect objects. All these different robots with their different capabilities can be represented by the following skillset.

$$\begin{aligned} skillset_{PaintRobo} = \{&(paint, \{location_1\}), \\ &(grab, \{location_1\})\} \end{aligned}$$

$$\begin{aligned} skillset_{MobiPick} = \{&(grab, \{location_1, location_2, location_3\}), \\ &(store, \{location_1, location_2, location_3\}), \\ &(move, \{location_1, location_2, location_3\})\} \end{aligned}$$

$$\begin{aligned} skillset_{Baxter} = \{&(inspect, \{location_3\}), \\ &(screw, \{location_3\})\} \end{aligned}$$

Since MobiPick can move freely, and there are no obstacles in the workshop (such as fences or barriers), this robot can reach all locations and use its skills everywhere. However, it should be noted here that a skillset shows all skills and locations that are in principle possible, but does not reflect the current situation. Logically, the MobiPick cannot offer its services at all locations at the same time, but must first be navigated to the appropriate location using corresponding commands. Therefore, the current position of the robot has to be considered as soon as there is more than one location. The skillset only specifies all possible operations but does not reflect the actual state of the system. Therefore, the current position of the robot must be taken into account. There will be more details on this later in the subsection 4.2.3 (Skill Matching).

4.2.2 Skill Requirements for Tasks

Now that we can compute the appropriate skillsets for a robot environment, we need the proper counterpart. For each task, specific skill requirements can be identified. These should later make it possible to find suitable robots or combinations of robots that can solve the task. As several different solutions are usually reasonable, the representation introduced must also be able to reflect this. Since the skillsets are rather general and list all possibilities, but the actual execution depends on the current location of the robot, in this part, special attention is paid to the location. Besides, it must be possible to chain different requirements so that successive tasks can be combined into one single requirement.

First of all, we have to define a simple requirement, which forms the basis for more complex requirements. A simple requirement for a task that needs the *capability c* at *locations l* is defined as:

Definition 3: Simple Requirement

$$\begin{aligned} simple\_requirement = (c, l) \\ c \in \{capability_1, capability_2, capability_3, ...\} \\ l \subseteq \{location_1, location_2, location_3\ ...\} \end{aligned}$$

As you can see, this definition is practically identical to the definition of a skill. The next important step is to define when a robot's skill satisfies a simple requirement, meaning that the robot has the ability to fulfil the requirement. The following definition is used for this purpose:

Definition 4: Skill satisfies Simple Requirement

A simple requierment $r = (c, l_r)$ is satisfied by a skill $s = (s, l_s)$ iff:

$$c = s \wedge l_r \subseteq l_s$$

Example: Given the following simple requirements and skills

$r_1 = (grab, \{location_1\})$
$r_2 = (grab, \{location_2\})$
$r_3 = (move, \{location_1, location_2\})$

$s_1 = (grab, \{location_1\})$
$s_2 = (grab, \{location_1, location_2\})$
$s_3 = (grab, \{location_3\})$
$s_4 = (move, \{location_1, location_2, location_3\})$
$s_5 = (move, \{location_1, location_3\})$

The table indicates whether a simple requirement is satisfied by a skill:

	s_1	s_2	s_3	s_4	s_5
r_1	✓	✓			
r_2		✓			
r_3				✓	

At this point we can check simple requirements of single atomic tasks. The next step is to formulate requirements for simple tasks. A simple task is a chronological sequence of atomic tasks that should be processed by only one robot. The robot therefore needs all capabilities that the requirements ask for. However, our current definition is not broad enough to reflect this. For this reason, I define the robot requirements in the following. These are a list of the individual simple requirements. However, each ability may only occur once, so that capabilities that are used several times, e.g. at different locations, must be combined. Often the locations are not concrete locations but variables that are later replaced when the task is generated.

Definition 5: Robot Requirement

$$robot\_requirement = \{r_1, r_2, r_3, ..., r_n\}$$
$$\forall x \in [1..n] : r_x \in \textit{simple\_requirement}$$
$$\forall r_1, r_2 \in robot\_requirement : r_1 = (c_1, l_1), r_2 = (c_2, l_2) : c_1 \neq c_2$$

An easy way to calculate the robot requirement resulting from a list of simple requirements is given with the following algorithm:

Algorithm 1: Aggregate Simple Requirements to Robot Requirements

Data: sr[0..n] with $\forall x \in \{0..n\} : sr[x] \in simple\_requirements$
Result: rr $\in robot\_requirement$

```
result ← {}
for x = 0 to n do
    (c, l) ← sr[x]
    foreach m ∈ result do
        (c_tmp, l_tmp) ← m
        if c_tmp == c then
            l_tmp.AddIfNotExsists(l)
            continue
        end
    end
    result.Add((c, l))
end
return result
```

Definition 6: Skillset satisfies Robot Requirement

A robot requierment $rr = (sr_1, sr_2, ..., sr_n)$ is satisfied by a skillset $sset$ if:

$\forall sr \in rr, \exists s \in sset$: *s satisfies sr* (by Definition 4)

To illustrate the above, let me take the task "Move an object from A to B" as an example. At first, I limit myself to a solution that involves only one robot that carries out all necessary steps. For more complex solutions we need further definitions, which I will introduce later. The task is divided into four atomic tasks which require different skills at different locations. A graphical representation of this decomposition is given in Figure 45.

The simple requirements of the for atomic task can be stated as follows:

$$r_1 = (grab, \{location_{green}\}) \qquad r_3 = (move, \{location_{purple}\})$$
$$r_2 = (store, \{location_{green}\}) \qquad r_4 = (grab, \{location_{purple}\})$$

If we now apply the the algorithm outlined above, we get the following robot requirement for this example:

$$\begin{aligned} robot\_requirement = \{&(move, \{location_{purple}\}),\\ &(grab, \{location_{green}, location_{purple}\}),\\ &(store, \{location_{green}\})\} \end{aligned}$$

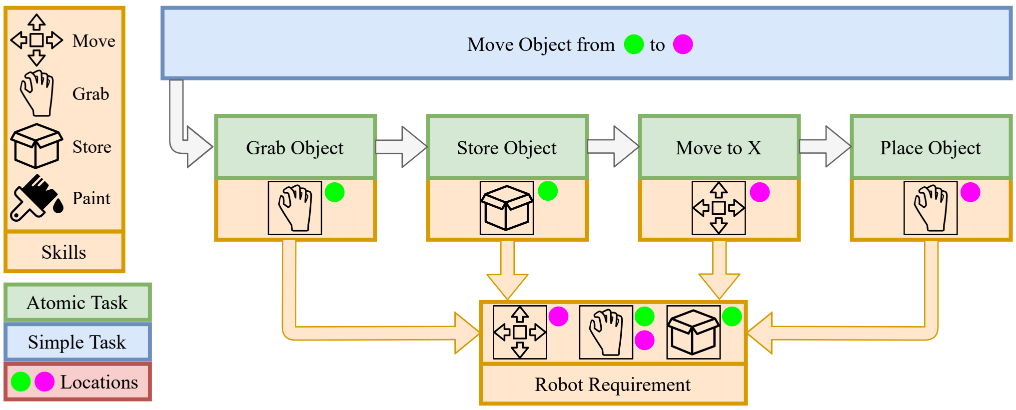


Figure 45: Basic example for calculating required skills for a given task. Green and purple are variables for two different locations.

Now that I have proposed a way to define requirements for tasks executed by only one robot, the last question is how to define requirements for tasks that involve multiple robots. These so-called task requirements can be given as lists of robot requirements, where each element of this list represents a robot. Each entry in the list corresponds to an independent robot which must later fulfil the requirements. The available robots may be used several times to meet the requirements. The formal definition is as follows:

Definition 7: Task Requirement

$$task\_requirement = \{r_1, r_2, r_3, ..., r_n\}$$
$$\forall x \in [1..n] : r_x \in \textit{robot requirement}$$

To clarify the definition, consider the following example: A task that consists in delivering a blue object to the worker's workbench. Here, we consider the possible situation that there is no longer a blue object in the warehouse and therefore the object must be coloured beforehand. Therefore, the object has to be first taken out of the warehouse, brought to a painting station and then handed to the worker. These three steps are already implemented as tasks in the system, along with the associated robot requirements. A graphical representation of the task is given in Figure 46. Note the three different colours representing the three locations: green for the warehouse, purple for the painting station and blue for the position of the worker. To determine the task requirements, all individual robot requirements are simply combined into a list respecting the definition. In this case, the results are as follows:

$$\begin{aligned}
robot_requirement_{take_object} &= \{(move, \{location_{purple}, location_{green}\}),\\
&\quad (grab, \{location_{purple}, location_{green}\}),\\
&\quad (store, \{location_{green}\})\}\\
robot_requirement_{paint_object} &= \{(paint, \{location_{purple}\}),\\
robot_requirement_{bring_object} &= \{(move, \{location_{purple}, location_{blue}\}),\\
&\quad (grab, \{location_{purple}, location_{blue}\}),\\
&\quad (store, \{location_{purple}\})\}
\end{aligned}$$

$$\begin{aligned}
task_requirement_{bring_blue_object} &= \{robot_requirement_{take_object},\\
&\quad robot_requirement_{paint_object},\\
&\quad robot_requirement_{bring_object}\}
\end{aligned}$$

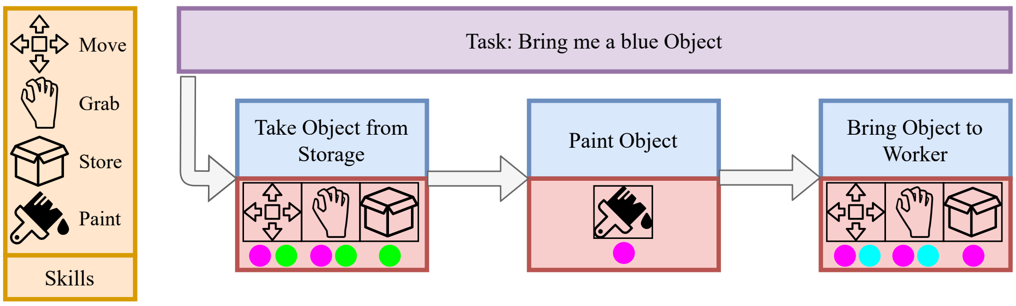


Figure 46: Task involving up to three robots: Bringing an object from the storage (green) to a painting station (purple) and then to the worker (blue).

For a given task requirement, it could be that subtasks have to be distributed among several robots, respecting their abilities. To represent the skills of a group of robots, I introduce the definition of a robot collection. This is a list of skillsets which represent the different robots. This collection will be used later to describe solutions for the task requirements. The definition is given below.

Definition 8: Robot Collection

$$robot\_collection = \{sset_1, sset_2, sset_3, ..., sset_n\}$$
$$\forall x \in [1..n] : sset_x \in \mathit{skillset}$$

Now that the foundations have been laid to describe a collection of robots in terms of their skills, the only thing missing piece is the definition of when such a group fulfils a task requirement. This is exactly the case if each robot requirement within the task requirement can be fulfilled by at least one robot of the collection, as defined below.

Definition 9: Robot Collection satisfies Task Requirement

A task requirement $tr = (rr_1, rr_2, ..., rr_n)$ is satisfied by a robot collection $robots$ if:
$\forall rr \in tr, \exists sset \in robots$: *sset satisfies tr* (by Definition 6)

In this section, we have defined the basics for defining requirements at the various levels. This hierarchy is shown graphically as a summary in Figure 47. The different levels later play an important role in the assignment of robots to concrete tasks, because the assignment works along this hierarchy. First the robot requirements are satisfied step by step, until finally the complete task requirement is fulfilled and a valid assignment is obtained.

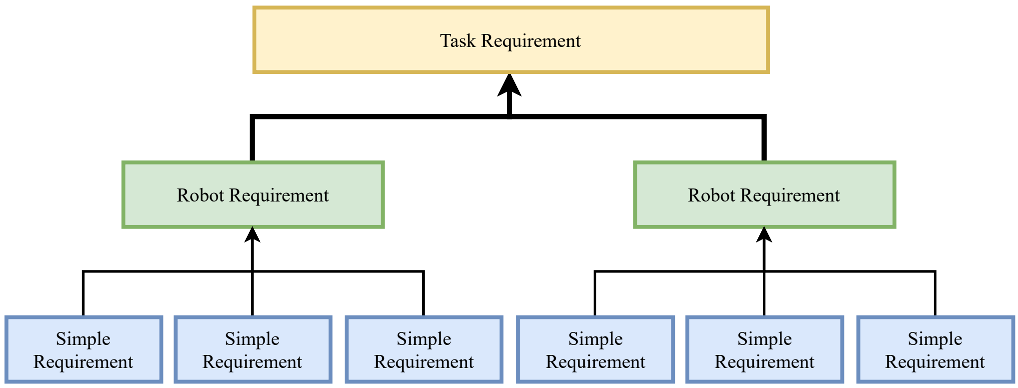


Figure 47: Hierarchy of the different types of requirements.

4.2.3 Skill Matching

An interesting piece of my work is how the robots get to their tasks. The idea is that each robot has a collection of skills that it provides as a service. A task can now have several possible solutions for which different skills are needed. It could also be that one solution requires only one robot, while another requires two robots working together. Because the tasks now have a requirement in the form of skills and the robots themself possess skills, an assignment of robots to tasks is possibly by matching the skills. If there are several possibilities or combinations to solve the task, a cost metric is used to decide which is the best solution. This metric is explained later in the subsection 6.3.2 (Assignment of Tasks to Robots).

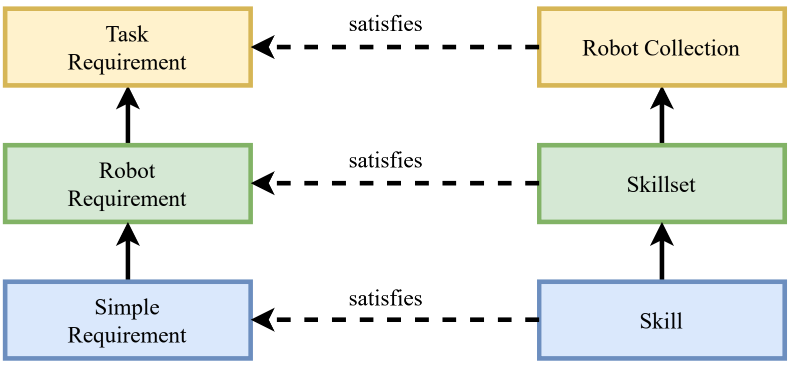


Figure 48: Different hierarchy levels in the requirements are fulfilled by equivalent levels in the skills.

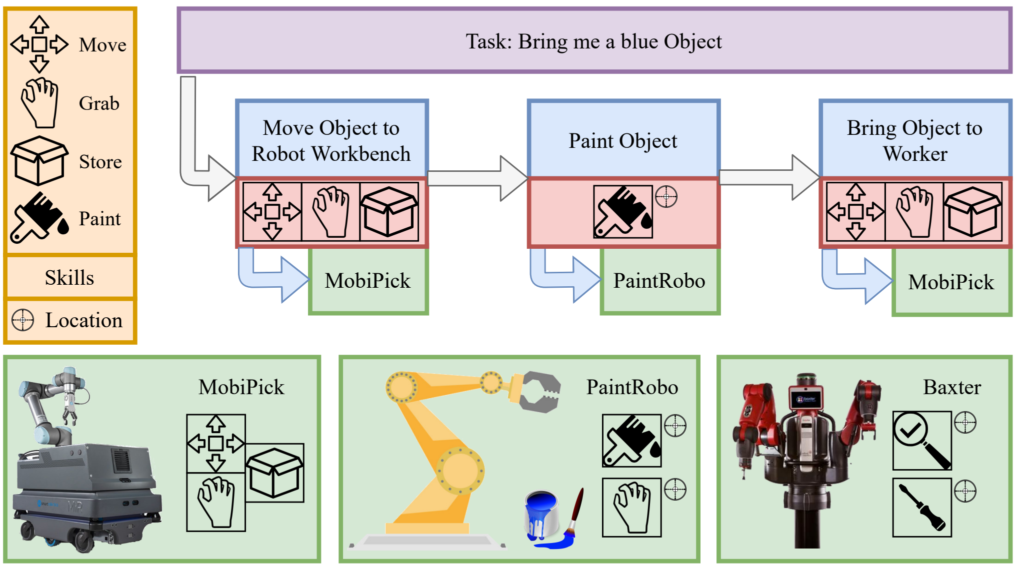


Figure 49: Multi robot skill matching example

4.3 Representing a Robot

After having introduced the basics of my knowledge base and the skill model, I can now start to model a robot. The concepts and inheritances available in the ontology are used for this purpose. First of all, we need to model the robot skills in a meaningful, way so that they can be easily mapped to the concrete robots later. For this purpose, I will make use of concepts in the ontology. Firstly, there is a general concept *RobotSkill* which stands for the different robot skills. A new concept is derived from this general concept for each possible robot skill. Besides, there is also a general concept for robots, which inherits from the *LocationEntity*. Yet another concept represents the type of robot. It inherits on the one hand from the general robot concept and on the other hand, from exactly those skills the robot can provide. Concrete robots in, e.g. an industrial hall are represented by specific instances of the corresponding concepts.

To model more complex robots, robots can of course inherit from each other. This makes sense especially if a robot unit consists of several separate robots. This representation allows a strict separation between concepts and instances. In addition, it is possible to model new robots with complex skills with very little effort.

An example of a robot model with concrete instances is given in Figure 50. Within the red dotted area there are already defined concepts. A total of three robots are defined here as a concept: a *UR Arm*, a *MiR 200* and a *MobiPick*, which is an assembly of the two other robots. The skills are simply assigned to the robot concepts by inheritance, thus the *MiR 200* receives the skill move and the *UR Arm* obtains the skills grab and screw. Since the *MobiPick* inherits from both, it ultimately gets all three skills. Last but not least, two concrete robots are created, a *MiR 200* and a *MobiPick* robot. With these concrete instances, tasks can be performed later. As mentioned in the beginning, this structure can be saved as a YAML file to be loaded at the beginning.

For my work, the following list of robot skills are implemented. This list can easily be extended later.

- move
- grab
- store
- place
- paint
- glue
- inspect
- magnetize
- press
- drill
- screw
- cut
- engrave

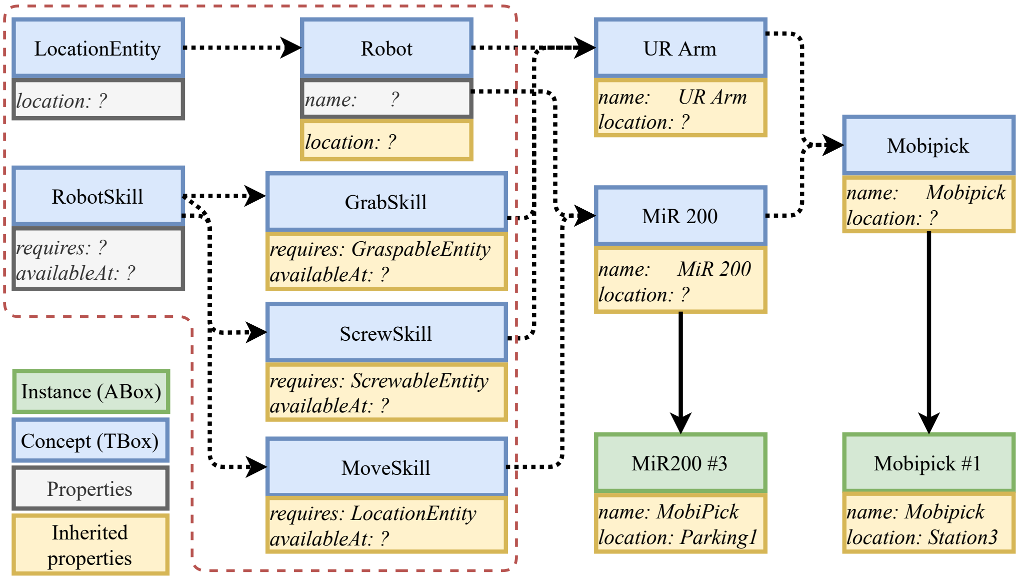


Figure 50: The model of three robots as concepts, and two concrete instances.

4.4 Representing the Environment

Now that we can model robots, the next step is to model the environment the robot interacts with. First of all, the user has the most freedom possible to define custom concepts and instances with arbitrary properties. This is also necessary because many industrial applications can be extraordinary and different. However, to enable the system to do meaningful things with self-defined concepts and instances, there are two approaches: First, predefined concepts can be inherited to indicate common properties. Second, properties can be actively queried in the tasks, but more about this later.

The most important predefined concept is the *LocationEntity*. It allows you to assign a location to all concepts and instances, even self-defined ones. Locations later play a central role in assigning a task to the robot. Of course, the location alone is not enough and the system must also know whether and how it can manipulate an object in a certain way. Therefore, there is a corresponding concept for every manipulation that can be performed with an object. Inheritance indicates that an object can be manipulated in this way.

A small example is given in the following Figure 51. There are two predefined concepts: *ScrewableEntity* and *MagneticEntity*. The first one indicates that the object can be used for screwing, the second one that an object can be raised via magnetism. In this model, both concepts of screws (wood and metal) can also be screwed as they inherit from *ScrewableEntity*. However, only the metal screw can be lifted by a magnet, as only this one inherits from *MagneticEntity*. In this example, the concrete instance *MScrew 4mm* has another special feature: It defines a property that is not derived from the concept itself. This property can be processed in a later task.

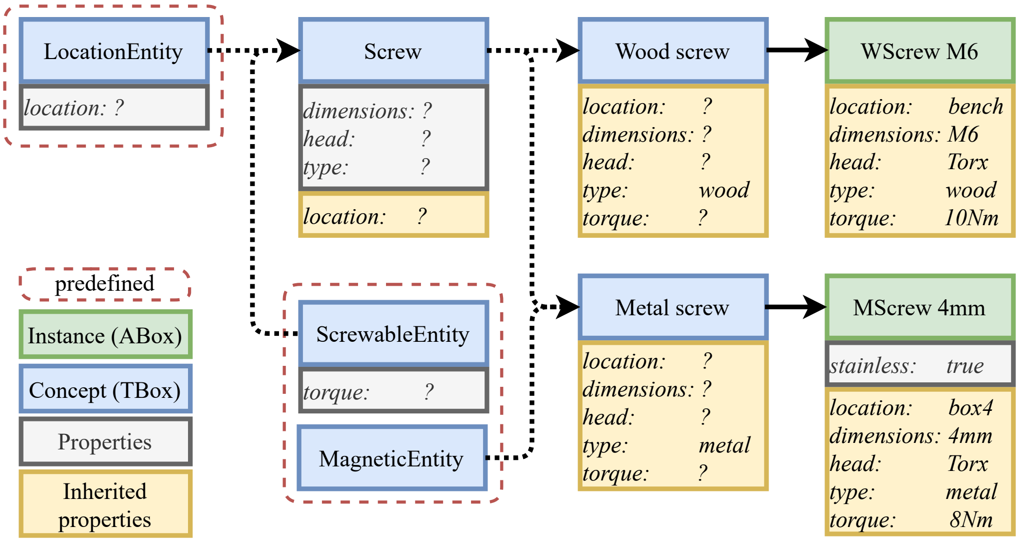


Figure 51: Representation of the environment using the example of two screws

4.5 Representing a Task

Having dealt with the modelling of the environment and robots, we now come to the heart-piece, the tasks.

