Master Thesis

Florian Johannßen

Nao Robots in the Cloud - An Interface to Transform and Execute Abstract Plans

Fakultät Computer Science
Studiendepartment Computer Science

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Johannßen

Nao Robots in the Cloud - An Interface to Transform and Execute Abstract Plans

Master thesis submitted in terms of
the degree course Computer Science
at the Department of Computer Science
Faculty of Engineering and Computer Science
Hamburg University of Applied Sciences

Supervisor: Prof. Dr. rer. nat. Gunter Klemke
Co-Supervisor: Prof. Dr. rer. nat. Kai von Luck

Submitted on 10.02.2014

Florian Johannßen

Thema der Master Thesis

Nao Robots in the Cloud – An Interface to Transform and Execute Abstract Plans

Stichworte

Knowledge Sharing, Cloud Robotics, Nao, RoboEarth, Robot Operating System, Plan Sharing

Kurzzusammenfassung

Im Rahmen dieser Masterarbeit wird das Thema Cloud Robotics behandelt. Dabei wird insbesondere die Möglichkeit erforscht, wie Roboter mithilfe des Internets abstrakte Pläne untereinander austauschen und verarbeiten können.
Es wurde eine Schnittstelle entwickelt, die es dem humanoiden Roboter Nao ermöglicht, generische Roboterpläne aus einem Cloud Service herunterzuladen und auszuführen, um von den Erfahrungen anderer Roboter profitieren zu können. Dazu wurden mehrere Experimente durchgeführt, um die Benutzbarkeit und Flexibilität dieser abstrakten Schnittstelle zu demonstrieren.

Florian Johannßen

Title of the paper

Nao Robots in the Cloud – An Interface to Execute Abstract Plans

Keywords

Knowledge sharing, Cloud Robotics, Nao, RoboEarth, Robot Operating System, Plan Sharing

Abstract

This master thesis investigates the topic of Cloud Robotics, especially to share knowledge among robots via internet technologies. It implements an interface to execute high-level commands on the humanoid Nao platform that improves the learning mechanisms of the robot.
Through this interface, Nao robots are able to connect themselves with a cloud service to download and process high-level plans which can be generated by different robot platforms. Thereby, this approach permits Nao robots to be no longer on their own as well as to benefit from the experiences of other robots. Final experiments verify the usability of the Nao interface.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Reader‘s guide . . . 3

2 Fundamentals **4**

2.1 Cloud Computing . . . 4

2.2 Robot Operating System . . . 5

2.3 Nao robot . . . 8

2.4 NaoQi Software Development Kit . . . 9

3 Analysis **10**

3.1 Scenario: Household assistance through internet-enabled robots . . . 10

3.2 Remote-brained robots . . . 11

3.3 Cloud Robotics . . . 13

3.3.1 Knowledge sharing for robots . . . 14

3.3.2 Outsourcing compute-intensive tasks . . . 14

3.4 Challenges . . . 15

3.5 Related Work . . . 19

3.5.1 RoboEarth . . . 19

3.5.2 A knowledge sharing Peer-To-Peer system . . . 22

3.5.3 RobotShare . . . 23

3.5.4 DaVinci . . . 23

3.5.5 Cloud-based robot grasping . . . 25

3.5.6 MyRobots.com . . . 26

3.5.7 Nao application store . . . 28

3.5.8 Smartphone robots . . . 28

3.5.9 3D CAD browser . . . 29

3.6 Objectives and requirements . . . 30

3.6.1 Use cases . . . 32

3.6.2 Functional requirements . . . 33

3.6.3 Conclusion 33

4 Design 34

4.1 Nao interface 35
4.2 Approaches of communication 40
4.2.1 Intelligent gateway 41
4.2.2 Transparent gateway 43
4.2.3 Without Gateway 46
4.2.4 Conclusion 48