4.5.1 Modelling a Task

The tasks are modelled outside the ontology because the scheme of concepts and instances is not suitable for representation. The tasks consist of a sequence of different steps, although these steps may vary depending on the environment. A reason for different executions can be e.g. the currently available robots or the current state of some objects. To respect such constraints, the tasks are stored as a directed graph.

Each graph has a fixed entry point and can have multiple endpoints. Any number of nodes may lie in between them. These nodes can have an in- and out-degree higher than one. Each node represents an action that starts when you enter the node and ends when you leave the node. If a node has several successors, the path taken depends on the node, for example, on whether an object has a particular property or not. An example of such a directed graph is given in Figure 52. Here you can see one entry point and two endpoints. In between, several nodes are representing different actions. There are also different paths possible to get to the ends.

The entry node defines the input parameters required by the task. For example, a task that paints an object would need as input parameters the object to be painted and the colour in which the object is to be painted. In addition, if the task should be triggered by the user, the data to train the NLU system is stored here; more on this in the next chapter. The end node only defines the end of the task and has no further function.

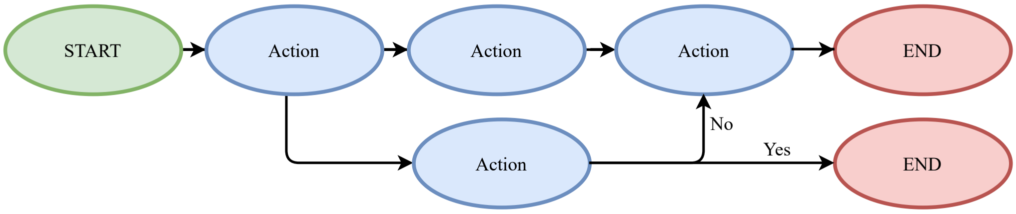


Figure 52: Using a directed graph to represent tasks

All nodes between start and end nodes define the actual task. These nodes can be divided into different classes. Depending on which nodes are used, the task has different overall complexities, based on the iTax taxonomy from the paper *A comprehensive taxonomy for multi-robot task allocation* [23]. In the following subchapters, I will explain the different node types and their relation to the taxonomy. I will also introduce a variable system to make the tasks as dynamic and flexible as possible so that they can be reused.

4.5.2 Robot Nodes

Robot nodes are the most basic type of nodes. For each robot node, there is a matching skill. They are used to execute the corresponding skill. Each robot node, therefore, represents a *simple requirement* and is later the basis for the skill matching. The individual robot nodes can have different variables as input and trigger specific transitions, depending on which skill is involved and the result of the execution by the robot. But all robot nodes have one thing in common: the location as an input parameter. This specification is essential for future skill matching. These nodes are later responsible for all robot movements. These nodes can also change the knowledge base, e.g. a correct execution of a paint job changes the colour of an object stored in the knowledge base, ensuring that the mapping in the knowledge base corresponds to the real world.

An example of two robot nodes is given in Figure 53. The two skills "painting" and "moving" are graphically represented. As you can see, they differ in terms of their input parameters and output transitions, except for the location which is important for matching with robots later.

A chaining of robot nodes is always executed by only one robot. Most times, several different operations have to be performed in sequence to achieve the goal of the task. Often it is also faster and more efficient if only one robot solves the task instead of several ones. Therefore, the system assumes in principle that all these tasks are performed by only a single robot. Of course, this is not always true, so there are ways to define the task in such a way that the system knows that the task can be processed by another robot from now on.

One option is to use a so-called robot turnover node. It signals that another robot can do the following part, but does not have to. If this node is included in the graph, the resulting task requirements change to represent the possible handover to another robot. An example of this is given in Figure 55.

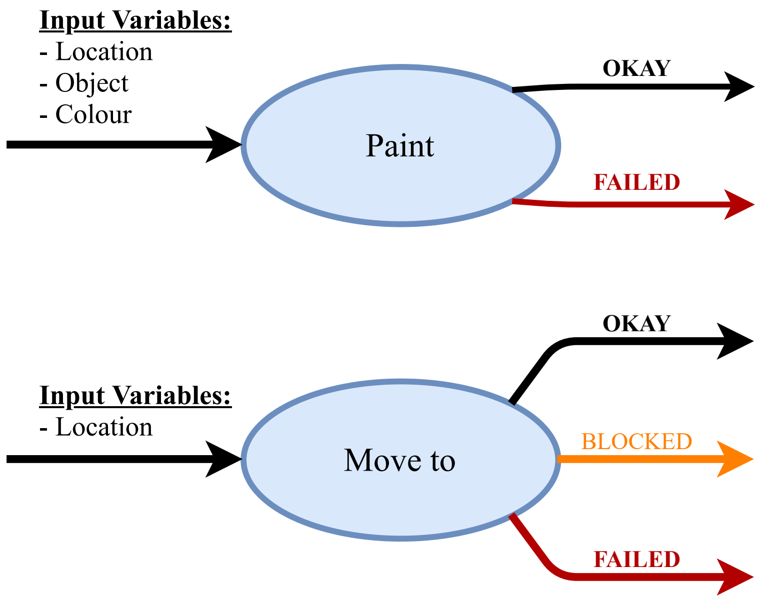


Figure 53: Robot nodes paint and move with corresponding input parameters and output transitions

Figure 54: Simple Task

4.5.3 User Interaction Nodes

Besides controlling the robots, it is also important to keep in touch with the user. Therefore, there is an interaction node. It can trigger a voice output. Speech input has not been implemented because it was not necessary for my application, but it is technically possible. The speech output allows the user to be informed about the current status of the task, for example, or to be made aware that a robot is about to leave.

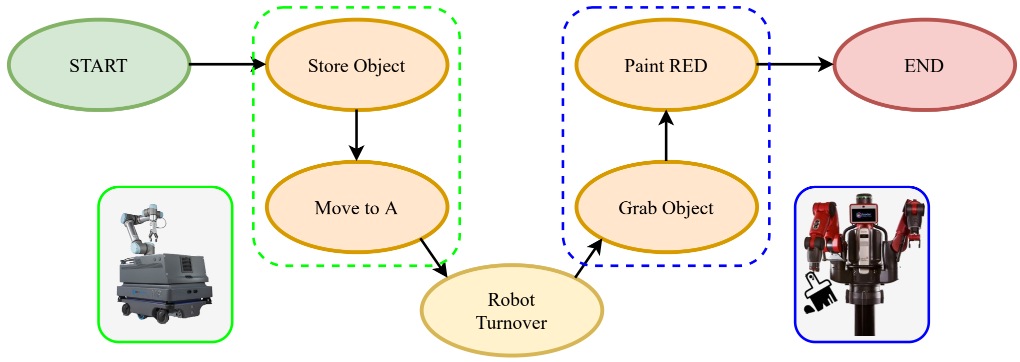


Figure 55: The turnover node allows the green actions to be performed by one robot and the blue actions by another.

4.5.4 Computational Nodes

So far, we can can give commands to the robot and interact with the user, but it is also important to be able to calculate certain things. For this reason, I introduced computational nodes. These allow us to query and modify knowledge from the database or local variables. For example, there is a search function that allows you to find the nearest round object with predefined properties in the knowledge base. The result is then stored in a local variable and can be used later. An application purpose for this are e.g. tasks of the type "bring me object x", in which the system must first locate the object in the environment using its knowledge base. Of course, simpler things are also possible, such as simply increasing a counter.

4.5.5 Control Flow Nodes

Since one series of actions rarely fits all cases, there are control flow nodes. These allow the sequence of the task to be adapted dynamically. In this way, tasks can be formulated in general terms and can later be used for a variety of applications. This also makes it possible to use different constellations of robots for the same task.

The simplest case is an if-statement. Using it, you can easily modify the flow of the task based on a local variable or the knowledge base. A simple example of this is shown in Figure 56. First, an if-statement is used to query whether an object can be painted or not. If not, a voice output is generated, and the task is terminated. But if so, a property of the object is checked that determines the colour of the painting and the painting action is triggered. This also allows a first simple error handling. Furthermore, loops can be constructed with the if-statement to model repeating processes without much effort.

But there are more powerful nodes to change the flow. A very important node is the "skill selector". It allows to model different solutions for the same (sub)task, which makes it possible to react flexibly. Thus, different possibilities of execution are encapsulated in such a skill selector. All available nodes can be used freely. The skill selector then decides which of the various executions is possible at all, and if there are several, the most optimal

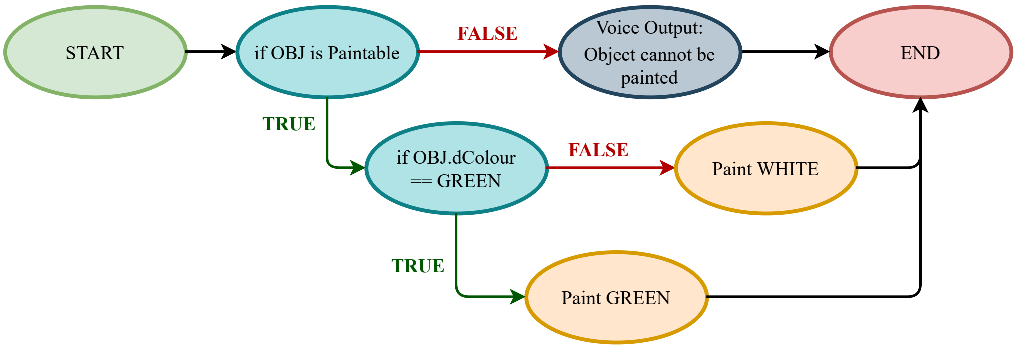


Figure 56: If Statement used for allowing different options to solve a problem

one is selected. What the most optimal one is, more about this later.

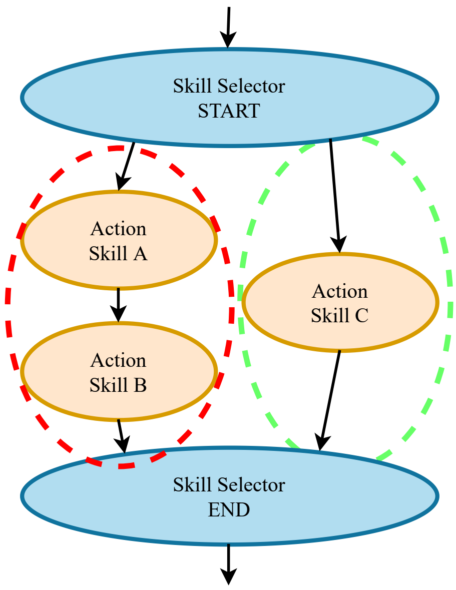


Figure 57: Skill selector node with two possible paths in red and green. Both ways lead to the same result, so only one needs to be executed.

To illustrate, an example is shown in Figure 57. There are two different ways to reach the same goal. One is the red path which requires a robot with the skills A and B, and the other green path which only requires a robot with the skill C. There must be no transitions between the red and green path. With the rules defined above, you get the following task requirements for the red and green path of the (sub)task (assuming that the location is always "bench"):

$$task\_requirement_{task\_red} = \{\{(SkillA, \{location_{bench}\}),$$
$$(SkillB, \{location_{bench}\})\}\}$$
$$task\_requirement_{task\_green} = \{\{(SkillC, \{location_{bench}\})\}\}$$

With the help of the calculated task requirements, a collection of robots can now be created that could be considered as solutions for the red or green path. If only one solution is left at the end, it will be taken, and if there are several ones, the best possible selection is made using a costs metric I will introduce later. If there is no solution, the current task is unfortunately not feasible.

Since you can define any number of alternatives of any complexity, this node allows the highest flexibility in the system. However, there is still a performance problem: All tasks are processed sequentially, and there are cases where it would be useful to have several robots working parallel. For this case there is the parallel execution node.

This node is very similar to the skill selector in terms of syntax. It also has any number of different paths that are encapsulated within the parallel execution node. Instead of only one of the paths being executed here, though, all paths are executed at the same time. A synchronization within the paths is not possible. The synchronization always occurs at the end of the parallel execution node. Furthermore, no transitions between paths are allowed. When calculating the task requirements, the possibility to process all paths in parallel is opened up. If there is no suitable solution, the parallel parts are still executed sequentially. This leads to the greatest possible compatibility, so that a solution is always found wherever possible. In addition, it allows to model tasks not only flexibly but also very efficiently.

An example of the application of these components is given in Figure 58. This is a simplified representation of the task "Mount the red ball". The system must first paint a ball red, deliver a screw and finally screw the ball to the workpiece. Here, the painting of the ball and the delivery of the screw can be done in parallel. After both subtasks are finished (painting and delivery), the ball can be screwed together.

4.5.6 (Sub)task Nodes

It often takes several small steps to complete a task. These small steps are recurring between the different tasks. Therefore, it makes sense to define subtasks that can be used over and over again. This saves you a lot of time during modelling. Furthermore, optimizations on these subtasks are directly available for many different tasks.

There is an extra node for this called subtask node. This can be used to call and execute any other tasks. The start parameters are passed as variables and then the task is executed, as simple as a function call. However, there is no return value, but instead values can be saved in the knowledge base that should be globally accessible.

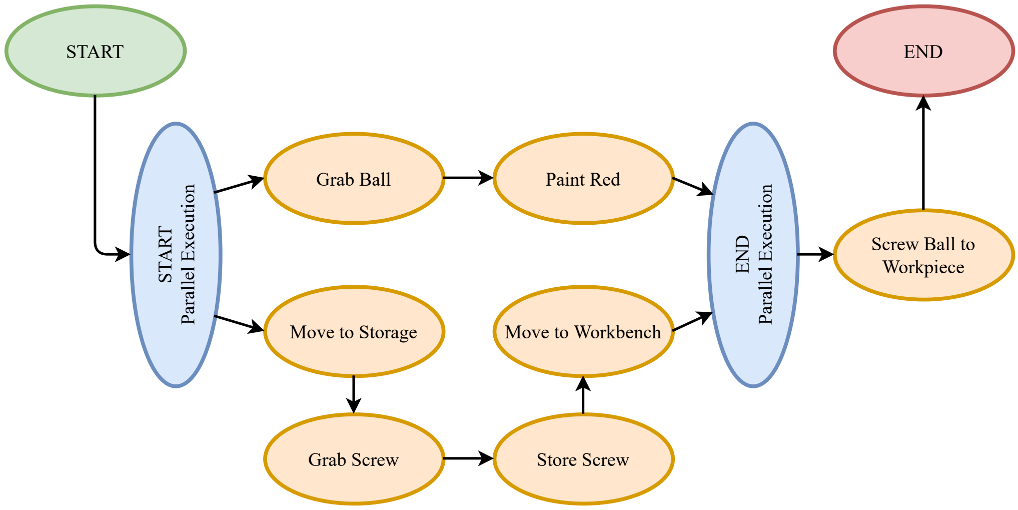


Figure 58: The parallel execution allows to improve the efficiency of the tasks.

A small example is shown in Figure 59. Here the task "bring object X to A" is modelled as a subtask. This allows the main task to use the subtask again and again by simply by calling it, which saves a lot of modelling work and makes it more clearly arranged.

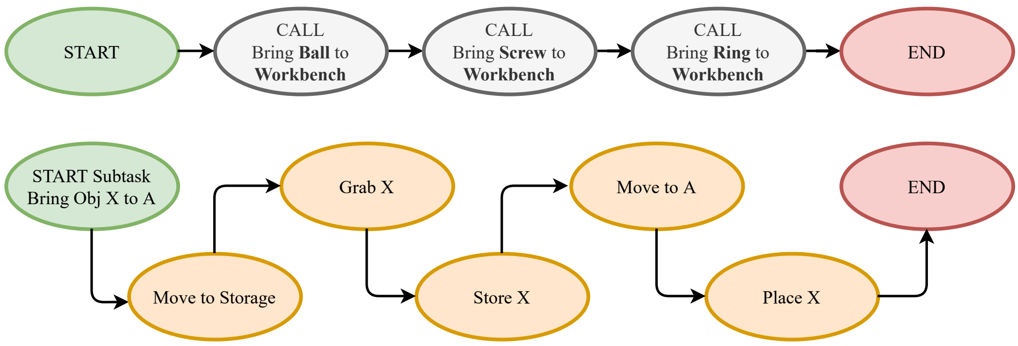


Figure 59: The calling of subtasks allows for efficient modelling.

4.5.7 Other Nodes

In the following, I will present a few more nodes that can be important for modelling and executing tasks, but do not fit into any of the other categories.

An important one is the exception node. It can be used if the execution of the task is, for whatever reason, no longer possible. The user then receives a warning message and the system goes into an error state. One reason for this can be, for example, that a robot suffers a defect during execution or the emergency stop switch was pressed. This exception node would also be a starting point for further work, by connecting an intelligent system, which tries to solve the problem or initiates a transfer of control to a human.

There is also a log node that creates an entry in the log file. This can come in handy for debugging purpose.

4.5.8 Summary of Task Node Types

To provide an overview of the many different types of nodes a task can consist of, I have summarized them all again in Figure 60.

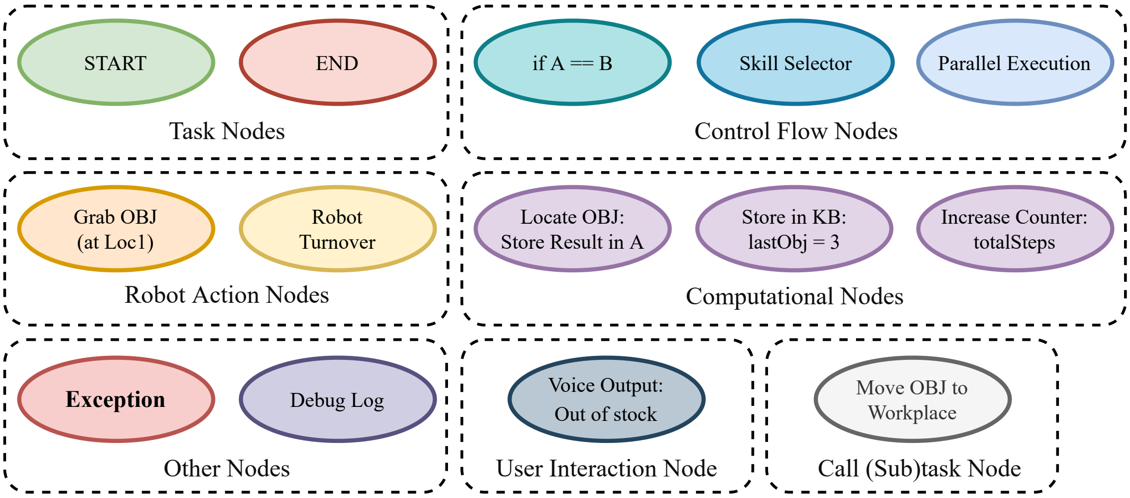


Figure 60: Summary of the different types of task nodes.

4.5.9 Generalisation using Variables

To keep the various tasks as general as possible, it is important to have a powerful variable system. In particular, the location can often only be specified as a variable as it depends, for example, on the storage location of the object or the location of another robot. Furthermore, in order to be able to use subtasks reasonably and flexibly, you should keep open variables in their specification.

As a consequence, each task has its own variable memory to which variables can be freely added or where they can be modified. As soon as a subtask is called, a new variable memory is created for it. Only the values transferred as parameters are stored in this variable memory. This means that each subtask added by a call runs in its own environment, which is not influenced by others. If variables are to be available globally, i.e. across subtasks, they should be stored in the knowledge base.

4.5.10 Different Task Types

With these modelling tools it is now possible to cover all use cases in an efficient way. Furthermore, it is possible to classify our resulting models using the iTax taxonomy [23]. This allows it to estimate in advance how complex the planning may become and what solutions to it there are. For the classification, only two nodes are relevant: the robot turnover node and the parallel execution node. Thereby it is important to also consider the subtasks that are called.

First of all, our system is not capable of representing a *no dependency* (ND). This is since the tasks are modelled as directed graphs and therefore, always have a given order. In industrial applications, however, the case without dependencies is of no further interest, since there is practically still a fixed order. So, if none of the three nodes is present, it is an *in-schedule dependency* (ID). Thus there are only dependencies within the task.

As soon as the robot takeover node or parallel execution node occur, though it is possible that several robots have to work together. There are dependencies regarding the synchronization of the task. However, there is only one possible solution. Therefore, these tasks are *cross-schedule dependencies* (XD). As soon as the skill selector node appears, it is a *complex dependency* (CD) because there are at least two different solutions. As a result, the skill selector node is the most expensive component for planning later.

Chapter 5
User Interaction

Another important aspect I want to discuss is user interaction. As the primary interface between human and machine, it is the part of the system that the user is confronted with. Therefore, a user-friendly and intuitive implementation is essential. To achieve this, I rely on a natural language understanding (short: NLU) system which I will explain here in more detail. I will also go into details of the dialogue model.

5.1 NLU

First, I will introduce the basics of an NLU system: what it can do and where its limits are, how it works and how I use it in my work. I have integrated two different NLU systems for my work.

5.1.1 Basics of NLU Systems

An NLU system tries to understand the intention of the user in freely formulated sentences. Understanding free formulations is one of the big advantages to grammar-based dialogue systems which can only understand predefined sentences. This aspect allows for the most intuitive use possible since the user does not have to learn fixed sentences in order to operate the system. He can just start talking and rely on the system to recognize his intentions.

The output of an NLU component which is further processed by the system differs from the output of a grammar-based dialogue. While a grammar can return exact information if the user says a predefined sentence (assuming the speech recognizer works correctly), an NLU system only returns information with certain probabilities. In addition, practically all NLU systems extract only two types of information from a sentence: the intent and the entities.

The intent corresponds to the will of the user, e.g. the fact that he wants the robot to bring an object or to screw something together. In addition to the intent, there are entities that help identify what exactly a user wants. These can be understood as properties of the intent. Most times, they specify what objects the user is referring to. In the example where the user needs an object, there is an entity for exactly this object. Given these pieces of information, intent and entities, a task can be initiated.

As a more detailed example, assume the user says the sentence "Please bring me a wrench", where the NLU system should recognize the "bring" intent. Additionally, the wrench should be recognized as an entity. Another sentence could be "Colour the cube red", here the intent "paint" should be recognized along with the two entities cube and red. The same result should be given upon the sentence "Can you please paint the cube red" or "The cube should be coloured red".

For the assignment of freely formulated sentences to intents to be successful, the intents must meet certain requirements. The most important one of these is that all intents are disjunctive. This means that intents are clearly different and do not logically overlap. An example of such an overlap would be the intents "paint red" and "paint green". In both cases, the user wants to have something painted, so the intents overlap and are almost impossible for the NLU system to distinguish. The solution to this problem is to combine the intents into a single intent and to achieve the differentiation of the colour with suitable entities. In cases where intents overlap, it is very often possible to merge them and distinguish them later by using corresponding entities.

To mention a disadvantage compared to the grammar-based system, multiple entities of the same type cannot be easily distinguished. As an example, take the sentence "The square should be glued to the ball". The system will recognize the two entities "square" and "ball", but cannot put them in relation to each other. To find out what should be glued where, it is necessary to use other suitable methods at a later step.

Finally, there is the question of how the NLU system can recognize intents and entities from speech. First of all, a general language corpus with example sentences is needed, which contains many sentences of a similar type and style in the language that the system is intended to understand later. For example, if a system is to understand news articles, a corpus of many news texts is suitable. With this corpus a statistical model is trained. The system first learns general characteristics of the language, such as articles and word order in a sentence. Almost all systems offer pre-trained models for many languages. Subsequently, it has to learn the actual intents and entities for the concrete use case.

To learn these intents and entities, one needs a collection of the intents expected in the use case and an annotated corpus of example sentences. The sentences are possible formulations of an intent in natural language, and one should find as many of them as possible. On this corpus of sentences, the entities which should be recognized are annotated. With this collection of sentences the model is trained again. Now it should be able to automatically assign freely formulated sentences to the correct intent and extract available entities. A schematic diagram of how an NLU system works is given in Figure 61.

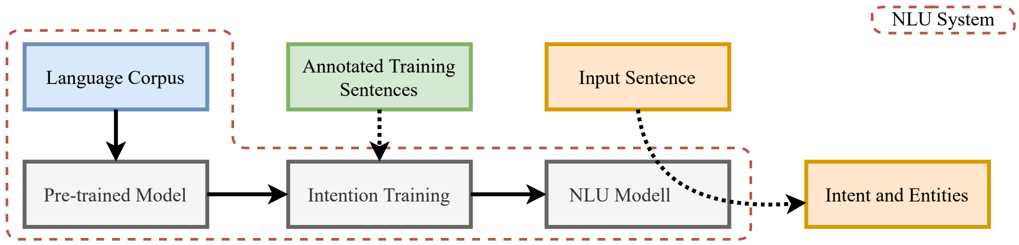


Figure 61: Functional concept of an NLU system.

A major advantage compared to grammar-based systems is that it is relatively easy to learn new intents. There is no need to understand and extend complex grammar, but just to formulate and annotate example sentences. This allows even non-experts to expand such systems without much effort. But there are certain differences between the different NLU systems in the handling of the entities. Some of them require a predefined list of all possible values of an entity, while other systems extract the values dynamically from the sentences. Again, other systems require a predefined list, but it can be dynamically extended at runtime. As a consequence, a general model for training sentences should be as flexible as possible to cover all these cases.

5.1.2 Generate Training Sentences

As already mentioned, different NLU systems require different information and formats for training. Therefore, I am introducing a general format for annotated sentences, so that a corpus of annotated sentences can be used to train all systems. These annotated sentences are stored within the associated tasks that are triggered by them.

In my format, the entities in the example sentence are replaced by a placeholder. There is a global list of all entities and their possible values so that these can be replaced sensibly. From this, all possible requirements for the different NLU systems can be generated later on. Resultingly, the writing effort is reduced to a minimum.

5.1.3 RASA NLU

The first system I have dealt with is the open-source RASA NLU [32]. This approach processes only written language and extracts the intents and entities from it. RASA runs as a service and can be accessed via an http interface.

To use RASA, you can simply download and install the software. Then you have to set up a so-called pipeline that defines how the data should be processed. Depending on the application and language, there are different ways to get the best possible results. An example of such a pipeline is given in Figure 62. After RASA has been configured, select the pre-trained language model to be used, for example MITIE or spaCy.

Now the system can be trained. For this purpose, the example sentences are converted into a format readable for RASA. RASA mainly needs many example sentences per intent.

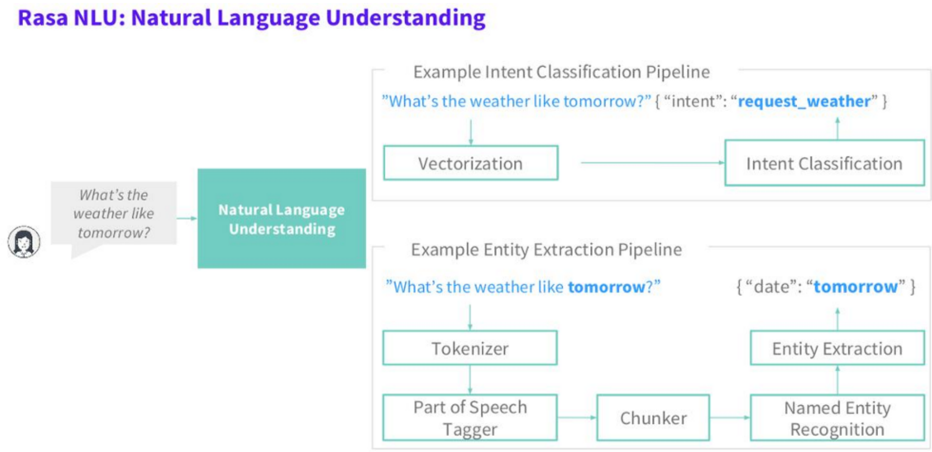


Figure 62: Internal processing of a sentence in RASA NLU to intent and entites.

The entities are marked in these example sentences. There is no separate list in which all values of an entity are listed. Therefore, I generate as much example sets needed so that each value of an entity occurs at least once. After the training data has been transmitted via the http REST interface, there is no graphical interface for the NLU system, and after a short training period, the system is ready for use.

Over the REST interface, written language can now be transmitted. But since we work with spoken language, this is a problem; the speech must be processed by a speech recognizer beforehand. I use the Microsoft speech recognizer. It runs in free txt mode and can recognize and write down almost any speech, without limitation. The output of the speech recognizer is connected to RASA so that the transcribed speech is sent directly to it. Finally, RASA returns the intent and entities as JSON objects.

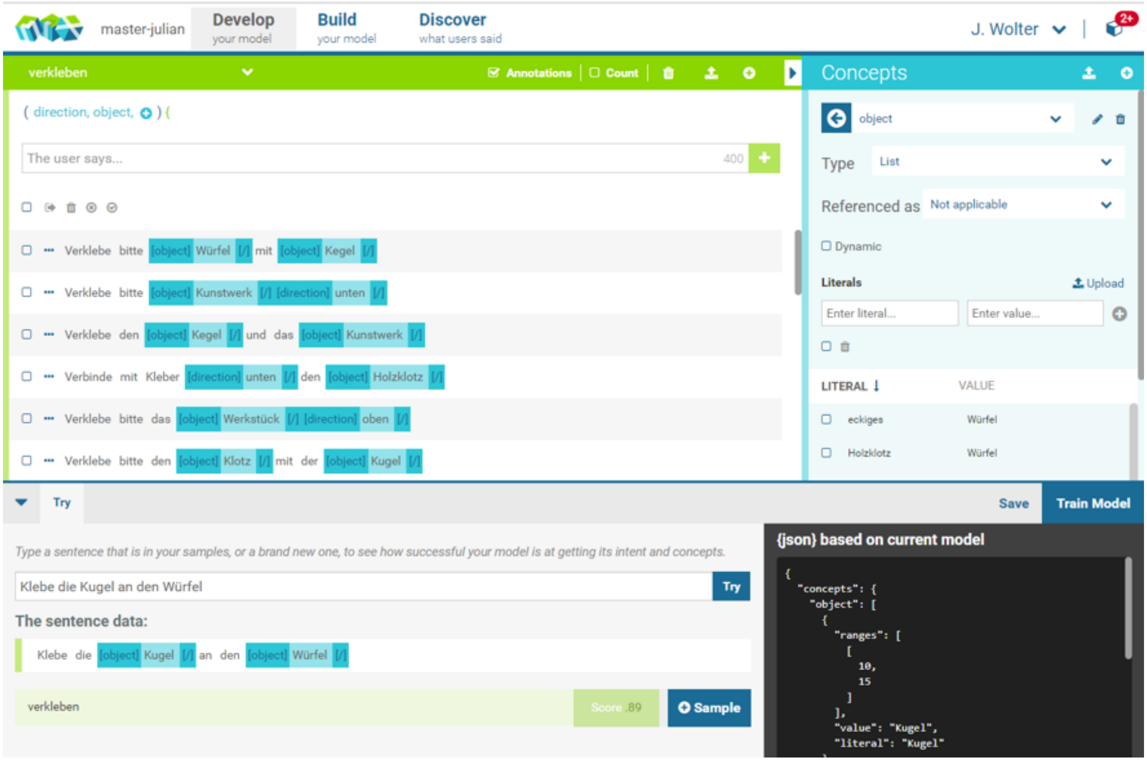
This approach unfortunately led to some problems, which is why I later adopted another solution. The biggest problem was the performance of the speech recognizer. Since RASA and the Microsoft speech recognizer cannot exchange any information about the language model, sentences are often transcribed incorrectly. For example, *"Kugel"* (ball) sometimes becomes *"Nudel"* (noodle). When such an error occurs, RASA usually has no chance to correct it. However, if the speech recognizer had at least basic knowledge about the character of the sentences, such errors could probably easily be avoided.

An alternative solution would be to speak the example sentences several times instead of delivering them as a text file, and to train all possible error codes over it. But this would be very time-consuming and thus relatively impractical. Also, the mistakes made would probably differ slightly from person to person, so it is difficult to cover all cases.

As it turned out, the best solution for me was to switch to a system that already integrated the speech recognizer into the NLU system. This allows both components to share their language model and thus better avoid incorrect recognitions.

5.1.4 Cerence Mix

Another NLU system is Cerence Mix (formerly Nuance Mix), which is available as an online platform. The environment is already completely preconfigured, you simply select the desired language and you can start right away. There is a web interface where you can enter the annotated sentences, as shown in Figure 63. It is also possible to continually test and debug the language model via the web interface.



{
  "concepts": {
    "object": [
      {
        "ranges": [
          [
            10,
            15
          ]
        ],
        "value": "Kugel",
        "literal": "Kugel"

Figure 63: Example of annotated sentences in Cerence Mix.

Unlike RASA, Cerence Mix manages all entities and their values in separate lists. This means that there is no need to duplicate example sentences for every value of an entity. After you have created all entities, you create the intents. For each intent, you can now store example sentences in which you annotate the previously created entities. Cerence Mix also offers additional functions to simplify the creation of sample sentences, such as phrase groups. In my case, however, I will not use these functions since I will generate the example sentences from my data. In addition to input via the web interface, it is also possible to provide the example sentences as a file that can be uploaded with a mouse click. I use this way for my application; the file is generated automatically and can be uploaded by the user afterwards. Unfortunately, Cerence Mix does not have an API interface for this yet.

After the training data has been provided, the NLU application must be built and deployed. Afterwards, it is available via a WebSocket interface. In contrast to RASA, not only written language but also spoken word can now be processed. For this purpose, the audio stream of the microphone is simply transmitted via the WebSocket connection. Cerenece Mix evaluates this and returns the intent and entities as JSON objects. Compared to the first approach, the error rate of the speech recognizer is noticeably lower.

5.2 Dialogue Manager

So far I have only discussed the NLU system, but without a complete dialogue management it is not very useful. In the following chapter, I will deal with just that. I will outline the basic mechanisms of dialogue and how they are implemented.

5.2.1 Role of Dialogue

The dialogue as the only mode of interaction with the user. It is the central interface between man and machine and it is what makes it possible to execute tasks. Therefore, the task of the dialogue manager is to assign the user's request to a suitable task. It must also ensure that the user provides all necessary parameters for the task. For this purpose, there are several options, such as further inquiries or an intent recognition. An overview of the architecture is given in Figure 64, which I will explain step by step in the following.

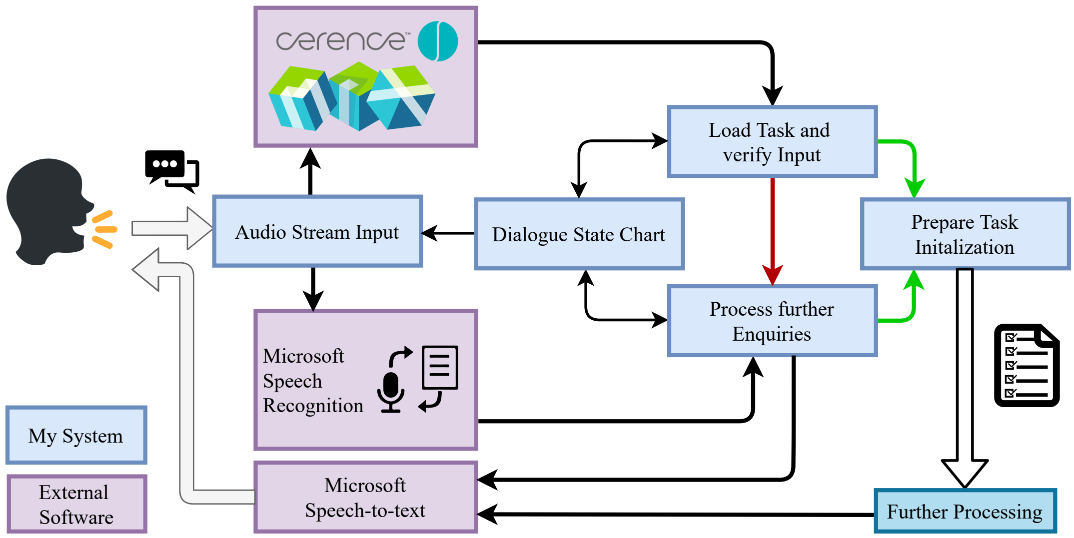


Figure 64: Architecture of the dialogue manager.

5.2.2 Voice Input and Output

A fundamental part of a dialogue is the input of the user's language, in this case via a microphone. For my system, I use a button that has to be pressed for the microphone to be active. As a result, the system knows exactly when it has to process speech from the user and when it does not. The system also knows whether or not something is addressed to it and when not, e.g. because the worker is talking to a co-worker. So I chose this approach instead of an always open microphone where the system has to decide whether to listen or not.

Now that an audio stream has been received via the microphone, it must be processed. There are basically two ways to do this: once is via the NLU system and the other via the dialogue manager. Based on the current state of the dialogue, the system can determine this at any time.

If the dialogue is at the beginning and the basic intent of the user has to be determined, the audio stream is encoded into a suitable format (e.g. OPUS) and sent to an NLU system, in case of Cerence Mix via a WebSocket interface. The result of the analysis is sent to the dialogue manager, who then decides what the next steps are.

If the dialogue is in a state where no intents have to be recognized, speech is converted to text form via the Microsoft speech recognizer. This is usually the case when the system has queries to the user, e.g. to obtain all parameters for a task.

The Microsoft speech output engine is used for speech output. The reason for this was the easy integration into the existing system. Cerence Mix also offers a speech synthesis service, but this sometimes had a very long delay, which disrupted the continuity of dialogue.

5.2.3 Validating Input of the User

An important point to consider is the validation of user input by voice. Since speech recognition and intent recognition do not work perfectly, input may be incorrect. This could have negative consequences if, for example, an object is painted in the wrong colour. It is therefore important to check the user input for reasonableness. However, the system should avoid repeating every single piece of information to ask for confirmation because that would be quite annoying to the user.

The quality of the output of the NLU system can be classified quite well, since a probability is transferred as well. You can use this value to create three categories. The first one is a value above 80%, where the detected intent is taken over directly without further questions. If the value is below but still above 50%, the user is informed of the detected intent and then asked whether this is the desired intent. The user can then give feedback to the system by simply saying yes or no. If the value is even lower, the recognized intent is discarded completely and the user is asked to communicate his intent again, but to adjust the wording slightly.

Unfortunately, this is not so easy to do when processing the results of the Microsoft speech recognizer. But even here there are possibilities. On the one hand, this recognizer is only used if the system queries the user. This means that the number of possible, valid answers can already be limited, since the start parameters of a task can be provided with types and conditions (in the form of concepts that are inherited). This already prevents substantial mistakes, so that, for example, the question about a location cannot be answered with a colour.

But there are also other mechanisms to avoid incorrect inputs. The process of creating and initiating a task is designed in such a way that incorrect input can too be caught early. Due to this design, incorrect entries can already be detected. If, for example, an object is requested from location x, but this object only is available at location y, this error is detected before it is executed. This allows the user to correct his mistake at an early stage and no unnecessary robot actions are triggered, which might even cause damage. The task flow will be described in more detail in chapter 7.

5.2.4 Further Enquiry

When a user starts a task, it is generated via an intent with associated entities. If not all required parameters have been transferred as entities, the system actively requests them. The reason for this may be that the user simply forgot to provide this information. Or the NLU system simply did not recognize the corresponding entity.

The requests are generated automatically and are based on the parameter specifications in the task. If, for example, an entry of the type location is missing, a location is explicitly requested. For this purpose, all parameters are additionally annotated with a name, that is used for the enquiry. The evaluation of the user input is then carried out using Microsoft speech recognition and, as mentioned in the previous chapter, is further processed and checked for validity. As soon as all parameters have been assigned a value, the corresponding task can be processed further.

5.2.5 Grounding

Since we want to build the most natural language interface possible, it is important to be able to understand as many sentences and words as possible. We often use relative specifications in our language that cannot be resolved without special processing, such as location information like *top* or *left*. Therefore, it is important to implement grounding mechanisms that can correctly identify the meaning in such cases. Grounding has been implemented for two types, for location information like *right* and *above* and for last used objects.

In both cases, a discourse memory is used which stores the locations and objects that the last interactions happened with. The work space is virtually divided into many squares. Each object fills one such square. If a location is now used, for example to indicate where a screw is to be placed, the desired location can be identified with the help of the location stored in the discourse memory as well as the relative specification of the user. It works in a similar way with objects. For example, if the order "Colour it red" is given, "it" is mapped to the object with which the robot last interacted with. The object is determined with the help of the discourse memory. The grounded locations or objects are now simply passed as parameters to the task.

This implementation also makes it possible to ground entities, regardless of the tasks. No special annotation is necessary. As a result, all tasks benefits from the grounding mechanism without the need to pay special attention to it.

5.2.6 User Interaction during Task Execution

During the execution of the task, it may also be necessary to have a basic form of user interaction. For instance, it is useful to inform the user before unexpected movements of a robot happen. For longer tasks, information about the current status can also be useful so that the user can better plan his time.

For this reason, *User Interaction Nodes* offers the possibility to provide the user with further information during the execution. In my version, only voice output is integrated. The system cannot request any further information during task execution. The reason for this is that the tasks should be processed as autonomously as possible in the background, without the user having to worry about them after they have been generated. For this reason, stong validation of the input is valuable, so that it is ensured that all necessary information is available at the beginning and that no further queries are necessary during the execution.

Chapter 6
Task Generation, Assignment and Execution

Now that the knowledge has been modelled and the user interface is in place, I will explain in the following chapter how the initialized task is finally transformed into robot commands.

6.1 Life Cycle of a Task

As a first introduction, I want to give an overview of the process that takes a task from its generation to the physical execution. For this purpose I have outlined the development of a task in Figure 65, beginning with task generation via natural language, and ending with the actual robot commands.

First of all, the task is initialized with the help of the speech interface. Afterwards, the task is generated, which means that a working copy is created and the variable memory is initialized. Since it is possible to call subtasks, they are unfolded in the next step, so the working copy of the task has no more calls. Since the start parameters are known, the task can be simulated in order to check the validity of the parameters. Now that the unfolded task is ready, the requirements for the later skill matching can be calculated. Next, the system finds and validates suitable solutions using the pool of available robots. If there are several possibilities, additional cost metrics determine the optimal solution. Then the robot nodes are assigned to concrete robots and unneeded execution paths are disabled. The systems then queue the task, which will result in the execution of the individual steps and finally, robot commands.

In the following chapters I will go into detail about how the individual steps work.

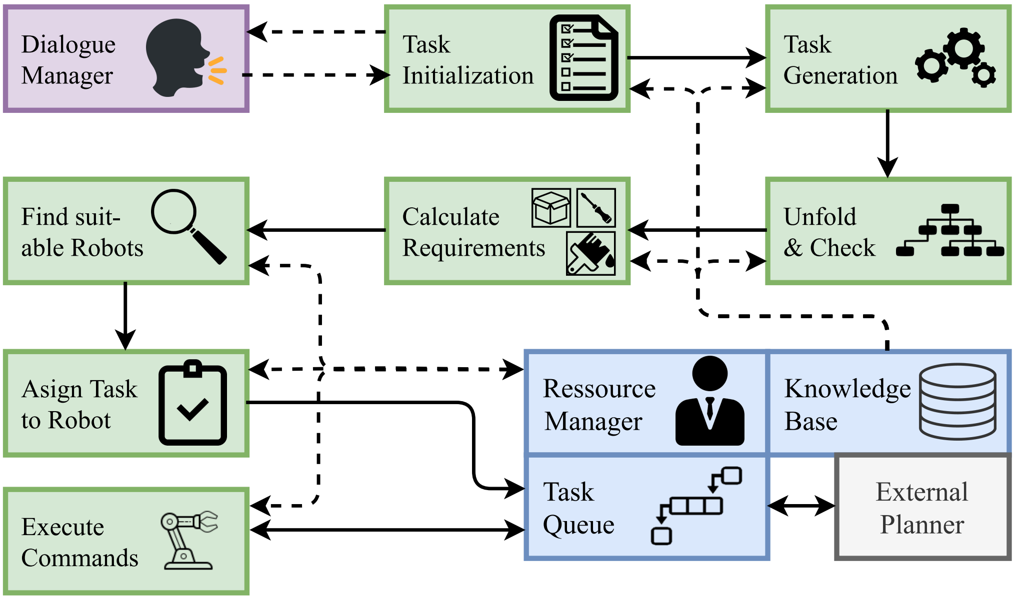


Figure 65: Lifecycle of a task

6.2 Initialization, Generation and Task Requirements

After the dialogue system has recognized an intent with entities from the language input as explained in the previous chapter, the task must be initialized. At this stage, the detected entities are checked for validity, among other things. Then the task is generated, which includes loading subtasks, but also generating the requirements and performing plausibility checks. I will discuss these steps in detail in the following paragraphs.

6.2.1 Initializing and Generating a Task

After the dialogue manager has recognized an intent, the system searches for an appropriate task among the set of modelled tasks. Once one is found, the input parameters are loaded; the dialogue system must provide a value for each of these. Since the values of the entities can in part be recognized freely, they have to be tested for reasonableness.

When defining the task, each parameter can also be given a type that can be checked. There can also be restrictions on the knowledge base, too. So it is possible to define that a passed object must inherit from a certain concept.

When all start parameters have been declared valid and are passed on, the task can be initialized. If there are problems, an error message is returned to the dialogue system. The message will be used for further dialogue with the user so that he can repeat or correct his input. If no further queries to the user are necessary, the initialization by dialogue manager is complete.

After successful initialization, the task is generated. During this process, a copy of the task is created, and variable memory is allocated to be used later for execution. The input parameters are already stored in the variable memory. This allows the task to be modified and tweaked in later processing steps without any changes in the knowledge base.

6.2.2 Unfolding a Task

Recall from Chapter 4 that a task can be composed of several (sub)tasks. All calls to subtasks are resolved as a next step. To achieve this, all called subtasks are loaded and inserted at the appropriate places in the graph representation. It is crucial that a new variable memory is generated for each call, which only contains the input parameters. A difficulty is that subtasks also have to be initialized and generated now, but their start parameters may not be known yet. For instance, the input parameters could depend on another subtask and only become known later on in the execution. To solve this problem, the task is simulated once.