5 Realization 49

5.1 Interface 50
5.1.1 ROS middleware 50
5.1.2 Exploring engine 51
5.1.3 Nao platform 52
5.1.4 Transformation engine 53
5.1.4.1 Nao-ROS-stack 53
5.1.4.2 CRAM Plan Language 54
5.1.4.3 Process modules 55
5.2 Gateway 62

6 Experiments 63

6.1 Manipulation process module 64
6.2 Navigation process module 66
6.3 Speech process module 66
6.4 Downloading an abstract plan 67
6.5 Résumé 69

7 Evaluation 70

8 Conclusion 72

8.1 Summary 72
8.2 Outlook 73

Chapter 1

Introduction

1.1 Motivation

The internet has become one of the most important communication media. It provides the opportunity to publish and retrieve knowledge globally. Humans are able to perform unknown tasks by asking the internet community for support in different situations. There are user manuals for almost any challenge in everyday life. Through the internet we have not to try out everything on our own. Instead, we benefit from the experiences of others by asking the community. Like humans, robots are also limited in their functionality, knowledge and processing power.
As presented by the International Federation of Robotics [1] IFR (2013), in 2012, about 3 million service robots were sold including robots for household, entertainment, education and research as well as those for handicap assistance. IFR (2013) has predicted that between 2013 and 2016 further 22 million service robots will be delivered to the end user. This number includes almost 15.5 million robots for domestic tasks like vacuum cleaning, window cleaning, lawn-mowing and other types. The amount of toy robots will be probably 3.5 million. Besides, about 3 million robots will be delivered for education and research during the period 2013-2016. Finally, only about 6400 robots will be sold for elderly and handicap assistance, but will be increased substantially within the next 20 years. Related to this forecast, our household and our life will be more depending on service robots during the next three years. That implies that robots have to operate in more dynamic and complex environments than in structured and predictable industrial factories. Since many robots try to solve similar challenges like grasping objects, navigating in household environments or detecting objects it would be helpful if they could support each other by sharing new-learned skills. The question is, if the internet can be used to connect this huge amount of robots with each other.

[1] International Federation of Robotics: `http://www.ifr.org`

Related to:

> "In the first movie of the Matrix trilogy, there is a scene where Neo points to a helicopter on a rooftop and asks Trinity, "Can you fly that thing? " Her answer: "Not yet." Then she gets a pilot program uploaded to her brain and they fly away."
> Guizzo (2011)

For humans, with our non-upgradeable brains, the possibility of acquiring new skills by connecting our heads to a computer network is still science fiction. Not so for robots. Nowadays, companies like Aldebaran Robotics[2] and Willow Garage[3] are able to deliver wireless capable and programmable robots with abstract interfaces. The specific tasks, such as inverse kinematic, voice recognition and path planning are supported by delivered Software Developer Kits.
Thus the preconditions have been created to connect robots with the internet which provides many benefits and applications.

- It permits robots to communicate with each other to improve their learning mechanism.
- Robots are able to download new skills from the internet to solve unknown tasks in foreign environments.
- Robots can offload compute-intensive tasks to remote servers.
- Manufacturers are able to develop cheaper and smarter robots.
- Internet-enabled Robots are no longer developed for one specific task. Household robots can also work in elderly residences without exchanging the software.

The target of this master thesis deals with the realization of the still unexplored approach of knowledge sharing for robots via internet technologies. This work investigates the idea of how a robot is able to perform an unfamiliar task by downloading new knowledge from the internet. An interface will be created to transform and download abstract robot programs inside a plan sharing system from the internet. Thereby, this master thesis enables robots to download commands which could be generated by another robot type with different hardware capabilities.

[2]http://www.aldebaran-robotics.com
[3]http://www.willowgarage.com

1.2 Reader‘s guide

The reader can work himself through this master thesis in a linear fashion, while the interested reader may also decide to concentrate on single chapters. This master thesis is organized in the following chapters.

Chapter 2 represents the basic terms and technologies, which are relevant for this work.