First, a temporary copy of the knowledge base is created as the knowledge base should not be changed by the simulation. For example, the robot node *paint* changes the colour of an object. After the copy is created, the task is executed step by step with the start parameters. When a (sub)task call occurs, it is initialized and generated as described in the previous section. If a start parameter does not fit, an error is returned, which is communicated to the user via the dialogue system. If other problems occur during the simulation, e.g. a desired object does not possess a required property in the later course, this is also noticed in this step. In addition, subtle problems with the user input can be recognized in this way, for example that an object is requested from a position where it is not located.

If the simulation is completed successfully, it is ensured that the task can be solved in its current state and that all subtasks are called correctly. Of course it can happen that between simulation and actual execution the environment changes to such an extent that a correct execution is no longer possible. This case will be treated separately later.

Another issue could be that the task results in an endless loop. Since it is impossible to predict whether a task will terminate or not (see the halting problem [33]), the simulation is aborted after a predefined timeout and declared as failed. One way around this problem would be to define the graph acyclically and prohibit calling oneself in subtasks. This would make loops impossible and since each path of the directed graph ends in a final state, it would terminate safely. But as a downside, this would make many use cases difficult or even impossible, which is why I decided against these restrictions.

An example of a task being unfolded is shown in Figure 66. The task is shown at the top, the subtask that is used underneath it. If the task is now unfolded, the call is replaced by the actual task. Variables that are already fixed are also transferred. In the end, you only have one big task without any cross-references, which makes processing much more manageable.

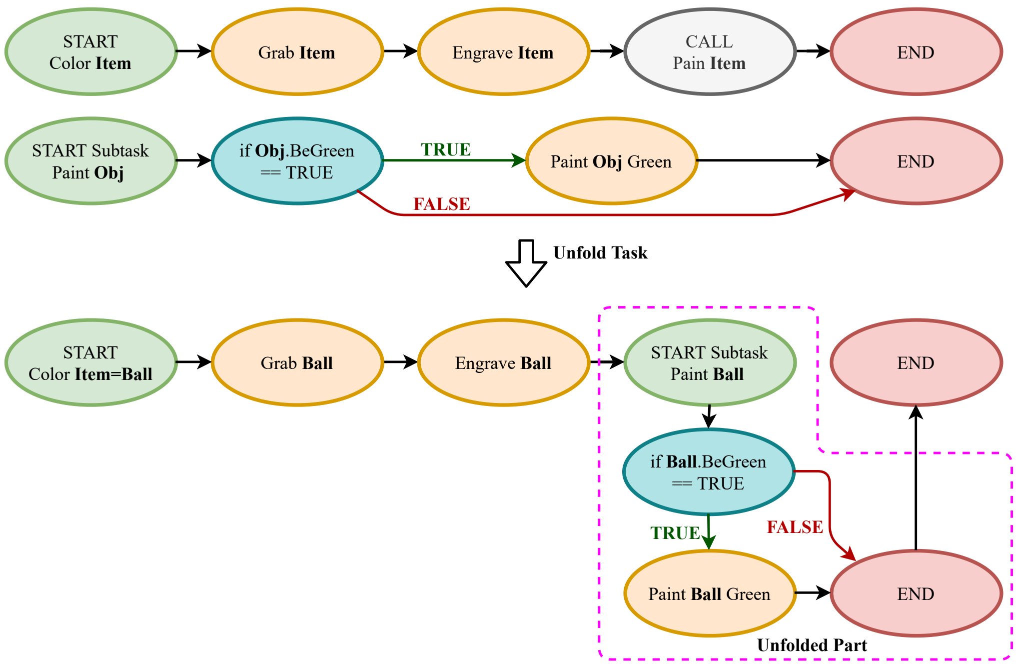


Figure 66: When the Task is unfolded, Task calls are replaced by the actual Task.

6.2.3 Computing the Task Requirements

Now that we have a complete unfolded task, we know all possible robot nodes for a path through the graph. These robot nodes ultimately indicate which skills a suitable robot must provide for the task. Using the definitions given in chapter 4, a suitable task requirement can be generated for each path.

For this purpose, the task tree is processed from bottom to top from each final state. A new task requirement is created for each branch. This results in a list of several task requirements at the end which represents all possible execution paths of the task. This representation allows to later find all possible robots that could solve the task.

An example for the determination of these requirements is shown in Figure 67. The task was already reduced to the robot nodes and the three possible paths were drawn in. Now for each path, the task requirements are determined, which look like the following.

$$task\_requirement_{red\_path} = \{\ \{\ (engrave, \{Location1\}), (move, \{Storage\})\ \}\ \}$$

$$task\_requirement_{green\_path} = \{\ \{\ (paint, \{Location1\}), (move, \{Storage\})\ \}\ \}$$

$$task\_requirement_{purple\_path} = \{\ \{\ (engrave, \{Location2\}), (move, \{Location2, Storage\})\ \}\ \}$$

Another, more detailed example is given in the chapter 7.4 (Scenario - Task Generation and Robot Assignment). The next step is to find suitable robots for each task request.

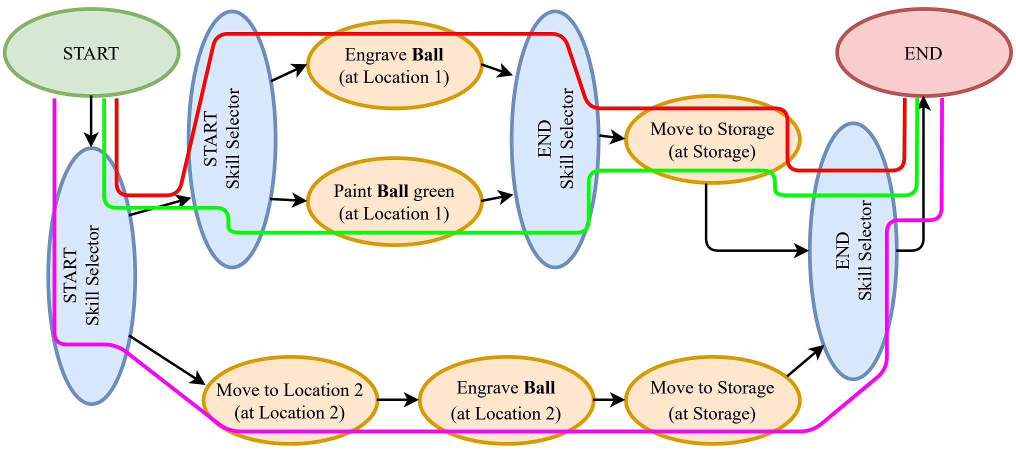


Figure 67: To compute the task requirements, the three paths (red, green and purple) are considered.

6.3 Skill Matching and Assignment

Now that we have initialized a task with the help of the dialogue system and processed it so far that we know all individual requirements, the next step is to assign tasks to suitable robots. I will explain this selection in the following chapter and how a decision is made, in particular if there are several options to choose from.

6.3.1 Finding suitable Robots

All available robots are stored in the knowledge base. Therefore, we use it as a basis for finding possible assignments. Of course, most likely not all robots are available and some have to be filtered out, but this will be part of the planning step later on. For now, we will consider all possible skill-based assignments.

First, we select a path and load all robots from the knowledge base. Then a check is made to see which robots provide only unneeded skills and these are removed directly. A suitable robot is then iteratively assigned to each robot requirement. A robot can be assigned more than once, even if the task would be processed more slowly with such an assignment. Nevertheless, if a better solution is available, it will be preferred over this assignment later during task planning. In this phase, however, it is only a matter of exploring all possibilities. Since each path through the tree has its own task requirements, a separate list of robots that satisfy the requirements is created for each path.

After all task requirements have been mapped to (several) collections of robots, validity must be checked in detail. Since we have already simulated the task and thus know the parameters for all robot nodes, this is now feasible. In some areas robots are unfortunately not yet as flexible as a humans, for example, gripping an object is usually only possible if the robot has been trained on exactly this object. For this reason, the robot service asks whether the robot can perform the specific task with the specific parameters.

6.3.2 Assignment of Task to Robots

Now that we have a list with all collections of robots that would be capable of solving the task, the system must choose the best of these options. If there is only one option in the first place, the solution is trivial. As this is usually not the case, though, I introduce a heuristic cost metric for each possibility in the following to facilitate making a good choice.

To do this, you must first determine what is a good choice and what is a bad choice. The most evident factor is that the user wants his task to be completed as quickly as possible. However, if one would only pay attention to this, the result would be that one task would possibly bind a relatively large number of robots and other tasks would be processed more slowly. This reduces the overall efficiency in the factory. In consequence, other factors must be taken into account in any case.

To weigh in other factors, I introduce a cost value for the robots. The more costly a robot is, the more valuable it is for manufacturing. Intuitively, strategically important robots will be costly to use and only taken when they are really necessary. To determine the cost of a robot, I have taken into account several parameters, which I list below:

- **Amount of Skills**
  A robot with many skills is much more valuable for production than a robot that can only perform single actions. Thus, such a robot has a higher cost value.
- **Unique Skills**
  If a robot has abilities that few others have, it will also be more costly.
- **Highly used Skills**
  Robots with skills that are used very often are also more costly compared to robots that have skills that are rarely used.
- **Location of the Robot**
  If the robot has to travel to its place of use first, this also increases the cost. Depending on the task, these are already included in the total time of the task, but do not necessarily have to be, e.g. if the robot moves parallel while another robot executing commands.

The first two parameters can be easily measured using the knowledge base. Although the last parameter can be determined at the time of generation, it can change during execution. Hence, I chose to only include it with a very small weight. The third parameter can only be determined automatically in the course of time, because only then is it clear, which tasks are performed frequently and, consequently, which skills are needed frequently. Alternatively, the user can specify manually which tasks he will perform frequently in the future. With the help of four mentioned parameters it is now possible to associate a cost with each robot. Calculation of all costs is performed once at the beginning because they only change if robots are added or removed.

In order to be able to estimate the runtime for each variant, each robot can give an estimate of the expected time that each of its scheduled tasks will take to be executed. If a robot is currently performing another task or is already scheduled, the time until this task is completed is added to the total time. This results in the total runtime for each robot to complete the task. This runtime per robot is now multiplied by the cost of the specific robot. Then all costs of the robots of this solution are added up and you get the total cost of this selection. Finally, the different solutions can be compared in terms of their costs. The most economical solution in our system is the one that is selected later.

6.3.3 External Planner

Naturally, the metric I propose is only an approximation of an optimal solution. Surely there are cases where my system does not act optimally. Finding the ultimate perfect solution is usually not possible, as its calculation is simply too complex and would take too long. Almost always this is not necessary either, since the best possible approximation is sufficient.

Algorithms that find such almost optimal solutions are usually very complex and a large research area, which is why I have kept to a rather basic approach here. In order not to re-invent the wheel again and again, it makes more sense to integrate an external, already existing planner into the system instead of developing everything from scratch. As a refinement, it is possible to have the planning solved by an external software via interfaces.

6.4 Execution

Now that we have assigned a task to one or more robots, we at last can execute the task and generate robot commands. To do so, I first compare the approaches of a highly autonomous robot and a central planning instance, and then present my approach.

6.4.1 Autonomous Robots vs. Central Planning

An autonomous robot is a robot that receives the complete task at the beginning and can then perform it entirely of its own accord. There will be no further communication with the central planning instance once subtasks have been successfully completed. The significant advantage is that the robot is this way completely independent of external systems as soon as the task is transferred. For example, if the network is disrupted or the planning instance crashes, the robot can still finish its task. In addition, there may be a privacy advantage since not all minor details are recorded by a central instance, e.g. how quickly a single worker cooperatively performs specific tasks.

However, such an approach also has considerable disadvantages. Since there is no longer a central planning instance, it is no longer possible to react on the fly to changes in the general situation. For example, tasks can no longer be combined by an external planner, so that one transport robot is used for two different tasks. Moreover, most factories are already automated and rely on a stable network and infrastructure. A failure of this infrastructure is improbable and would bring the entire operation to a standstill anyway. Besides, the system would no longer be well predictable. For example, changes made autonomously by a robot to a task can no longer be tracked. Central collision avoidance would also no longer be possible, as each robot would look out only for itself.

In contrast to the autonomous robot presented above, the robots in the context of my work are completely dependent of a central planning instance. Only the smallest possible subtasks are assigned to the robots. Once this subtask is completed, the system must allocate a new subtask to the robot. This gives the system maximal control over the robot at all times.

To sum it up, I would say that a fully autonomous robot (regarding planning and execution) in an industrial environment is generally not desirable. The purpose of such a robot is more suitable in the field of search & rescue missions, but not for industrial applications.

6.4.2 Communication between Robots and the Central Planning Component

Having chosen a central planning component, I now have to define how the concrete subtasks will be sent to a specific robot. In the current state, the abstract tasks are only assigned to robot instances, but the commands still have to be generated and transmitted to the correct robots. For this purpose, there are robot services that provide an interface for each robot type. These services transform the abstract commands into commands that the robot can understand. These robot services can either be implemented directly in the software or in the network through external services. An overview of the system is given in Figure 68.

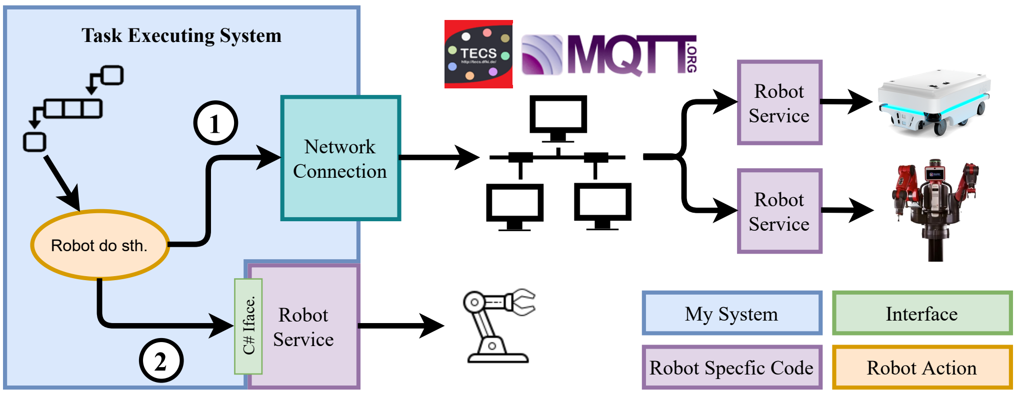


Figure 68: Robot Services translated the robot action nodes into the specific robot commands.

If the services are located in the network, I have implemented two interfaces that can be used. On the one hand, there is the TECS framework known from the HySociaTea project [34]. It is based on the Apache Thrift framework and uses the Thrift IDL to define data structures. These defined structures can be automatically translated into a variety of different programming languages, such as C#, Java, Python, etc. There is a central TECS server to which the clients can connect and which communicates on an event-based level. The clients can communicate with each other via a publish-subscribe mechanism, remote-procedure-calls and message-passing techniques. The previously defined data structures are transferred, and the generated classes in the programming language are automatically filled with the values. On the client-side, robot services

Figure 69: Logo

can use UDP multicast for automatic configuration. This makes it possible to easily embed and automatically integrate various robot services in different programming languages.

As an alternative, there is a MQTT connection, e.g. by using the Mosquitto broker [35]. Also here all robot services are connected to a central broker. The robot services log on to the system via predefined data structures and topics and can provide their services. Only the event-based publish-subscribe model is used for communication since MQTT supports only this model. JSON is used to represent the data during transport.

Even though MQTT offers fewer features than TECS, it has the significant advantage that it is already an industry standard. As a result, the protocol has already been tested millions of times and has been used productively hundreds of times productively in industrial environments. This means that there is already a considerable amount of expertise, and possibly there are already highly redundant MQTT brokers installed at the factory so that existing infrastructure can be used.

6.4.3 Error Handling

Since robots are not perfect, errors can occur during execution. There are different classes of errors; on the one hand, there can be a mechanical defect of the robot, which makes further operation impossible. On the other hand, the route of a moving robot maybe be blocked, for example. In this case, the problem could be solved by replanning the path or instructing a human to remove the obstacle. There are also countless other different classes of error, e.g. an arm that does not grip an object correctly, a human that enters the safety area of a robot and so on. The software can correct some of these errors (e.g. replanning the route), others require the assistance of a human (e.g. removing an obstacle or driving the robot manually). Others still make execution altogether impossible (e.g. technical defect of a robot).

A simple error handling is possible by simply triggering a different transition in the task model. For example, the robot task "move" can have three transitions: one for successful execution, one for a navigation error and the last one for other errors. In this way, basic errors can be solved together with a worker, e.g. in case of a navigation error. The system can generate a voice output which instructs the worker to check the route. However, more complex or intelligent error solutions are not possible, not to mention a transfer from the machine to humans.

Considering this is a whole field of research, I will not deal with it in my system. If a complex error should occur, external software has to decide what the best solution is. For example, the TRACTAT project [36] addresses, amongst other things, this transfer of control from machine to human in case of failure and develops strategies for the best possible solution. For example, in his master's thesis, Umar Farooq [37] presented a system that uses multimodal presentation to assist a human worker which resolves a problem of a robot.

Chapter 7
Scenario

Now that I have presented my system in detail, I will show its capabilities by modelling a real-world example. For this purpose, I take the MRK4.0-Lab as a reference to a real workplace. I design a short scenario and model the relevant parts of the environment in my framework. Afterwards, appropriate tasks are modelled which can then be generated and executed by my system. However, I will not implement any robot services, meaning that the execution of the tasks is only simulated and no real robot commands are executed, as this would go beyond the scope of this thesis. Finally, I will present examples of possible applications and conclude.

7.1 MRK4.0 Innovation Laboratory

The MRK4.0-Lab is an innovation laboratory for human-robot collaboration in the industry located in Saarbrücken [38]. It was founded in 2016 as part of a German-Czech partnership. The laboratory aims to explore and test the cooperation between man and machine in a realistic environment.To explore the possibilities in such a collaboration it is equipped with many different robots and sensors. For a first impression, I have added Figure 71, which shows the robot area of the laboratory.

Figure 70: Logo

For my scenario, I will not give details in my thesis on all existing robots and objects in the lab, but only those that are relevant for my example scenario. To show them, I created the map shown in Figure 72, in which all appropriate locations and robots are marked. What you immediately notice is that the plan is divided into two areas. This is because the drone can only fly in the yellow area of the workshop. However, moving robots can travel to all regions (yellow and green). The positions of

Figure 71: Picture of the MRK4.0 Laboratory

the stationary robots are marked on the map, as well as the storage and the dedicated workplace where the worker is located. I will describe the individual functions and skills of the robots in more detail later.

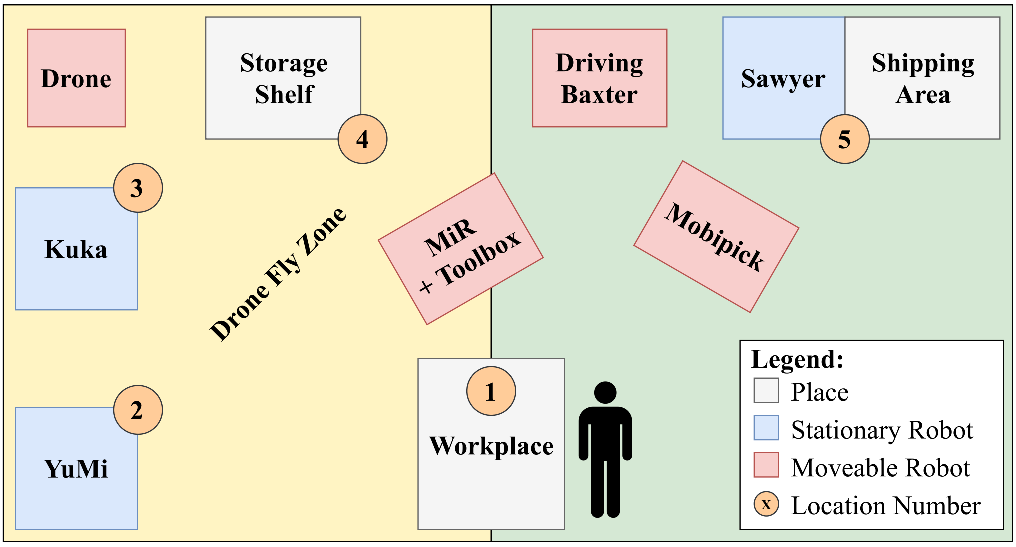


Figure 72: Map of the MRK4.0. Notice that the drone can only fly in the yellow area

7.2 Example Scenario: Individual Wristwatch Production

To focus as much as possible on the practical aspects, I apply my system to an example scenario with the goal to produce wristwatches which are highly individual. To fabricate them, an employee works at a central workstation, and from there also controls the robots by voice. An image of the workplace is shown in Figure 73. This workplace has a wide range of options for material delivery and collection by robots. Since it also extends into the flight zone, the drone can also be used for material transport.

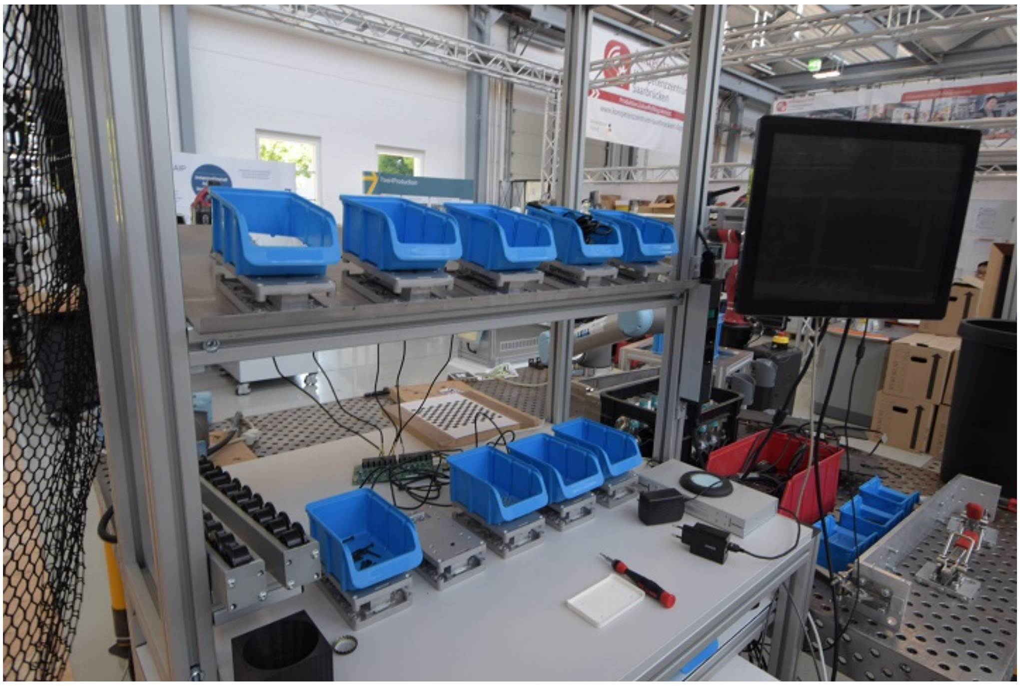


Figure 73: Workplace

For reasons of practicality, I do not reproduce a real watch production but rather simplified the process considerably for reasons of practicality. The watches produced in my scenario consist of four components, which have to be adapted to the individual wishes of the customer. A depiction overview is given in Figure 74. All these components are located in the storage shelf and must be picked up and joined together. They can be joined together in one or more ways, e.g. by screwing or glueing. The workflow is always identical. First, the worker takes an individualized housing, in which a clockwork is integrated. Afterwards, the case with the clockwork is sealed with a cover glass. Then the bracelets are attached, and the watch can be shipped. Between the steps, it is possible to perform a robotic inspection to check the intermediate product for defects.