Chapter 3 analyses the topic Cloud Robotics by introducing a scenario and related projects. It aims to demonstrate, which opportunities exist to create knowledge sharing among robots via cloud services. Furthermore, it shows how this master thesis differs from the presented projects as well as defines use cases and requirements to specify the objectives of this work.

Chapter 4 designs an interface for the Nao robot platform to communicate with a cloud service to transform and execute high-level plans. Besides, it introduces several communication approaches to create a plan sharing system including the Nao interface. Finally a conclusion reveals the advantages as well as disadvantages and argues why the chosen concept is implemented in the realization chapter.

Chapter 5 demonstrates the implementation of an interface permitting Nao robots to transform and execute high-level commands. A plan sharing system is introduced, which allows Nao robots to download high-level plans by accessing the web service RoboEarth.

Chapter 6 shows the execution of some experiments to evaluate the correctness and robustness of the Nao interface.

Chapter 7 draws a résumé about the master thesis, which challenges were investigated and how they were solved. Finally, an outlook shows opportunities to continue with the work of this master thesis.

Chapter 2

Fundamentals

This chapter introduces the basic technologies in context of this master thesis.

2.1 Cloud Computing

The following definition of Cloud Computing by the National Institute of Standards and Technology (NIST) Mell and Grance (2011) is taken into account.

> “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The cloud model is composed of five essential characteristics: *on-demand self service, broad network access, resource pooling, rapid elasticity, measured service.* Mell and Grance (2011) defined the following three common types of service:

Software as a Service allows the consumer to access the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices either through a thin client interface, such as a web browser or via a desktop interface. The consumer does not control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities.

Platform as a Service permits the consumer to deploy his own applications onto the infrastructure of a cloud service provider. Furthermore, the consumer does not manage the underlying cloud infrastructure, but has control over his own deployed applications to accommodate the configuration settings to his own preferences.

Infrastructure as a Service provides the consumer to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components.

2.2 Robot Operating System

ROS represents a robotic-specific middleware which is developed as part of the STAIR[4] project as well as the Personal Robots Program[5] at Stanford University in cooperation with the robotic manufacturer Willow Garage Quigley et al. (2009a). Since 2008, Willow Garage supports the further development of ROS. Nowadays, this framework is used by hundreds of research groups and companies in the robotics industry. ROS brings along a quickly growing community. This framework provides a set of tools to support many different common robotic problems like path planning, collision detection, image processing and many more. For example, with the aid of ROS, it is possible to command the robot to walk from place A to place B without thinking about which walking algorithm will be used. This meta operating system is extendable with modules, called stacks. There are modules for many robots like Nao, TurtleBot or PR2.
As described by Martinez and Fernández (2013), ROS-projects are structured in a special way, called stacks. A stack can be a wrapper of a specific robot for the ROS middleware but also a more generic utility library like navigation or image processing. Furthermore, stacks organize the source code by using packages where each of them provides one particular functionality. Through this structuring, ROS allows developers to implement robotic code which is reusable.
In ROS, processes are called nodes which are organized in a Peer-to-Peer network without a central point of communication. Any node in the system can access this network as well as is able to interact with other processes. Nodes are processes, where computation can be performed. Usually, a system includes several nodes to control different functions. Concerning performance, it is recommended to have many nodes providing only one single functionality, rather than a large node processing everything in the system. Nodes are written with an ROS client library which is listed in section 2.2. The master node provides name registration and lookup for all custom nodes. Without the master node, it is not possible to communicate with nodes, services, messages, and others. Figure 2.1 shows an overview about the most important concepts.