Naturally, there are some restrictions; for example, some clockworks have to be screwed while others are glued. Also, not all bracelets are compatible with all cases. To adress this, I will show in what follows how how such constraints can be modelled in my approach. I will also show how the same task can be solved with different robots.

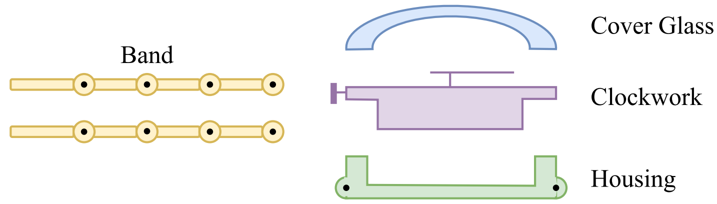


Figure 74: Wristwatch components

7.3 Knowledge Base

As a next step, the scenario presented above has to be modelled the environment. For the sake of clarity, I have divided this step into three parts. First, I will introduce the modelling of the objects along with the constraints on e.g. their processing. Then, I model the individual robots and and lastly, the actual tasks that will be generated and executed. In the following sections, I will explain the model by example and go into detail about some aspects worth mentioning.

7.3.1 Object Models

As already explained in previous chapter 4, a distinction is made between abstract concepts (TBox) and concrete instances (ABox). Abstract concepts can be used, for example, to designate specific properties via inherit. This also allows constraints to be implemented later, for example, that a certain clockwork may only be glued and not screwed.

First, we will take a brief look at an excerpt of the model. The part of the model that describes the the housing component is illustrated in Figure 75. First of all, the concepts that are within the red dashed line are part of the system and therefore already predefined. For the housings, a general *Housings* concept was first defined. This concept inherits the *LocationEntity*, which is later important for skill matching, as well as the concepts *GraspableEntity* and *StorableEntity*. The last two concepts are important so that the system knows that all housings can be gripped and stored, e.g. if a robot is to transport them from A to B. A more specific concept is then derived from the general concept of *Housing* per manufacturer. At this point, there already are first differences. Only the concept representing the manufacturer "Edelmeier" additionally inherits from the concept *GlueableEntity*. This makes it possible to glue the housings of this manufacturer to the clockwork later, whereas the same is not possible with the manufacturer Berger. Besides, there is the *GoldFinish* concept, will ultimately allow instances that inherit from it to be checked for scratches. Further properties or restrictions can be defined, e.g. that objects derived from GoldFinish can only be touched with a textile gripper and not with a metal gripper. With the help of these concepts, the three housings shown in the illustration can finally be modelled and will inherit the desired properties mentioned above. Similar concepts and instances can be created for the other components such as clockwork, cover glass and bands.

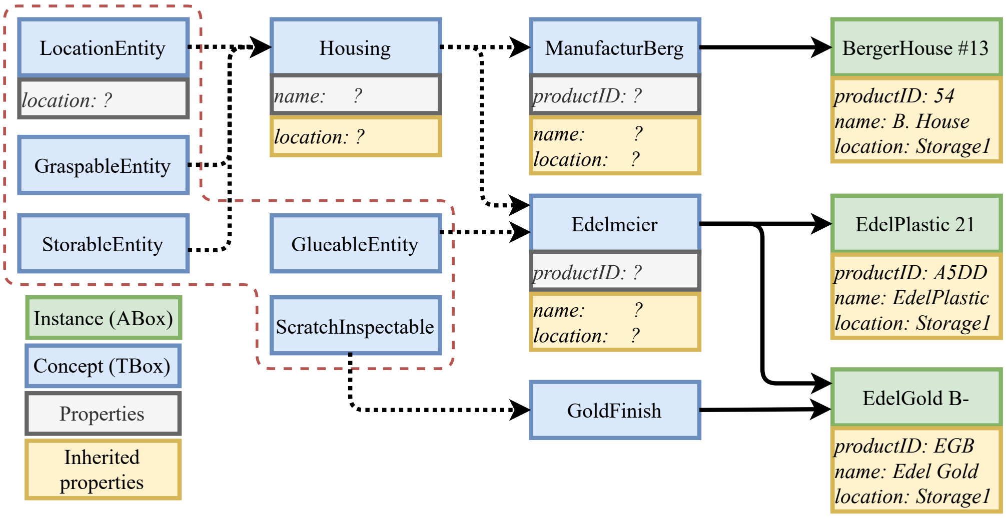


Figure 75: Extract from the representation of clock housings. The dashed red frame contains the concepts provided by the system.

Now that we have modelled individual objects, we still have to represent relationships between the objects. For this, we take a closer look at the housings and clockworks. For them to fit together, they must be the same size. Besides, it must be ensured that the joining method is compatible, e.g. that the screw fits the holes. To give an example of how to cover these two cases, I have drawn another part of the model in Figure 76. To simplify presentation, things, the inheritance of the *LocationEntity* for the *Screw* has been omitted as well as the substructure for the housings, which were already shown in the previous illustration.

First, we look at the part outlined by the blue dotted line. There, a junction between the housing and the clockwork as well as between the clockwork and the cover glass is defined. These respective components involved in the joint inherit from this interface specification concepts. This ensures that later, the later concrete instances contain the required values that represent the dimensions of the parts. In this way, it will later be easier to check whether the combination of two components fits or not simply by comparing the properties of these concepts.

The concepts in the green dotted frame are ensure for ensuring that only the correct screws are used for the joint between the housing and clockwork. For this purpose, there is the general concept *ScrewParameter*, which defines the essential parameters, such as the diameter of the screw, respectively the screw hole. All further concepts then inherit from this, since these parameters must necessarily match; otherwise, the components will not fit. If, for example, the hole is too large for the screw, the part cannot be used. This check can also be realized again by a simple comparison in the task, as in the previous case. However, the length of the screw must also be checked. To perform this check, the measures provided by the concepts *ScrewHole*, *Screwable* and *Screw* have to be calculated and compared to each other. But also this can be implemented by a quite simple computational node.

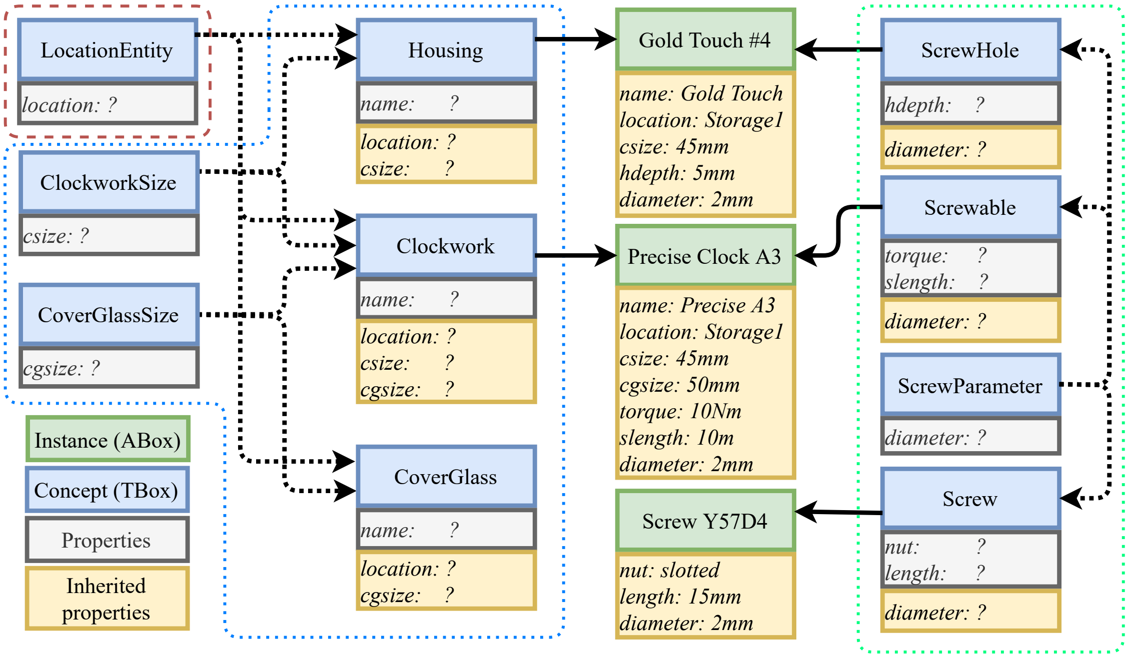


Figure 76: Modelled wristwatch components. The concepts in the blue dotted frame ensure that only components of the right size can be combined. The concepts in the green dotted frame ensure that they are joined together properly.

7.3.2 Robot Models

After I finished modelling the objects that agents can interact with, I now shift the focus to the robots. In what follows, I will present every robot with a picture and describe their abilities and function. In particularly interesting cases, I will also show a part of the model.

MobiPick

The MobiPick is an in-house development of DFKI. A MiR (Mobile Industrial Robots) platform was combined with an Universal Robotics (UR) arm. Besides, a shelf was installed on the MiR, which is equipped with many additional sensors and actuators. On this table, the UR arm is fixed, so that he can put stuff on the table and grab things from outside. For a better impression I have added pictures of the MobiPick in Figure 77 and 78. The combination of a moveable platform and a robotic arm makes MobiPick an exciting robot. For example, it can grab objects through its UR arm, store them on its table and then drive to another location and unload them there. This makes it ideally suited as a transport robot.

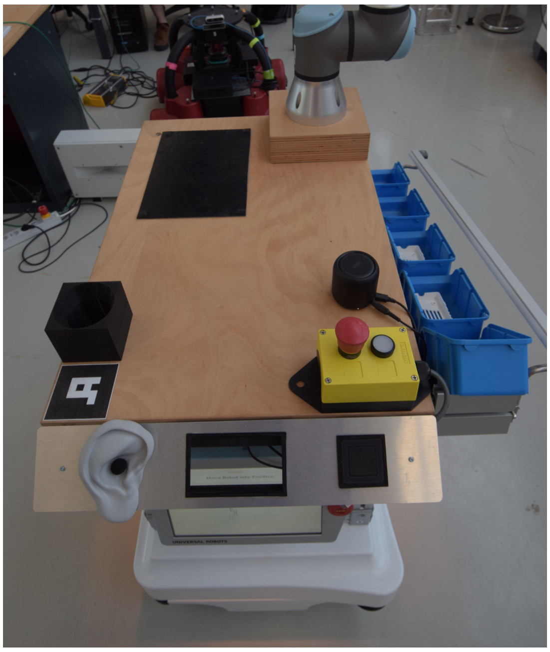


Figure 77: MobiPick

Figure 78: MobiPick

Because of the combination of these two robots to a new robot, the model also looks quite interesting. Therefore I have sketched the relevant part of the model in Figure 79. Most of this is already known from previous explanations in my work. But the interesting point here is that by combining two robots, we obtain a robot with an additional skill (see red arrow). This makes the combination of both more useful in the end.

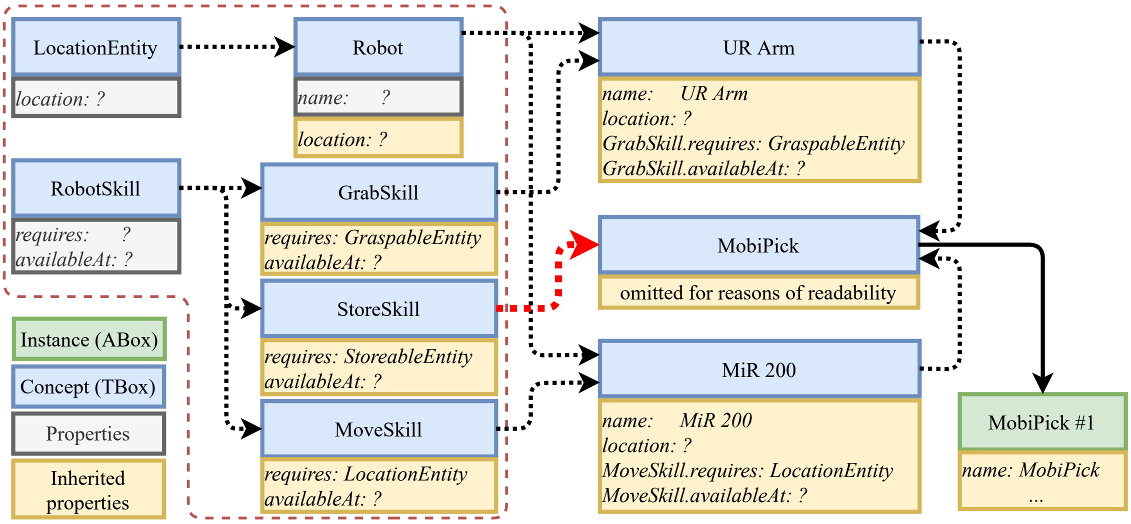


Figure 79: Model of the MobiPick. The combination of two robots leads to a new robot, which gains a skill that the single robots did not have (red arrow).

YuMi

YuMi is a robot from ABB that was developed explicitly for collaboration with humans and consists of a body with two arms, as you can see in Figure 80. Although it only has a relatively small working area, it can carry out very precise operations. The end actuators can be exchanged; in our case, one arm is equipped with a camera and the other with a gripper.

In this scenario, the YuMi robot will mainly perform two jobs and therefore have two skills. On the one hand, the camera can be used to inspect objects. With it, one can check, for example, whether sensitive surfaces are free of scratches or whether the clockwork has been placed correctly. On the other hand, YuMi has exceptionally high precision. It is thus capable of reliably glueing housing, clockwork and cover glass. It is assumed that the adhesive has already been applied to the components (e.g. by the manufacturer). This enables YuMi to precisely glue two parts together.

Figure 80: YuMi with camera for inspection and gripper for precise manipulation of objects

DJI Mavic 2 Drone

The DJI Mavic 2 drone, shown in Figure 81, can be used to transport small objects quickly. However, its field of operation is limited. In this scenario, the drone has the move and store skill, limited to the accessible locations. Restrictions regarding the weight are not encoded in the model but are instead captured when suitable robots for the task are selected.

Figure 81: DJI Mavic 2 drone with infrared markers

Sawyer

Sawyer is a robotic arm from Rethink Robotics which was also designed for human-machine collaboration. A picture of the arm is shown in Figure 82. In my scenario, Sawyer only has the *grab* skill and is used to pick up ready-to-ship packages from robots that deliver them and place them on the shipping table.

Kuka

The Kuka robot arm, shown in Figure 83, has a camera attached to its end actuator. This allows him to perform inspections, just like the YuMi.

Figure 82: Sawyer

Figure 83: Kuka

MiR Platform with Toolbox

The combination of the MiR platform with a toolbox is also a DFKI in-house development. A picture of it is shown in Figure 84. In terms of skills, this combination is quite similar to MobiPick; only the grab skill is missing.

Figure 84: MiR euqipped with toolbox

Baxter with Mobility Base

Last but not least, we have a combined robot from Rethink Robotics and Dataspeed Inc.: Baxter and the associated Mobility Base, which allows it to move freely.

Baxter was explicitly developed for human-machine interaction and was one of the first robots in this field on the market in 2011. The Sawyer arm later replaced it. The operating system used to control Baxter is the open-source ROS. Baxter has two robotic arms, a central display and many sensors so that it can be operated safely without a cage. The end actuators are interchangeable, but in our scenario, only simple grippers are used.

With the help of the Mobility Base, the previously stationary Baxter becomes a freely moving robot. Thanks to Mecanum wheels, the base can move freely in all directions, i.e. forwards and backwards, but also sideways and turn around on its axis. This makes it much more flexible than the MiR.

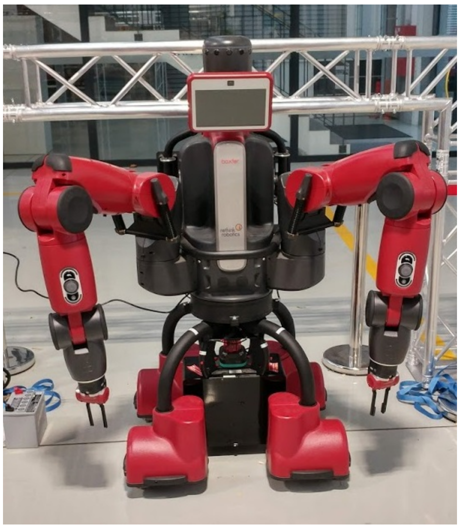


Figure 85: Baxter with Mobility Base

In this scenario, the Baxter with Mobility Base serves as a mobile grabber, so it has the skills to grab and move. Among other things, it can be used to load and unload the MiR with Toolbox.

Summary

Finally, I have summarized all robots with their abilities in table 4. If a robot has a skill, the table lists the locations where it can provide the skill.

Skill	MobiPick	YuMi	Mavic 2	Sawyer	Kuka	MiR + Toolbox	Baxter
Grab	$\{1, 2, 3, 4, 5\}$		$\{1, 2, 3, 4\}$	$\{5\}$			$\{1, 2, 3, 4, 5\}$
Move	$\{1, 2, 3, 4, 5\}$		$\{1, 2, 3, 4\}$			$\{1, 2, 3, 4, 5\}$	$\{1, 2, 3, 4, 5\}$
Store	$\{1, 2, 3, 4, 5\}$		$\{1, 2, 3, 4\}$			$\{1, 2, 3, 4, 5\}$	
Glue		$\{2\}$					
Inspect		$\{2\}$			$\{3\}$		

Table 4: Summary of the robots with their skills

7.3.3 Task Models

After having modelled objects and robots, the last step is to model the actual tasks.

It is usually worthwhile to differentiate between basic tasks and compound tasks that can be called later by the user. The reason for this is that you often define basic tasks in advance, such as picking up an object from the storage location, because you usually want to use this basic task as part of a larger user-callable task. Note that the model does not make this distinction, as it only serves to maintain a better overview; the distinction can be made in other ways on an individual case by case basis.

So in the following, I will first introduce basic tasks that I will later reuse to create tasks that can be called by the user. Again, I will only show a small section of particular interest.

Basic Task: Inspect Object for X

First off, I want to describe the task "inspect object for X", even if it already contains the subtask "move object from A to B". The main reason is that the move task is very complex, and I want to start with a simple example. This task aims to examine a pre-defined object for a particular property, e.g. whether it is free of scratches and can thus be delivered to the customer. This basic task is used, for example, when the operator gives the system the command to bring the finished wristwatch to the shipping station.

In Figure 86, the task is shown as a graph. First, the object is localized with the help of the knowledge base. Afterwards, the task stores the current position in a variable called A. Next, a check is performed whether the object is inspectable at all. A subtask is called that takes the object to any location B. The required inspection is then performed at this location. If the robot found a defect during the inspection, the task throws an error, which must be handled accordingly in the calling task. Afterwards, the object is brought back to its original location. The location B is never specified, since it is freely selectable, as long as a robot can provide the required skill, in this concrete case the inspection skill, at this location B. If A and B should be the same position, calling the task "bring object to B" will have no effect.

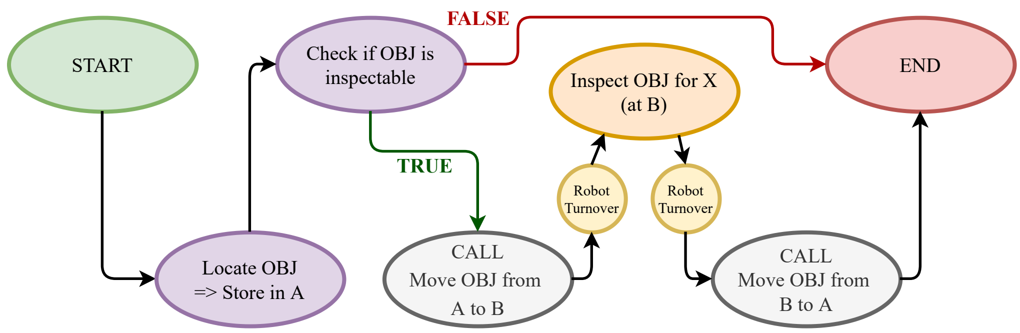


Figure 86: Model of the basic task "inspect object for X".

Basic Task: Move Object to B

One of the basic tasks, and also most complicated due to the variety of options, basic tasks in the "move objects to B" task. There are many different approaches to fulfil this task. On the one hand, the MobiPick can solve this task alone for all cases and the drone at least for a limited area. All other robots have to work collaboratively to accomplish the task. This exemplifies the advantage my system has through the functionality of calling subtasks: More complex tasks need to be modelled only once and can be reused afterwards. Intelligent optimization of these basic tasks leads to the fact that other tasks benefit from this optimization as well.

Figure 87 shows one way to model the move task. Since this graph is a bit more complex, I have broken it down into several parts which I would now like to discuss one after the other. First, the current position of the object is determined and checked to see if it is identical to the target position. Otherwise this is the case, the task can be ended directly. If this is not the case, the object must be transported. For this purpose, there is a Skill Selector which offers two possible solutions. The task can either be tackled by a single robot or several robots working cooperatively together.

The purple dashed area with the marking 1 shows the case where only one robot solves the task. Here the different robot actions are executed one after the other until the robot reaches the goal of the task.

Things get more complicated when several robots have to solve the task cooperatively. Therefore, I have divided this part into three subsections, each marked by purple dotted lines and the numbers 2 to 4. The robot turnover node between the parts gives the system the possibility to use another robot for the following commands. In the first step (box 2), it must be ensured that a robot is available at the starting point of the object that can grip it. There are again two possibilities for this, either a robot is installed stationary locally, or a mobile robot must travel to the location to perform the gripping. Two separate paths again represent these two possibilities. In the concrete scenario, the shipping area is equipped with a stationary robot that can grab objects, whereas the storage shelf has no such robot. In this case, for example, the Baxter with Mobility Base could be used to provide the grab skill. After the object has been gripped, a mobile robot is needed to transport it from A to B. This is illustrated by section 3. A perfect robot for this purpose would be, for example, the MiR with the toolbox. Finally, the object must be unloaded. This last section 4 is identical to the previously mentioned section 2, since, except for another location parameter, the same procedure is required.

This representation of the task already covers all possibilities in the concrete scenario of moving objects from A to B. However, it is not yet fully optimized. If a mobile gripper robot is required, the movement of the robot and the transport platform is sequential. But there is no reason why these two robots should not be moved in parallel. This can be achieved by using the parallel node, but this was omitted in the graphical representation for readability.