[4]STanford Artificial Intelligence Robot: `http://stair.stanford.edu/`
[5]`http://personalrobotics.stanford.edu/`

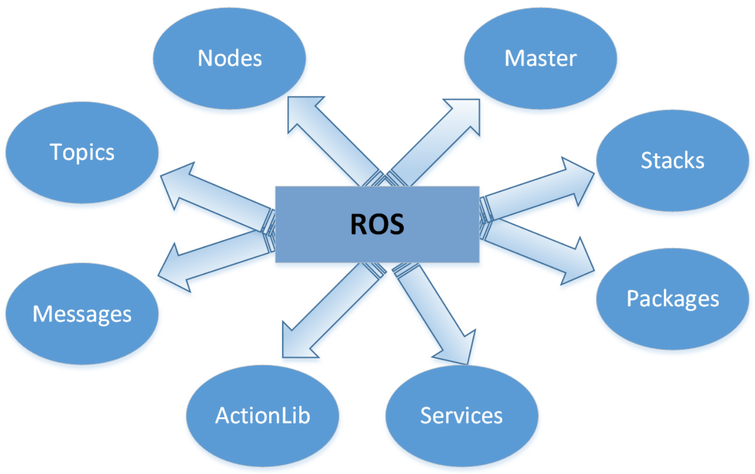


Figure 2.1: Core concepts of the ROS framework Martinez and Fernández (2013)

Client Libraries

ROS provides libraries in several programming languages for the implementation of nodes and inter-process communication[6].

- Main libraries
  - rospy is a pure Python client library for ROS and is designed to provide the advantages of an object-oriented scripting language to ROS. The design of rospy favors implementation speed (i.e. developer time) over runtime performance so that algorithms can be quickly prototyped and tested within ROS.
  - roscpp (C++) presents the most spreaded client library which is mainly used to implement high-performance functionality in ROS.
  - roslisp is a client library for writing ROS nodes in Common Lisp. The library is written to support quick scripting of nodes, and interactive debugging of a running ROS system.

As Woodall (2013) has described, ROS also provides experimental support for the programming languages Java, Lua, EusLisp, R as well as some others.

[6]Describes the problem of how data can be exchanged among multiple threads and processes

Communication between nodes

Generally, nodes communicate with each other through messages. A message contains data which transfers information to other nodes. ROS has several types of standard messages and allows to develop custom types of them.
ROS provides three kinds of patterns for inter-process communication. The first one is to subscribe or to publish data on ROS-Topics. When a node is sending data, it is publishing this information on a topic. Nodes can receive messages from other nodes simply by subscribing to the topic. For subscribing, it is not necessary that a publishing node exists. This permits developers to decouple the production and the consumption. It is important that the name of the topic must be unique to avoid name conflicts. These message streams represent an URL-like approach to transfer information among nodes.
Using services represent the second opportunity to provide an interaction among nodes. When a node includes a service, all the nodes can communicate with it.
The actionlib package provides a standardized interface to support usual client server communication. Through this library it is possible to create a client which sends so-called ROS-Goals to a server to perform tasks with longer computation duration like face detection or laser scanning. The ActionLib toolbox provides mechanisms to cancel the request during the execution or get periodic feedback about the progress of the request. This is not possible by using the publish/subscribe approach or services.

2.3 Nao robot

The Nao robot is developed by the worldwide leading manufacturer of humanoid robots Aldebaran Robotics and is on the market since 2008. Nao is the most applied humanoid robot in education and research. Since 2011, Aldebaran Robotics has announced the newest version of the robot, called Nao Next Gen.