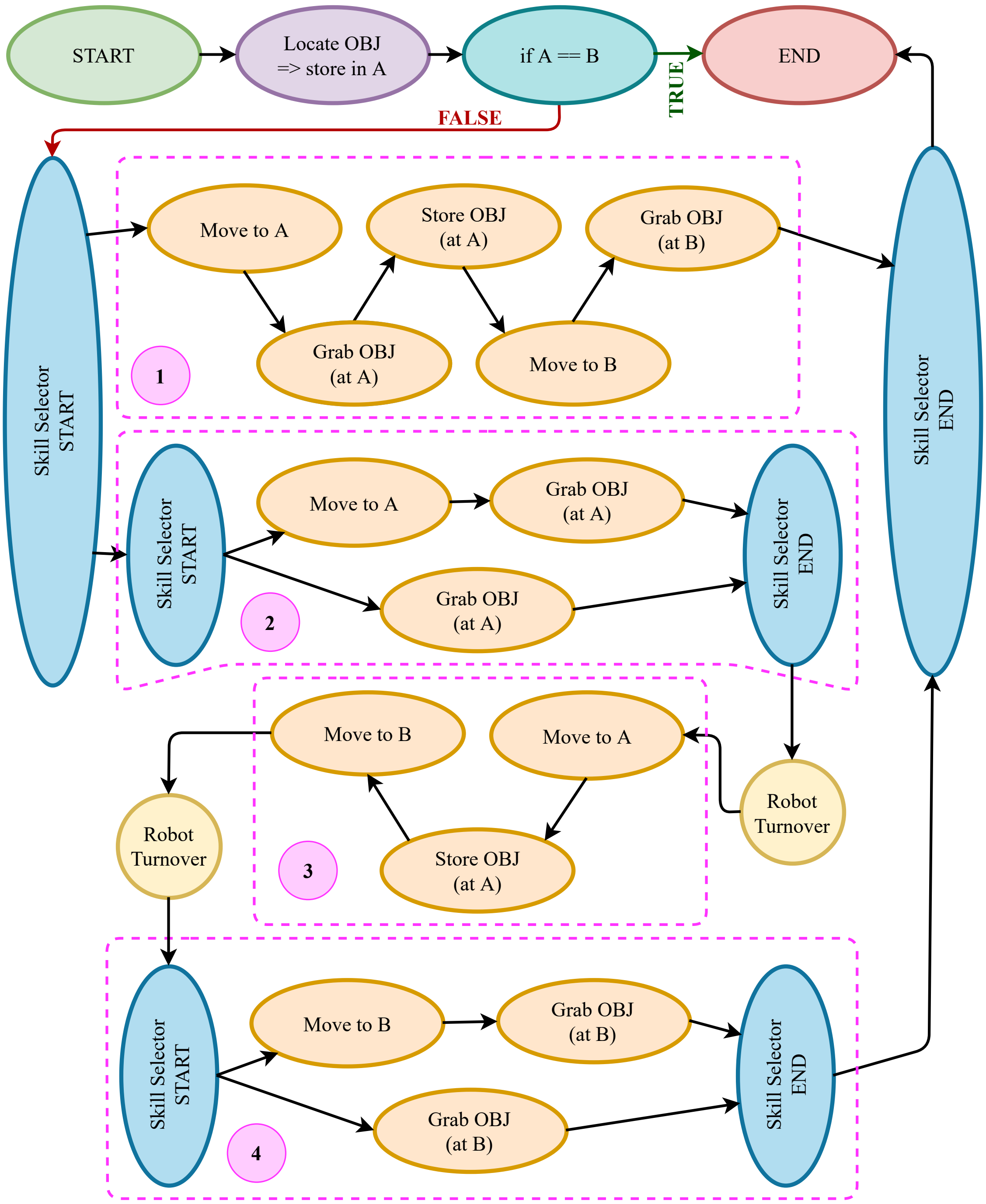


Figure 87: Model of the task "move object to B".

Callable Task: Ship Object X

This task can be used by the worker to bring the finished watch directly to the shipping area. Necessary inspections are carried out automatically. The input parameters of the task look as follows:

$$Ship(x) \text{ where } x \text{ is } ObjectEntity \wedge x \text{ is } FinishedWristwatch$$

Here, it is defined that the task has an input partner named x. As a condition this x must be a concrete instance derived from the two abstract concepts *ObjectEntity* and *FinishedWristwatch*. The last concept has not yet been introduced, but as its name suggests, it represents the concept of a finished wristwatch. Only if the input parameter fulfils these two conditions can the task be initialised at all.

These restrictions of the parameter can now also be used to generate example sets automatically. First, a list of matching entities is generated that fulfil the conditions. The placeholders then replace these entities in the annotated sentence, and the result is sent to the training engine of the NLU system. These entities can also be extended dynamically at runtime. Cerenece Mix offers an API for this and RASA does not need a complete list of all possible entities anyway. So if the worker names the objects differently than the training corpus specifies, this is no problem. In this case, the example sentences could look as follows:

- Please bring the $\underbrace{\text{finished wristwatch}}_{\text{x}}$ to the shipment desk.
- You can send the $\underbrace{\text{finished product}}_{\text{x}}$ now.
- Please send the $\underbrace{\text{assembled watch}}_{\text{x}}$ to the shipment desk.

The only missing piece left is the model of the task which is shown in Figure 88. As you can see, this task is much cleaner due to the use of subtasks. First of all, it is checked if the watch has a gold look. If this is the case, it is checked for scratches first. If the inspection fails, the worker is notified by voice. If the inspection is flawless or not necessary, the watch is taken directly to the shipping area.

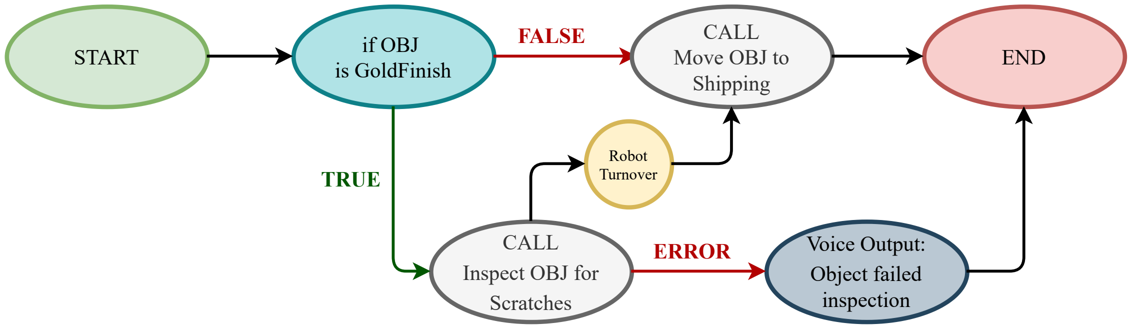


Figure 88: Model of the task "ship object X".

Callable Task: Bring me Object X

Finally, I would like to show an example of how easy it is to make basic tasks accessible to the worker. Recall that the task of moving an object to another location is one of the central and often reused tasks. Therefore, in this scenario, the task is also made accessible to the user by allowing him to have a robot handle requests to the storage location. For this, I have graphed the model of this task in Figure 89, and as you can see, it consists of only a few nodes which only serve to intercept the error of empty stock. This also shows that this distinction between basic tasks and callable tasks is fluent and can be easily overcome, provided that a few mistakes in the user input are caught.

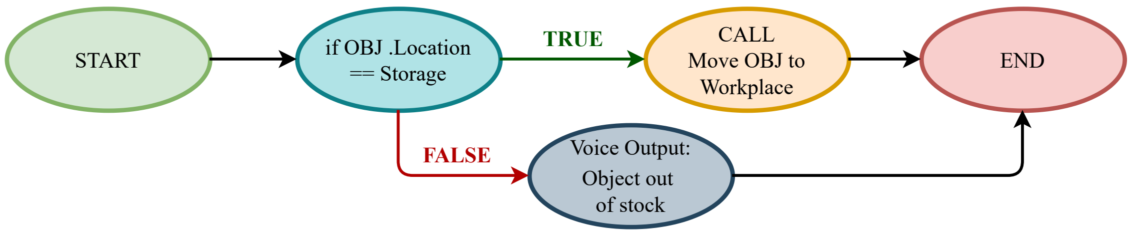


Figure 89: Model of the task "bring me object X".

7.4 Task Generation and Robot Assignment

A large part of the discussion so far has been focused on the models in the scenario, so I would now like to show a concrete generation and assignment of a task. For this, I will use as an example that the worker wants to perform an inspection on the *EdelGold B-* housing. The worker has already given a voice command that the NLU component has correctly recognized this. Besides, the transferred parameters match the definition of the task. As a result, the dialogue system would finally pass the following call to the task generation:

$$Inspect(\textit{ EdelGold B- })$$

First, the system would generate the task by loading the model from the knowledge base. After that, the system unfolds the graph by replacing all calls with the appropriate graph of the subtasks. Since the move subtask is very large, I have shortened it and shown the result in Figure 90. For a better understanding, I divided the graph into three subparts, marked by purple dashed boxes.

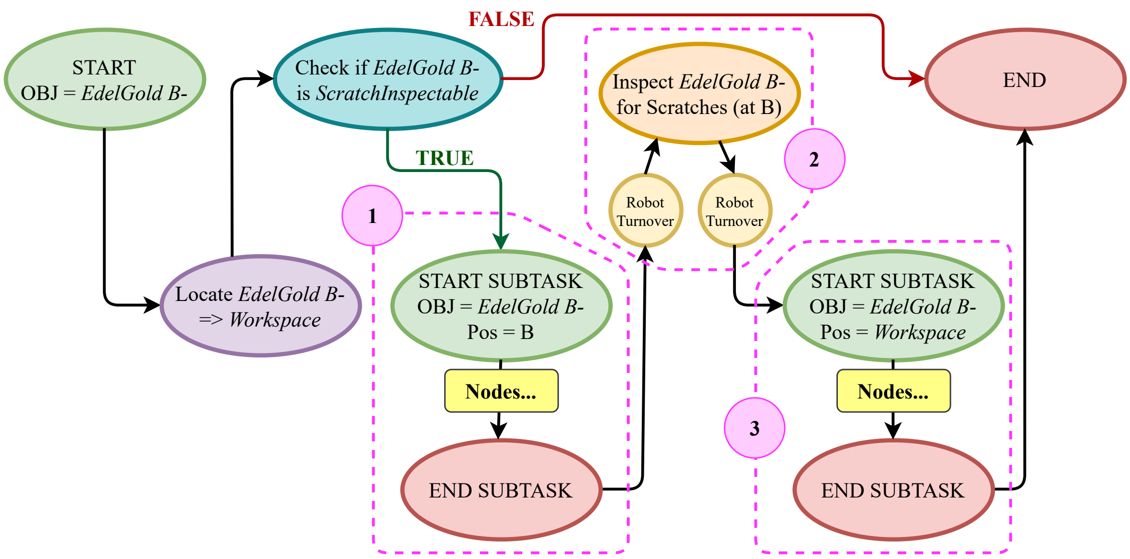


Figure 90: Generation of the task "inspect *EdelGold B-*".

Now that only one large graph is available, the system can use it for checking validity. Accordingly, every path is simulated once. This serves to check whether all parameters meet the requirements. It is crucial for calls of subtasks, because the input parameters are not fixed until now. For example, the current position of the object to be inspected, in this case, the *EdelGold B-* housing, can only currently be assigned a real value. However, the location at which the inspection is to be carried out, here called $location_B$, is still variable. This location depends on the robots performing the inspection. If at least one path has passed the test, the next step is to determine the requirements for each possible path.

For clarity, I divide the calculation of requirements into the three parts marked in the graphic. Thereby, part 1 and 3 are virtually identical. If we look at the "move object to place" task, we see that there are practically 5 different ways to perform it, which are:

1. A robot carries out the complete task
2. A robot loads at the *Workspace*.
   A robot transports the *EdelGold B-* from the *Workspace* to $location_B$.
   A robot unloads at $location_B$.
3. A robot loads at the *Workspace*.
   A robot transports the *EdelGold B-* from the *Workspace* to $location_B$.
   A robot drives to the $location_B$ and unloads.
4. A robot drives to the *Workspace* and loads.
   A robot transports the *EdelGold B-* from the *Workspace* to $location_B$.
   A robot unloads at $location_B$.
5. A robot drives to the *Workspace* and loads.
   A robot transports the *EdelGold B-* from the *Workspace* to $location_B$.
   A robot drives to $location_B$ and unloads.

These five variants can be translated into the following task requirements for the first part of the task:

$$\begin{aligned}
\text{Variant 1: } \{\ \{\ & (move, \{\textit{Workspace}, \textit{location}_B\}), \\
& (grab, \{\textit{Workspace}, \textit{location}_B\}), \\
& (store, \{\textit{Workspace}\})\ \}\ \}
\end{aligned}$$

$$\begin{aligned}
\text{Variant 2: } \{\ & \{\ (grab, \{\textit{Workspace}\})\ \}\ , \\
& \{\ (move, \{\textit{Workspace}, \textit{location}_B\}), \\
& \quad (store, \{\textit{Workspace}\})\ \}\ , \\
& \{\ (grab, \{\textit{location}_B\})\ \}\ \}
\end{aligned}$$

$$\begin{aligned}
\text{Variant 3: } \{\ & \{\ (grab, \{\textit{Workspace}\})\ \}\ , \\
& \{\ (move, \{\textit{Workspace}, \textit{location}_B\}), \\
& \quad (store, \{\textit{Workspace}\})\ \}\ , \\
& \{\ (move, \{\textit{location}_B\}), \\
& \{\ (grab, \{\textit{location}_B\})\ \}\ \}
\end{aligned}$$

$$\begin{aligned}
\text{Variant 4: } \{\ & \{\ (move, \{\textit{Workspace}\}) \\
& \quad (grab, \{\textit{Workspace}\})\ \}\ , \\
& \{\ (move, \{\textit{Workspace}, \textit{location}_B\}),
\end{aligned}$$

$$(store, \{\textit{Workspace}\})\ \},$$
$$\{\ (grab, \{location_B\})\ \}\ \}$$

$$\text{Variant 5: } \{\ \{\ (move, \{\textit{Workspace}\}) \\ (grab, \{\textit{Workspace}\})\ \},$$
$$\{\ (move, \{\textit{Workspace}, location_B\}), \\ (store, \{\textit{Workspace}\})\ \},$$
$$\{\ (move, \{location_B\}), \\ \{\ (grab, \{location_B\})\ \}\ \}$$

As you can see, such requirements can quickly become very complicated. On the other hand, the task requirements for section 2 are straightforward, since only one robot action is performed that looks like this:

$$\text{Part 2: } \{\ \{\ (inspect, \{location_B\})\ \}\ \}$$

As for part 3 in the graph, the result is very similar to the first part, only the locations for the store action is different. Thus I will not write them down.

Since a robot turnover node separates the three parts, they can be combined as desired, because each piece can (but does not have to) be solved by a different robot. This results in an extensive list of different task requirements which all represent a valid solution for the task. Here, I will only present one task requirement, where both transport are handled by only one robot, out of $4 * 1 * 4 = 16$ possible ones.

$$\text{Task Requirements 1: } \{\ \{\ (move, \{\textit{Workspace}, location_B\}), \\ (grab, \{\textit{Workspace}, location_B\}), \\ (store, \{\textit{Workspace}\})\ \},$$
$$\{\ (inspect, \{location_B\})\ \},$$
$$\{\ (move, \{\textit{Workspace}\ , location_B\}), \\ (grab, \{\textit{Workspace}\ , location_B\}), \\ (store, \{location_B\})\ \}\ \}$$

Afterwards, suitable robots or collections of robots must be found that can meet these requirements. For this aspect, it is helpful to take a look at table 4 where I have summarized all skills of the different robots. Since the variable $location_B$ can still take any value, it is now restricted by the skills of the available robots and set to a value. In the concrete case, only the locations 1 and 2 are possible, because only there a robot with the inspection skill is available. In the following I list all possible solutions for the $Task\ Requirements$ 1:

$$\text{Robot Collection Candidates for Task Requirements 1: } \begin{aligned}[t] &\{\{MobiPick, YuMi\},\\ &\{MobiPick, Sawyer\},\\ &\{Mavic, Sawyer\},\\ &\{Mavic, YuMi\},\\ &\{MobiPick, Mavic, Sawyer\},\\ &\{MobiPick, Mavic, YuMi\}\}\end{aligned}$$

If the system now assigns the task to a concrete robot, the remaining variables are also resolved, such as the $location_B$ where the inspection is performed. Therefore, the system now assigns values to the parameters and attaches robots to all robot actions. However, before the system can mark an assignment as valid, it must be checked whether each robot action is possible. For example, the grab skill does not necessarily mean that any object can be grabbed. In our case, for example, the check to see if the Mavic 2 drone can store the housing would fail because it is too heavy. Thus this assignment is discarded.

Finally, the system now has several valid robot collections that can solve the task. For each collection, the heuristic costs are calculated, which this solution causes. Factors for this include how long the processing takes and how valuable the robot is for the entire production. The system selects the most cost-effective solution, and then the task is assigned to the robots, unnecessary nodes like skill selectors and robot turnovers are removed, and the resulting execution graph is added to the central task queue. The processing of the task now begins.

7.5 Possible Applications

The models I have shown here allow many different applications, all of which can be triggered most times via one sentence. This allows a worker to quickly delegate a task to a robot, e.g. if he has to leave the workplace for a while or he has to do another work that cannot be done by a robot. Besides, he can order required work parts in advance by voice which are delivered by robots. Processing steps that can only be carried out by robots, e.g. due to the required force or precision, can also be very quickly commanded by voice. Finally, this allows the worker to intuitively and efficiently cooperate with the robots to achieve the goal of a high-quality wristwatch begin manufactured according to personal preferences of the customer.

As an example, the production process for a complete wristwatch could look as follows. The customer's specification for his watch looks like this:

- Housing: EdelGold B-
- Clockwork: Precise Clock A3
- Cover Glass: Crystalclear 45mm
- Band: Brown Leather

screwed (Housing, Clockwork)

glued (Clockwork, Cover Glass)

This is what the worker's dialogue with the system could look like, as well as his working steps:

1. "Please bring me the EdelGold B-"
   → *Bring me Object EdelGold B-*
2. "I need a Precise Clock A3"
   → *Bring me Precise Clock A3*
3. "Hand me the appropriate screws"
   → *Using the KB and discourse memory, correct screws are delivered*
4. The worker screws the clockwork to the housing by hand
5. "Glue a Crystalclear 45mm cover glass to my current workpiece."
   → *Glue Crystalclear 45mm to current workpiece*
6. "Supply me with a set of brown leather band."
   → *Bring me Object brown leather band*
7. The worker attaches the leather band to the clock
8. "Send the finished watch to the customer."
   → *Ship watch, including inspection before shipping*

In the end, the worker was able to build the clock in cooperation with the system by using simple voice commands. It took over the precise and powerful work like glueing and checked the final product before sending it to the customer. This scenario described in the previous sections illustrates that my model can be successfully be used for a collaboration of robot and human on even more complex tasks.

Chapter 8
Explorative Evaluation of the NLU System

I conducted an explorative user study on the natural language system, which is the focus of this chapter. The objectives were to find out how well people handle the system without getting a lot of training beforehand, whether such a system can be used intuitively, and whether is effective in performing the required tasks. For this purpose, a virtual workplace was created in which the participants had to solve tasks. Besides, the cognitive load was recorded using the NASA TLX. Finally, a questionnaire was used to record the participants' experiences with the system. In the end, all data were evaluated, and the results are summarized in the following. Furthermore, the data could be used to improve the natural language model further.

8.1 Scenario / Implementation

To evaluate the speech system, a virtual industrial hall was created as a scenario. The participant is a worker at this workspace, who has several robots that help them perform the work. The goal is to create a workpiece consisting of different coloured balls and rectangles that can be glued or screwed together. However, only black objects are available in the warehouse, so the worker must use the robots to change colours. Also, the objects can only be connected using robots. In the end, the object must be sent to the customer by a robot. The graphical representation of the workspace is given in Figure 91.

All virtual robots are connected to the system described in this thesis and can be controlled by voice. The system then detects the intent, plans the task, and automatically distributes the necessary tasks to the robots in order to achieve the goal. To supply the system with speech, the participant has a small box that switches the microphone on and off. To avoid interference, automatic voice activation has been omitted. Also, the box can be used to provide feedback on whether the action performed was the desired one or not (see Figure 92).

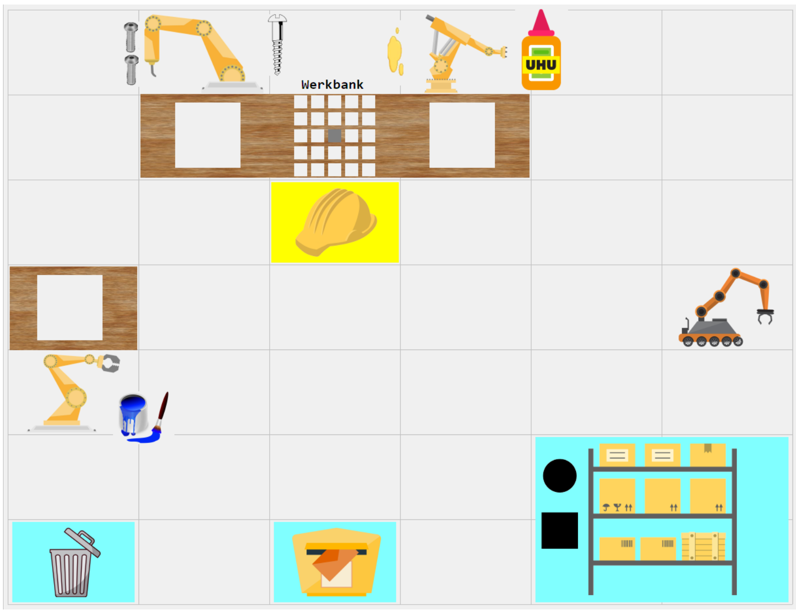


Figure 91: Visual representation of the workspace used for the study

Figure 92: Box for opening/closing microphone and giving feedback

In addition to the simulated workplace, the participant also has a detailed display of the workpiece, a live transcript of the dialogue, status fields for the microphone, and an overview of all possible objects that he can create the robots with (see Figure 93). This is to help the user to orientate himself better in the virtual environment. During the study, all voice data were recorded in order to be able to trace errors of the voice recognizer later. Also, all actions and intermediate states of the system were logged for later evaluation.

Cerence Mix, a web service for speech processing, was used for speech recognition and intent detection. The deep integration between the two systems leads to the best results in free text recognition. Previously, a combination of the Microsoft speech recognizer and RASA was tested. However, this turned out to be problematic because the speech recognizer had no information about the speech model and therefore gave significantly

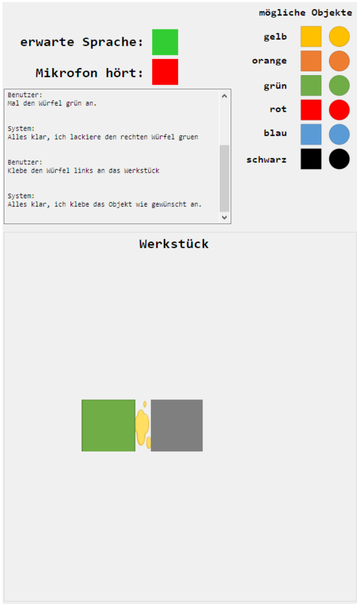


Figure 93: Info display showing the current state of the system

worse results, so this approach was discarded. All data from Nuance, amongst others the recognized speech as well as the analysis by the NLU system, were stored for later evaluation. The training data for intention recognition was generated from the knowledge database for the tasks and uploaded to Mix via the web interface.

As a feedback channel, the system had, on the one hand, the buttons that lit up after action was completed. This signalled to the participants that they cloud now give feedback (positive or negative). However, most of the feedback was provided via voice output. The system was able to confirm recognized commands or, in case of incomplete information, to request the parameters still needed. The speech output was generated locally via the Microsoft Speech Platform.

8.2 Process

One thing that has been taken into account right from the start is that people talk differently with computers than to other people because they know that computers are limited in their ability to understand speech (compare Jurafsky[39]). Since the goal is to build a natural language interface, people were animated to communicate with the computer as naturally as possible to eliminate this effect.