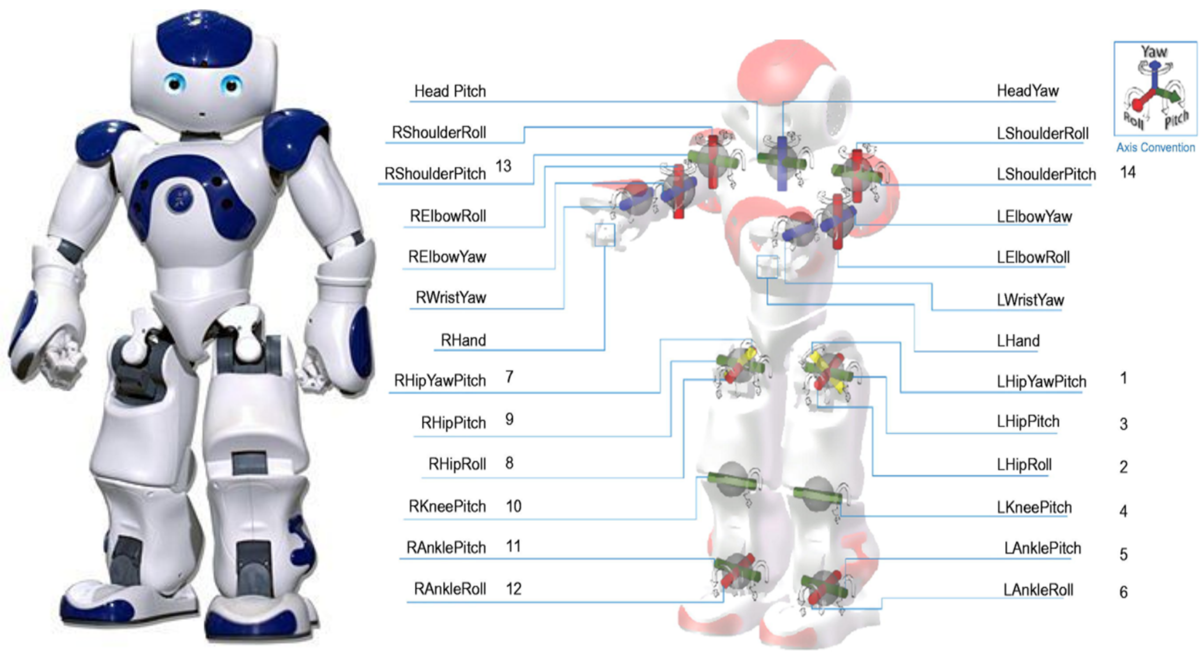


Figure 2.2: Nao Next Gen (Nao (2014); Nao (2012))

The Nao robot is equipped with a 1,6 GHz Intel atom processor, two HD cameras, four mircophones, two loudspeakers as well as a WLAN interface. Besides, Nao provides several sensors to perceive its environment. The Nao robot provides a battery duration of 1,5 hours. The CPU is located in the head of the robot. The detailed documentation of the Nao platform can be found under Aldebaran-Robotics (2013a).

2.4 NaoQi Software Development Kit

The humanoid robot Nao provides the NaoQi SDK[7]. This framework includes a programming interface to develop applications in different programming languages like C++, Java or Python. With the aid of NaoQi, developers are able to control the Nao hardware components. The core of the NaoQi SDK represents the main broker which acts as a server running on a linux kernel inside the “head“. This framework includes several modules to access the joints, face recognition and other functions. The main broker decides which modules should be executed. It is possible to execute source code onboard the robot by loading programs directly into the “head“. Besides, the NaoQi SDK permits developers to create application running remotely on a laptop. Figure 2.3 shows the main broker with most of the Nao modules. To get further information see the documentation Aldebaran-Robotics (2013b)