The study was organized in several parts. First, the candidates were welcomed and informed about data protection. An explanation of the scenario was then read out loud so that they could understand the virtual workspace (see Appendix A.1). Afterwards, a soundless video was shown to show the participants that the robots can move and manipulate objects. This should give them a rough idea of what is possible. They did not get any information about what they had to say to trigger the actions, though.

Since I want to capture the natural language of the candidate, it was essential not to put words in their mouth. Therefore, the explanation was kept general, and many things were not mentioned by name, so that they would later be as unbiased as possible in naming the objects. In particular, the possible objects were only shown as pictures, but not named. Only the possible colours were mentioned by name. To exclude technical problems with the microphone, the candidate should trigger a predefined test command by voice (which has nothing to do with the actual study).

After this introduction, the participants should have gotten a rough idea of what they can do in the factory automated with robots; only did they not yet know how to control the robots by voice. Therefore, a free test phase was started, where they could try out commands. The goal of the stage was to get familiar with the system and to perform all the actions they have seen in the video (trigger every possible skill). After the participants had triggered every skill at least once and feel comfortable with the system, the cognitive load is measured using the NASA TLX [40] test and afterwards they could move on to the next part of the study.

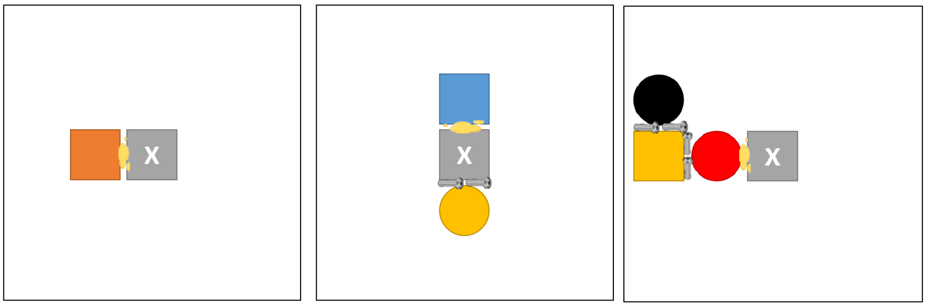


Figure 94: Workpieces that should be builded in the study

The participants then got three tasks in the form of construction plans, which they had to rebuild and then send to the customer (see Figure 94). For this, they had to use the robots, which they could control by voice. After this task was completed, the cognitive load was measured again. Finally, there was a questionnaire as feedback (see Appendix A.2).

8.3 Evaluation

First of all, I would like to explain the representation of the results. I will represent most of the results as box plots. I will use the convention shown in Figure 95 to illustrate the median and other relevant statistical data. All the other results will be presented as pie charts. I have added further detailed information in the appendix.

For the evaluation, I introduce the following new metric: the success rate. This rate was calculated from the data obtained during the study. It indicates how often a request was successfully executed in relation to unsuccessful queries. Reasons for unsuccessful requests can be that, e.g., the speech recognizer understands wrong words or that the

NLU system returns a wrong intent. Another reason could be that the user requests a task that is not possible. Also, the feedback from the red and green buttons was used to calculate the metrics. The success rate should provide a good indication of how well the voice component of the system is performing. Since users do not receive any specific training, it can also indicate how user-friendly and beginner-friendly the chosen approach is. Since the system should perform well on these points, the first hypothesis will be: **The success rate increases rapidly and levels off at a high level**.

Another goal was that the system should not be demanding to use. Therefore I measured the cognitive load via the NASA TLX to be able to check the following hypothesis in the evaluation: **Using the system after free training is not very demanding.** Maybe the first NASA TLX value will be a little higher after the free practice phase because, for the participants, the system is something completely new. But after the predefined construction tasks are completed, I expect that the cognitive load will not be very high, as assumed in the hypothesis.

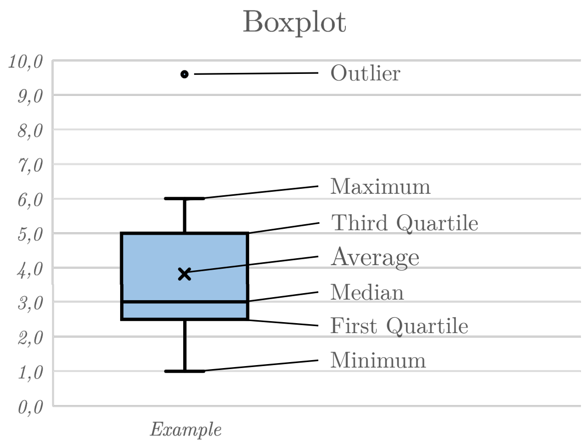


Figure 95: Convention for presenting the results in a box plot

A total of 15 persons participated in the study for evaluation. Their distribution is given in Figure 96. The gender was nearly equally distributed (46% female), and they were aged between 18 and 27. In Figure 97, the test persons are broken down according to their previous experience with speech systems and robots, scaling from 1 to 5 and no experience at all. Only about a quarter had no experience with speech systems, so most participants already used some speech systems like Siri or Alexa. About three quarters stated that they had no experience with robots.

To test the first hypothesis, I calculated the success rate for the free test phase and each of the construction tasks. The results are shown in Figure 98. As you can see, the rates increase over time. This increase is also significant to all three tasks. This has shown that the free practice phase is sufficient to become familiar with the system and to work successfully. In concrete terms, the values increase from about 65% to up to 90% in the free practice phase. Some test persons even achieved a 100% success rate. This is a clear indicator that the chosen language model meets the requirements.

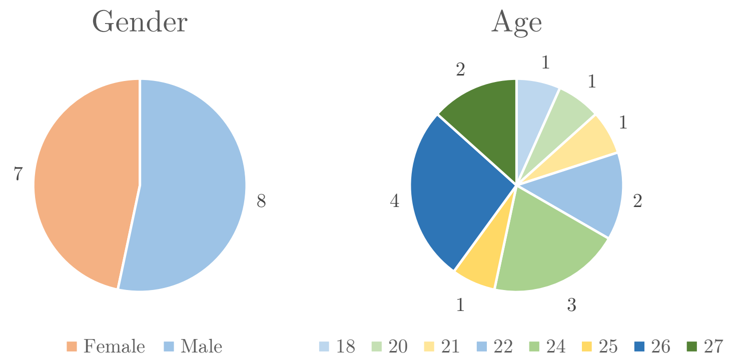


Figure 96: Distribution of gender and age

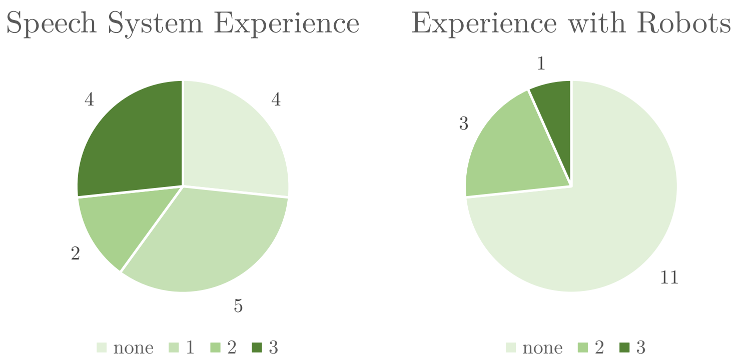


Figure 97: Distribution of experience (scale from 1 to 5)

Afterwards, I calculated the NASA TLX value, using the unweighted version. The results of the two scores (after the free exploratory phase and at the end of the study) are shown in Figure 99. The scale of the score ranges from 0 to 100, with 100 being the most demanding. The cognitive load decreased significantly after the first free exploitative phase. However, one participant (number 7) stung slightly upwards and was removed from the sample without affecting the significance. The average load dropped from 7 to 5. This value is very low and also confirms my hypothesis that the system is not exhausting to use.

In the next step, I evaluated the questionnaire in detail, which you can see for the first four questions in Figure 100. First, we asked the participants whether they found the system intuitive, which most of them answered very positively. This matches the insights gained from the success rates. Then we asked whether they had the feeling that they could reach their goal with the system and whether the system understood them well. Here the results were more in the middle range. One reason for this could be the training phase, in which I provoke misunderstandings so that the users can learn what is possible and what is not. But most participants found it easy to very easy to formulate sentences for the system. The main reason for this is probably that the users can use natural language and were motivated to talk to the system like a human being so that they did not have to change their normal behaviour much.

In Figure 101 the last 4 questions were analysed. We have evaluated the feedback of the

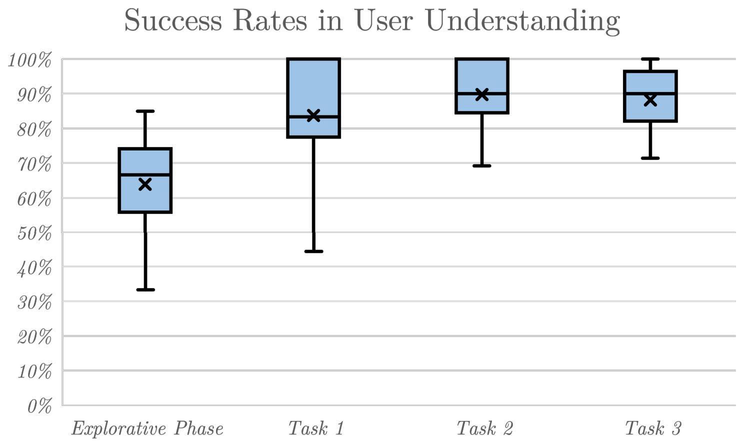


Figure 98: Success Rates significantly improves after explorative phase

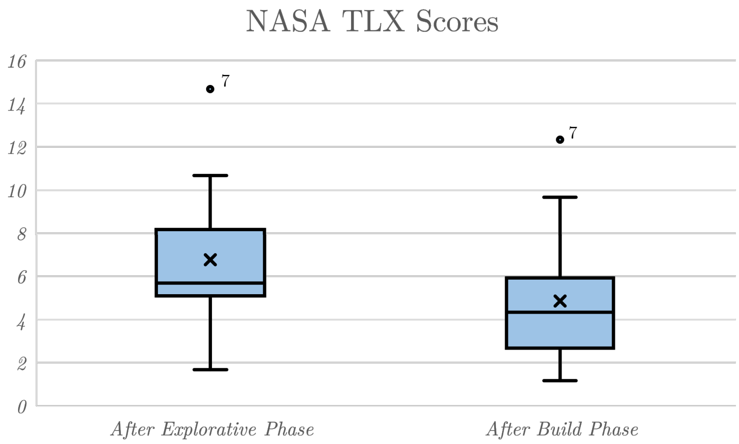


Figure 99: NASA TLX Score decreases significantly after the explorative phase

system, which, for example, asks if not all information for a task has been communicated or understood. Here the test persons found the feedback from the system consistently good to very good. This might explain why the training phase is so successful, and then the success rates increase strongly afterwards, as feedback plays an important role in the training. Without good feedback, the candidate cannot adjust to the system. However, the perceived misunderstanding ranks only in midfield. Perhaps this may be due to the free exploratory phase, in which misunderstandings are more or less provoked. Through misunderstandings, the user learns what the system can and cannot do. Besides, misunderstandings later allow me to improve the system. Most users were motivated to discover many actions in the system and had no problems positioning the objects. So most of the participants had fun exploring and learning the system. Even the quite simple positioning descriptions were not an obstacle for most of the participants to move freely.

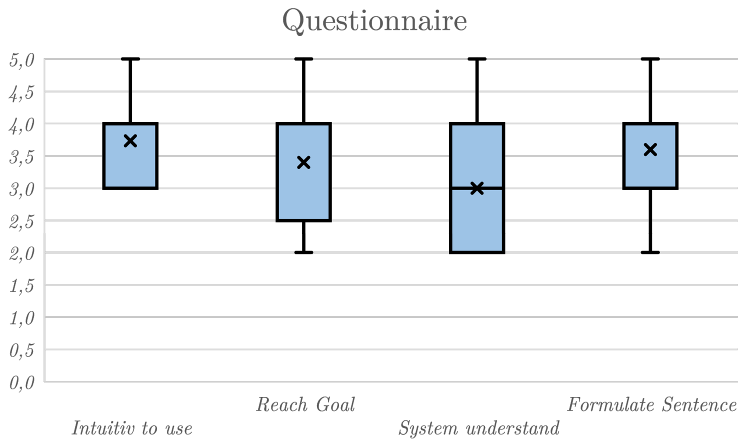


Figure 100: Result of the first 4 questions...

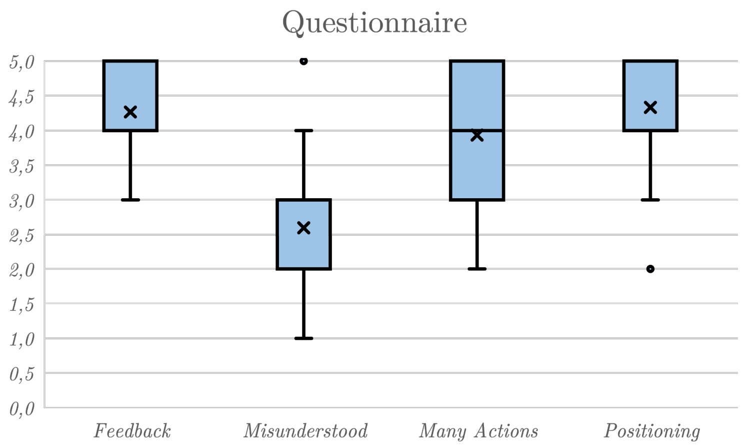


Figure 101: ...and the last 4 questions of the Questionnaire

The questionnaire thus largely confirms the hypotheses made. One can see that the users received the study well and that they were keen to participate actively.

In conclusion, I would say that the study has confirmed both hypotheses. The system is a very intuitive way for the majority of the participants to control robots in the industrial field without much training. All test persons found suitable commands for the desired tasks without instruction and could build the required three workpieces without problems. None of the test persons had the feeling that the system did not understand them at all, or that they did not reach their goal.

Last but not least, the collected data was used to optimize the language model further. Correctly formulated sentences that could not be assigned to an intent were included in the language model. This improves the recognition performance of the speech system.

Chapter 9
Conclusion

In the last chapter, I summarize my findings regarding the proposed research questions and present a conclusion. Afterwards, I provide an outlook on possible enhancements and extensions for my system as well as some ideas for further research work.

9.1 Summary

The goal of this thesis was to develop concepts and design a system that can execute user-provided tasks with the help of a task model and a matching language model. Most importantly, the user can intuitively communicate tasks to the system via natural language. By utilizing the skill model, tasks are assigned and executed by suitable robots. Several steps were necessary to achieve this goal, which I will sum up in the following.

First of all, I investigated and compared existing research on the three areas that are of major importance for my work: knowledge representation, user interaction and task management. The insights gained from this have significantly influenced the final system. Especially the representation of the tasks and the language model are inspired by already existing work.

After I finished my research on other papers, I first designed a rough architecture which represents the later interplay of the different components. It is important to note that the knowledge base is later of essential relevance for all activities since all parts of the systems make their decision based on it.

Therefore, I paid close attention to the design of the knowledge model. First, I explained the underlying representation and storage, as well as the TBox/ABox concept. Afterwards, I presented the theoretical basis for my robot skills. I have defined the different levels of robot skills and task requirements and specified when one or more robots satisfy the requirements for a task. This definition takes into account not only the skill of the robot itself but also its location in a factory. This additional parameter allows for scenarios in larger environments, such as an industrial hall. I designed the model in such a way that several different robots can co-operate on a single task.

After the theoretical introduction to the robot skills and task requirements, I introduce a detailed modelling of robots and objects in the knowledge base. Through inheritance of abstract concepts, skills can be assigned to concrete robots, and specific objects can be given selected properties. Tasks are defined inside the knowledge base and concepts model which kinds of interactions with objects are possible. For illustration purposes, the system can draw a directed graph for each task. A task consists of individual nodes; each node can have several successors and predecessors. There are different types of nodes available for the tasks that trigger various actions, such as an interaction with the user, a mathematical operation or a query of the knowledge base. But the most relevant nodes are the robot actions. These trigger concrete actions that a robot will perform later if the user lets the system execute the task. The robot action nodes also determine what skills are required for a task to be performed by a robot.

Now that several various robots can solve the same task using different skills, concrete tasks inevitably lead to different possible paths of execution. There are particular nodes in the graph which indicate that the system can choose any execution alternative, but only one and that the final result will be the same. This opens up several possibilities for the later planner to execute a task; the planner can decompose the task into subtasks in many variations and choose the best. In the next step, the task must now be initiated by voice command.

To provide the best possible user experience, I decided to create an NLU system. As port of this, a training corpus is needed to train the system. For this purpose, annotated example sentences are stored in the knowledge base for each task, from which the system can generate the required training data. Due to difficulties with the combination of speech recognition and RASA, I opted for the integrated solution from Cerence Mix, which leads to a more delightful experience. As a result, the system can now understand freely formulated sentences and map them to concrete tasks, including parameters. Furthermore, there is a dialogue system that can ask back questions in case of unclarities, for example, if a parameter was not recognized correctly. After the dialogue has gathered all the necessary parameters and these have been checked for validity, the task is initialized.

After initiating the task, the system calculates the task requirements for each possible execution alternative. With the help of these requirements, a planner can decide if individual robots or combinations of several robots satisfy these requirements. The result will be a list of robots capable of solving the task. To find out which solution is the most appropriate, an abstract cost metric is introduced for each combination. Several constraints are taken into account, e.g. how many skills a robot has, whether it is currently being used or whether it is often needed for other jobs. The planner then decides for an assignment and populates the task with the corresponding robots. Afterwards, the task is placed into a central queue, which processes the individual nodes of the task one by one until the task is completed.

My system implements solutions to the research questions theoretically and practically. To validate the results, I modelled a concrete scenario, in this case, the MRK-4.0 laboratory, and tested possible actions. All common cases could be sharply covered in the knowledge base, and the system assigned the tasks to the robots in a reasonable way. Another important goal was to make interaction with the system as easy as possible; therefore, I used an NLU system. I have done an explorative user study on the user interaction component and evaluated the results, which revealed that the system was understood by the majority of the users intuitively.

9.2 Future Work

Even though the system already provides many functionalities, there are still open fields for improvement and further research.

First of all, it is generally worthwhile to improve user interaction. One way to enhance user interaction would be the use of multimodality. Often the objects the user wants to interact with are right in front of him, but he still has to name them explicitly. If, for example, the direction of gaze or pointing gestures were implemented as a modality, the user could point to objects to reference them. This additional modality would make the system more intuitive, and probably lead to shorter sentences, making the system more efficient to use. Besides, new findings could be made on how multimodality affects the industrial environment in interaction with robots.

Currently, there is a strong NLU component and subsequent dialogue component, which can take over all functions to start a task. However, during the execution of a task, only a limited interaction with the user is possible. For example, the system cannot actively request further information regarding a task during its execution but must do so before initializing the task. Therefore, an extension of the dialogue functionality would make sense, possibly the dialogue could also be outsourced to an external dialogue component, such as SiAM-dp or step-dp. Such an extension would make it possible to obtain information from the user even during execution, which could simplify the initialization of tasks, especially for a complex task with many parameters.

Another exciting direction future research could take is the field of teach-in of tasks. It would be interesting to take this system as a basis and to extend it with a rapid teach-in function. Thereby the user would be allowed to define their tasks, for example, by voice, which can then be executed later. The advantage of a teach-in is that this can quickly be done by the user directly on-site, without the need for specially trained technicians to program the task. As a result, new processes can be learned on-site almost in real-time, e.g. if the requirements for a product change. The NLU system could prove to be very useful for this purpose. Moreover, the abstract task structure, which is not bound to concrete robots but skills, would provide a reasonable basis for this. As a result, the possibilities and applications of the system will significantly expand.

The task model offers a further possibility of extension. Currently, the skills and functionality represented by the different nodes are adapted to the needs for my application. However, in another scenario, one could imagine needing other types of nodes, e.g. those that provide more powerful communication capabilities. You could also start some tasks automatically when a specific event occurs. If, for example, a particular product is ordered in the online store, a task is triggered, which automatically brings the required objects to a free worker and shows him the order on display. This would make it possible to automate some processes even further by using my system.

Finally, my system can be combined with other external components. A considerable amount of research has already been done in the area of transfer of control from machine to human. If an error occurs, e.g. because the route of a robot is blocked and it cannot find a way out on its own, an external component could decide how to solve this problem. As an example, a drone could explore the environment on its own and delegate control to a human in case an obstacle is too close, who could, for instance, reduce the safety radius temporarily. It is also possible that remote control is not enough and that a person will have to remove the obstacle manually. In any case, such an extension would make the system much more robust in safety-critical situations or in the event of a failure.

Appendix A
Study

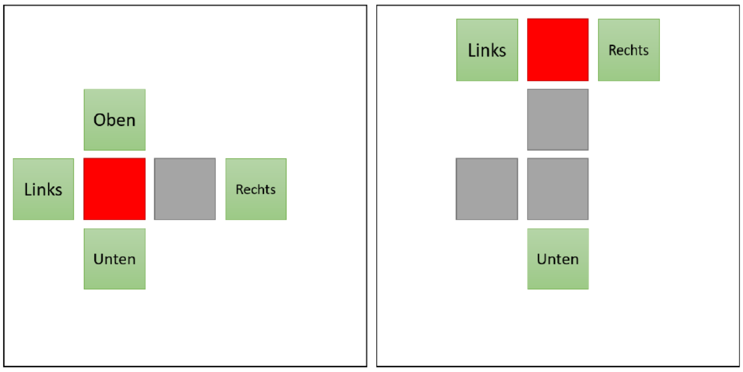
A.1 Study Introduction Text

The following text was read out in German as an introduction to the study. The pictures were shown to the participant.

> *Hallo und schon mal vielen Dank für die Teilnahme an meiner Studie. In meiner Studie geht es darum, herauszufinden wie Menschen Aufgaben an Roboter mithilfe natürlicher Sprache delegieren, um kooperativ ein Ziel zu erreichen.*
>
> *Für mein Experiment habe ich folgendes virtuelles Szenario vorbereitet: Du befindest dich in einer Industriehalle, in der etwas hergestellt wird. Dazu stehen dir verschiedene Assistenz-Roboter zur Verfügung stehen. Diese Roboter unterstützen dich bei deiner Arbeit und führen Aufgaben selbstständig durch. Dazu formulierst du mittels natürlicher Sprache, was das System für dich erledigen soll, und die Aufgabe wird passend aufgeteilt an die benötigten Roboter weitergeleitet. Wichtig ist, dass eine Aufgabe immer nur eine „Bearbeitungsschritt" sein sollte, du also nicht mehrere Aufgaben auf einmal an das System übergibst. Ist die Aufgabe so nicht möglich oder es fehlen gewisse Informationen, fragt das System nach. Nach jeder Aufgabe kannst du positives oder negatives Feedback geben, ob das System deine gewünschte Aufgabe erfolgreich ausgeführt hat oder nicht.*
>
> *Dein Ziel in der Industriehalle ist es, ein Werkstück zu kreieren. Dafür stehen dir folgende Objekte zur Verfügung.*
>
> *Diese Objekte kannst du mithilfe der Roboter verändern und kombinieren, dafür hast du rechts und links neben deinem Werkstück auf der Werkbank Platz, um mit dem Objekten arbeiten zu können. Natürlich kannst du letztlich deine Objekte mit dem Werkstück verbinden, um etwas Neues zu erschaffen. Das Werkstück besteht aus*

einem grauen Anfangsstück in der Mitte, an das von vier Seiten Objekte angefügt werden können. Dazu stehen dir die beiden oberen Robotern zur Verfügung, die über Sprache angesteuert werden können. Da deine Position fix ist, brauchst du die Hilfe der Roboter.