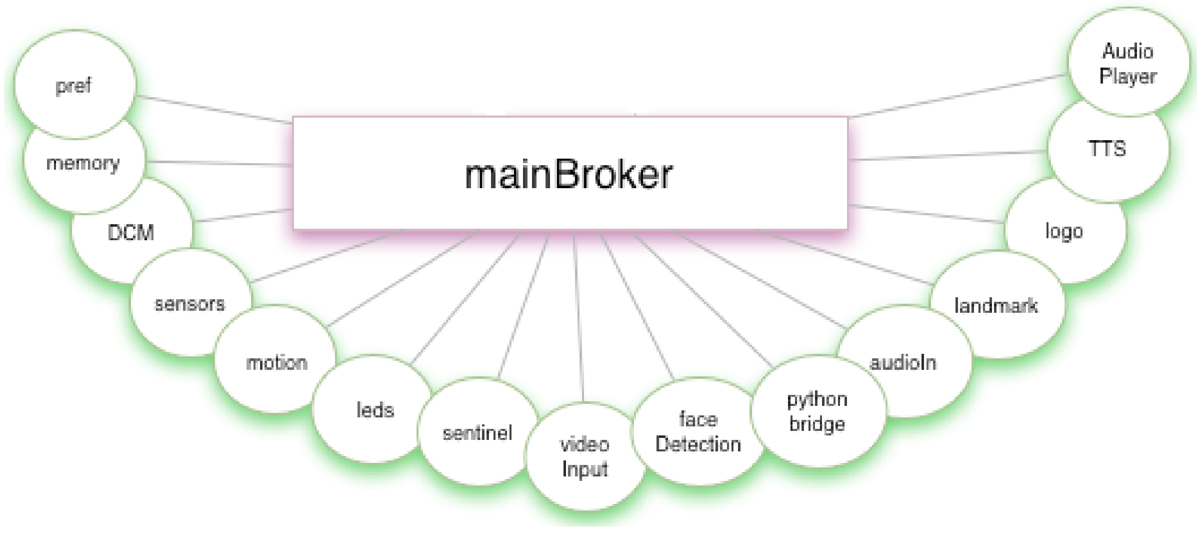


Figure 2.3: NaoQi SDK Aldebaran-Robotics (2013b)

[7]Software Developer Kit

Chapter 3

Analysis

This chapter analyses the topic of Cloud Robotics. It starts with a scenario giving an insight into possible use cases which will be considered in this master thesis. It introduces initial ideas as well as applications like offloading compute-intensive tasks or sharing knowledge among robots. Mainly, this chapter presents an overview about current projects and demonstrates opportunities how to create Cloud Robotics-related systems by discussing common challenges. Finally, use cases and functional requirements concrete the objectives of this work.

3.1 Scenario: Household assistance through internet-enabled robots

Related to Weiser (1991), this scenario is based on the research topic of Ubiquitous Computing. This paradigm means that all computing devices are fully integrated into human environments. Sally is a 85 years old woman who lives alone in an apartment, including a kitchen, bathroom, bedroom and living room. Because Sally is very forgetful, a sensor system via RFID[8] chips is integrated into her apartment to detect the positions of objects like reading glasses or remote control. Since Sally is not able to solve everyday tasks like cooking, washing dishes or setting the table for dinner, she is a proud owner of a new humanoid service robot which is equipped with two legs, two arms with grippers, a head including cameras, loudspeakers and microphones as well as several sensors for collision detection and Wi-Fi access. Sally has bought this robot to get a helping hand for her household. It is assumed that the robot has information about the environment and is able to navigate in Sally´s apartment. The robot is connected with the sensor system to receive the location of the objects. The robot provides an abstract programming interface to be controlled in a high-level way.

[8]Radio-Frequency Identification

It is also equipped with software to support path planning, localization, perception and manipulation. Thus, basic functionality is already provided on the robot. The robot is also connected with the WLAN router of Sally‘s apartment.
So, if Sally commands the robot to bring her the remote control, the robot is able to connect to a service via the internet to receive instructions and information. If another robot has already solved such kind of task in a similar environment and has published its abstract plan on the internet, Sally‘s robot can download the program. Note that this information could be generated by another robot with different hardware capabilities. The downloaded plan is composed of several abstract actions like move to a position or grasping the remote control which the robot can execute as well as meta information about the remote control, like the physical properties. Besides, the robot gets the location of the remote control through asking the sensor system. Based on these new skills the robot is able to bring the remote control to Sally.
Even the seeming simplest tasks such as bringing Sally her remote control require complex decision making. The robot has to decide where to stand in order to reach the object, which hand to use, which grasp policy to apply, where to grasp, how much grasp force to apply, how to lift the object, how much force to apply to lift it and how to hold it. Moreover, if other robots already have answered these questions, it would be efficient for Sally‘s robot to process this information.