Die Positionsangabe kann über „links, rechts, oben und unten" definiert werden. Die Positionierung des neuen Objektes bezieht sich immer auf das zuletzt hinzugefügt Objekt am Werkstück. Bereits montierte Objekte werden dabei übersprungen. Dein Werkstück wächst also ähnlich wie die Schlange im Spiel Snake. Hier ist ein Beispiel. Das rote Objekt ist das zuletzt hinzugefgüte Objekt zum Werkstück.



Auf dem Display rechts unten siehst du eine vergrößerte Ansicht deines aktuellen Werkstückes. Oben rechts im Bild siehst du alle möglichen Objekte, die du mithilfe der Roboter herstellen kannst. Um ein Gefühl dafür zu bekommen, was möglich ist, sieh dir nun bitte folgendes Video an.

An dieser Stelle wird nun ein tonloses Video abgespielt, um zu sehen "was" in der Fabrik möglich ist.

Um mit dem System zu kommunizieren drücke einmal den Knopf, danach hörst du ein Geräusch. Sobald das Geräusch ertönt, hört das System dir zu. Wenn du fertig gesprochen hast, drücke wieder den Knopf. Danach ertönt erneut ein Geräusch und deine Eingabe wird verarbeitet. Wenn der Knopf blinkt erwartet das System eine Eingabe von dir, falls nicht wird deine Eingabe gerade verarbeitet. Zusätzlich wird dir der Status oben rechts am Display angezeigt. Zusätzlich kannst du nach jedem Verarbeitungsschritt dem System Feedback geben, ob du mit dem Resultat zufrieden warst oder nicht. Dies kannst du durch den roten und grünen Button geben. Sobald diese Buttons leuchten, hast du die Möglichkeit Feedback zu geben.

Du hast nun die Möglichkeit, die Aktion kurz zu testen. Sage dazu einfach „Hallo

Roboter".

Deine erste Aufgabe ist folgende: erstelle ein Werkstück nach deinem Belieben, du kannst zum Beispiel ein Buchstabe formen, ein Muster kreieren oder was auch immer dir in den Sinn kommt. Probiere das System ruhig aus. Alle Befehle dazu gibst du dem System per Sprache. Wenn du mit deinem Werkstück zufrieden bist, versende es einfach mit der Post. Du kannst gerne auch mehrere verschiedene Werkstücke anfertigen, wenn du noch Zeit dafür hast.

Dir stehen nun 10-15 Minuten zur Verfügung, um eines oder mehrere Werkstücke zu kreieren und das System zu erkunden. Anschließend würde ich dich bitten noch ein kleiner Fragebogen mit 6 Fragen zu beantworten.

Der Nutzer interagiert nun das erste Mal mit dem System

Nachdem du nun etwas mit dem System vertraut bist, ist deine nächste Aufgabe mehrere vorgegebenen Werkstücke nachzubauen. Dazu zeige ich dir gleich ein Bild von einem fertigen Werkstück und du sollst dieses exakt so nachbauen. Wenn du das Werkstück erfolgreich gebaut hast, versende es. Wenn du alle Werkstücke erfolgreich gebaut hast, steht dir noch ein kleiner Fragebogen bevor und danach ist die Studie auch schon beendet.

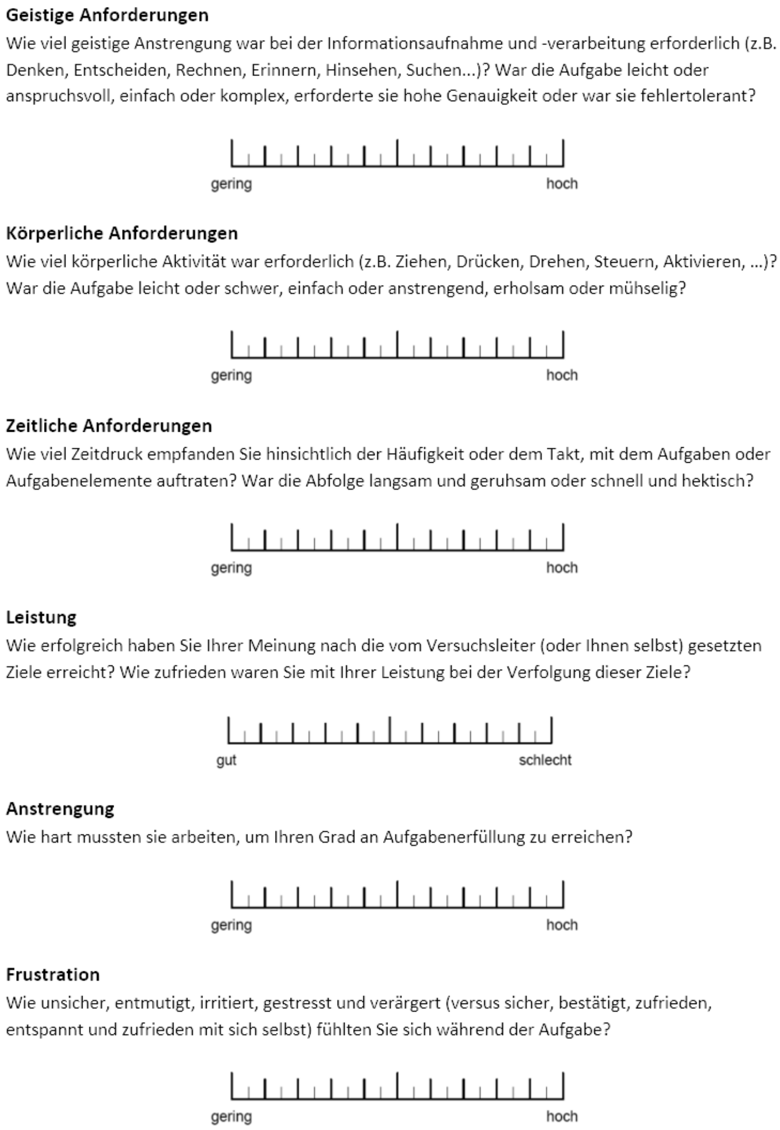
A.2 Questionnaire

The following questions were asked in German. The participant could give a score of 1-5 for the questions and an additional "none" for the first 2.

- Vorerfahrung
  - Hast du bereits Erfahrung mit Sprachsystemen oder Dialogsystemen?
  - Hast du bereits Erfahrung im Roboterumfeld?
- Studie
  - Das System war intuitiv nutzbar.
  - Ich habe mein Ziel schnell erreichen können.
  - Das System wusste stets, was ich von ihm wollte.
  - Wie einfach war es für dich, Aufgaben zu formulieren?
  - War das Feedback für dich verständlich?
  - Wie oft hattest du das Gefühl, dass deine Aufgabe nicht korrekt umgesetzt wurde?
  - Wie motiviert warst du, möglichst unterschiedliche Aktionen auszuprobieren?
  - Wie schwer fandst du die Positionierung der Objekte?
- Freitext Felder
  - Haben dir weitere Funktionen gefehlt?

– Sonstige Anmerkungen

In addition, the NASA TLX score was determined by asking the predefined questions that the participant could answer on a scale of 1-20.



Geistige Anforderungen
Wie viel geistige Anstrengung war bei der Informationsaufnahme und -verarbeitung erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen...)? War die Aufgabe leicht oder anspruchsvoll, einfach oder komplex, erforderte sie hohe Genauigkeit oder war sie fehlertolerant?

gering hoch

Körperliche Anforderungen
Wie viel körperliche Aktivität war erforderlich (z.B. Ziehen, Drücken, Drehen, Steuern, Aktivieren, ...)? War die Aufgabe leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?

gering hoch

Zeitliche Anforderungen
Wie viel Zeitdruck empfanden Sie hinsichtlich der Häufigkeit oder dem Takt, mit dem Aufgaben oder Aufgabenelemente auftraten? War die Abfolge langsam und geruhsam oder schnell und hektisch?

gering hoch

Leistung
Wie erfolgreich haben Sie Ihrer Meinung nach die vom Versuchsleiter (oder Ihnen selbst) gesetzten Ziele erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?

gut schlecht

Anstrengung
Wie hart mussten sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?

gering hoch

Frustration
Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Aufgabe?

gering hoch

Figure 102: NASA TLX Questionnaire

Appendix B
Screenshots of the Software

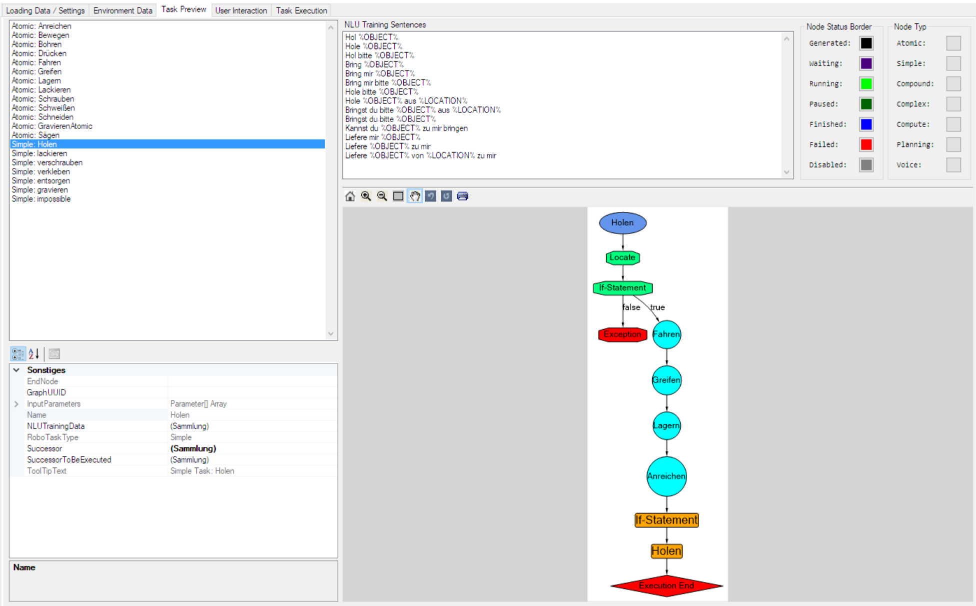


Figure 103: Screenshot of the stored model of a task and its graphical representation in my software

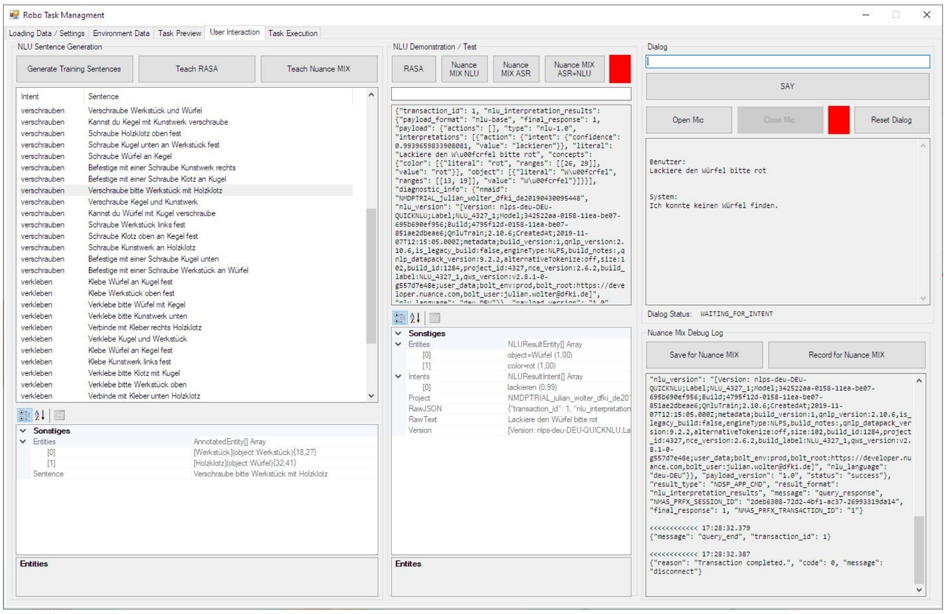


Figure 104: Screenshot of the dialogue manager with attached NLU results from Cerence Mix.

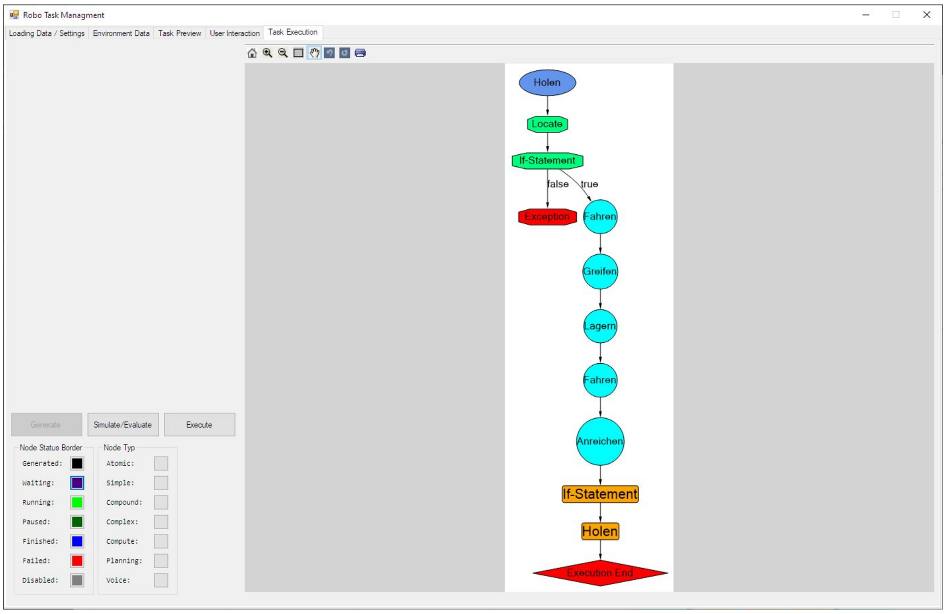


Figure 105: Screenshot of the generated task that can now be executed.

Lists

List of Figures

List of Tables

List of Acronyms

HySociaTea	Hybrid Social Teams for Long-Term Collaboration in Cyber-physical Environments
IDL	Interface Definition Language
JSON	JavaScript Object Notation
MA-PDDL	Multi Agent PDDL
MQTT	Message Queuing Telemetry Transport
MRTA	Multi-Robot Task Allocation
NASA TLX	NASA Task Load Index
NLU	Natural Language Understanding
OWL	Web Ontology Language
PDDL	Planning Domain Definition Language
Prolog	Programming with Logic (*programmation en logique*)
ROS	Robot Operating System
SRDL	Semantic Robot Description Language
TECS	Thrift Eventbased Communication System
URDF	Unified Robot Description Format
Xacro	XML Macros
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language (orig. Yet Another Markup Language)

List of Definitions

Bibliography

[1] DENAVIT, J. AND HARTENBERG, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics 22* (June 1955), pp. 215–221.

[2] URDF. `http://wiki.ros.org/urdf`. Accessed: 15.11.2019.

[3] TENORTH, MORITZ AND BEETZ, MICHAEL. KNOWROB: Knowledge Processing for Autonomous Personal Robots. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Piscataway, NJ, USA, 2009), IROS'09, IEEE Press, pp. 4261–4266.

[4] KUNZE, LARS; ROEHM, TOBIAS AND BEETZ, MICHAEL. Towards Semantic Robot Description Languages. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'11)* (2011).

[5] TENORTH, MORITZ; PERZYLO, ALEXANDER; LAFRENZ, REINHARD AND BEETZ, MICHAEL. The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments. pp. 1284–1289.

[6] PARAISO, E. C. AND BARTHES, J. . A. SpeechPA: an ontology-based speech interface for personal assistants. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (Sep. 2005), pp. 657–663.

[7] FELLBAUM, CHRISTIANE. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[8] HYLTON, J. A. Identifying and Merging Related Bibliographic Records. Tech. rep., Cambridge, MA, USA, 1996.

[9] VEGESNA, ANUSHA; JAIN, PRANJAL AND PORWAL, DHRUV. Ontology based Chatbot (For E-commerce Website). *International Journal of Computer Applications 179*, 14 (Jan 2018), pp. 51–55.

[10] F. NOY, N AND MCGUINNESS, DEBORAH. Ontology Development 101: A Guide to Creating Your First Ontology. *Knowledge Systems Laboratory 32* (01 2001).

[11] HOLADA, MIROSLAV AND PELC, MARTIN. *The Robot Voice-control System with Interactive Learning*. 10 2008.

[12] FOSTER, MARY ELLEN; BY, TOMAS; RICKERT, MARKUS AND KNOLL, ALOIS. Human-Robot Dialogue for Joint Construction Tasks. In *Proceedings of the 8th International Conference on Multimodal Interfaces* (New York, NY, USA, 2006), ICMI '06, ACM, pp. 68–71.

[13] WHITE, MICHAEL. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation 4* (01 2006), pp. 39–75.

[14] MORROW, J.D. AND KHOSLA, PRADEEP. Manipulation task primitives for composing robot skills. pp. 3354 – 3359 vol.4.

[15] THOMASON, JESSE; ZHANG, SHIQI; MOONEY, RAYMOND AND STONE, PETER. Learning to Interpret Natural Language Commands Through Human-robot Dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), IJCAI'15, AAAI Press, pp. 1923–1929.

[16] FOSTER, MARY ELLEN AND MATHESON, COLIN. Following Assembly Plans in Cooperative, Task-Based Human-Robot Dialogue. In *Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue (Londial 2008)* (London, 2008).

[17] MELLO, LUIZ S. HOMEM DE AND SANDERSON, ARTHUR C. AND/OR Graph Representation of Assembly Plans. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence* (1986), AAAI'86, AAAI Press, pp. 1113–1119.

[18] SUN, YICHAO AND VELOSO, MANUELA M. Interruptable Autonomy: Towards Dialog-Based Robot Task Management. In *AAAI 2013* (2013).

[19] COLTIN, BRIAN; VELOSO, MANUELA AND VENTURA, RODRIGO. Dynamic User Task Scheduling for Mobile Robots. In *Proceedings of the 9th AAAI Conference on Automated Action Planning for Autonomous Mobile Robots* (2011), AAAIWS'11-09, AAAI Press, pp. 27–32.

[20] NESSELRATH, ROBERT. SiAM-dp : An open development platform for massively multimodal dialogue systems in cyber-physical environments, Dec. 2015.

[21] HAHN, VANESSA. A Dialogue Management Framework for Dialogue Adaption based on User Backchannels. Master's thesis, Saarland University, 2016.

[22] F., PATERNO; SANTORO C., SPANO L. D. AND D., RAGGETT. MBUI - Task Models. `https://www.w3.org/TR/2014/NOTE-task-models-20140408/`, 2014. Accessed: 19.02.2020.

[23] KORSAH, G. AYORKOR; STENTZ, ANTHONY AND DIAS, M. BERNARDINE. A Comprehensive Taxonomy for Multi-robot Task Allocation. *Int. J. Rob. Res. 32*, 12 (Oct. 2013), pp. 1495–1512.

[24] ZLOT, ROBERT MICHAEL. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. PhD thesis, USA, 2006.

[25] STOCK, SEBASTIAN; MANSOURI, MASOUMEH; PECORA, FEDERICO AND HERTZBERG, JOACHIM. Online task merging with a hierarchical hybrid task planner for mobile service robots.

[26] MÜLLER, RAINER; VETTE, MATTHIAS AND MAILAHN, ORTWIN. Process-oriented Task Assignment for Assembly Processes with Human-robot Interaction. *Procedia CIRP 44* (2016), pp. 210 – 215. 6th CIRP Conference on Assembly Technologies and Systems (CATS).

[27] KOCH, PAUL J.; [VAN AMSTEL], MARIKE K.; DĘBSKA, PATRYCJA; THORMANN, MORITZ A.; TETZLAFF, ADRIAN J.; BØGH, SIMON AND CHRYSOSTOMOU, DIMITRIOS. A Skill-based Robot Co-worker for Industrial Maintenance Tasks. *Procedia Manufacturing 11* (2017), pp. 83 – 90. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[28] VAQUERO, TIAGO STEGUN; MOHAMED, SHARAF CHRISTOPHER; NEJAT, GOLDIE AND BECK, J. CHRISTOPHER. The Implementation of a Planning and Scheduling Architecture for Multiple Robots Assisting Multiple Users in a Retirement Home Setting. In *Artificial Intelligence Applied to Assistive Technologies and Smart Environments, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 25, 2015.* (2015).

[29] GHALLAB, M.; HOWE, A.; KNOBLOCK, C.; MCDERMOTT, D.; RAM, A.; VELOSO, M.; WELD, D. AND WILKINS, D. PDDL—The Planning Domain Definition Language, 1998.

[30] KOVACS, DANIEL L. A Multi-Agent Extension of PDDL3.1. *Proceedings of the 3rd Workshop on the International Planning Competition (IPC), 22nd International Conference on Automated Planning and Scheduling (ICAPS-2012)* (2012), pp. 19–27.

[31] BRACHMAN; FIKES AND LEVESQUE. Krypton: A Functional Approach to Knowledge Representation. *Computer 16*, 10 (1983), pp. 67–73.

[32] BOCKLISCH, TOM; FAULKNER, JOEY; PAWLOWSKI, NICK AND NICHOL, ALAN. Rasa: Open Source Language Understanding and Dialogue Management. *ArXiv abs/1712.05181* (2017).

[33] DAVIS, M. *Computability & Unsolvability*. Dover Books on Computer Science Series. Dover Publications, 1982.

[34] SCHWARTZ, T.; FELD, M.; BÜRCKERT, C.; DIMITROV, S.; FOLZ, J.; HUTTER, D.; HEVESI, P.; KIEFER, B.; KRIEGER, H.; LÜTH, C.; MRONGA, D.; PIRKL, G.; RÖFER, T.; SPIELDENNER, T.; WIRKUS, M.; ZINNIKUS, I. AND STRAUBE, S. Hybrid Teams of Humans, Robots, and Virtual Agents in a Production Setting. In *2016 12th International Conference on Intelligent Environments (IE)* (2016), pp. 234–237.

[35] LIGHT, ROGER A. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software 2*, 13 (2017), p. 265.

[36] TRACTAT. `https://tractat.dfki.de/`. Accessed: 17.06.2020.

[37] FAROOQ, UMAR. Multimodal Presentation to Support Human for Resolving Semi-Autonomous Agent Failures Using Mixed Reality and Transfer of Control. Master's thesis, Saarland University, 2019.

[38] Innovationslabor MRK4.0. `https://www.power4production.de/wp-content/uploads/2018/09/MRK4.0_Projektblatt.pdf`. Accessed: 17.06.2020.

[39] JURAFSKY, DANIEL AND MARTIN, JAMES H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[40] MOFFETT FIELD, CA. Task Load Index (TLX): Computerized version (Version 1.0). *NASA, CA: Human Research Performance Group* (1986).