3.2 Remote-brained robots

The consideration to connect robots with remote servers has already been researched in the 1990s by Inaba (1993). Inaba tried to minimize the needed hardware resources of the robot. It was very difficult to execute experiments with robots in a real environment, because they had to carry along heavy computers for computation. Inaba‘s approach solved this problem by designing a robot system which separates the brain from the body as shown in figure 3.1. It opens the way to access intelligent software systems from the brain by different robot bodies through radio links.
As presented in Figure 3.1, the brain-body-interaction takes place via the *Brain-Body Interface*. The body is equipped without a computer. Instead, it provides sensors for perceiving and actuators for operating in the real environment. Besides, it includes transmitters to send data to the particular subsystem. The vision subsystem receives video signals for encoding and the sensor subsystem gets sensor signals for analyzing. As shown in figure 3.1, the interface forwards this information to the brain which provides parallel processing via a powerful transputer[9] network. The brain is outsourced to the so-called mother environment. This software receives all sensor information. After processing, it sends the computed output back to the *Brain-Body Interface* to control the robot.

[9]Microprocessor architecture intended for parallel computing

The control subsystem delegates commands to the signal receiver of the body. The brain benefits from the evolution of the mother environment which means if the mother is upgraded to a more powerful computer the brain gains more computation power.
Note that different robots access the mother environment with the same wireless interface. The performance of the wireless interface between the brain and the body is the key in the remote-brained approach. Inaba et al. (1994) investigated how to design a lightweight mechanical animal as a remote-brained robot which outsourced a powerful vision system to the mother environment. The implemented vision system inside the brain of the robot provides dynamic and adaptive interaction capabilities with humans. Experiments have shown that this approach permits free movement of mechanical animals by wireless interaction with the remote brain.

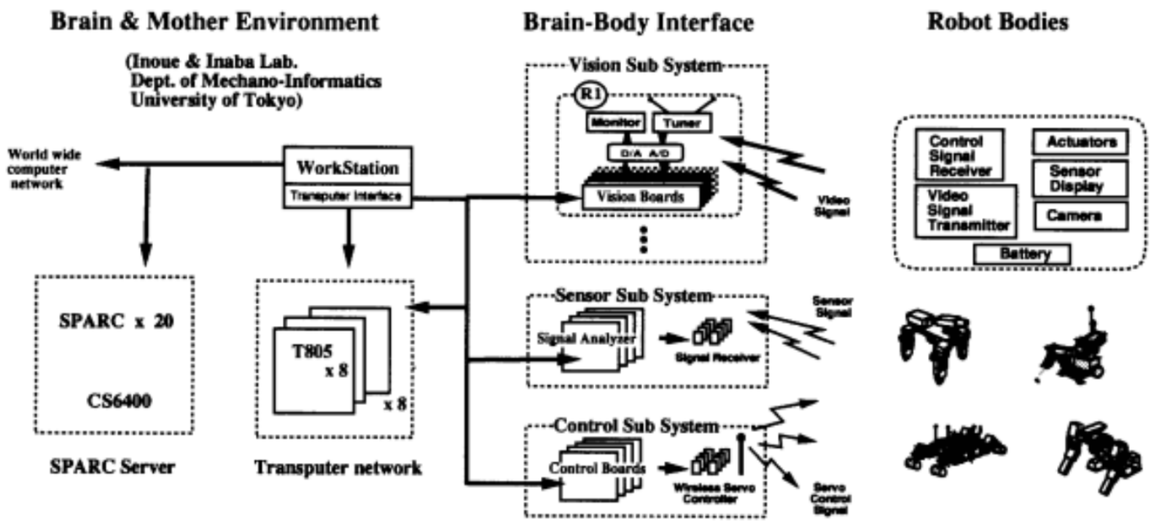


Figure 3.1: Description of the remote-brained robot approach Inaba (1993)

Furthermore, Inaba et al. (2000) also developed a platform for robotic research based on the remote-brained approach presented in figure 3.1. It shows different types of robot bodies which interact with the brain via radio links. Besides, Inaba et al. (1997) designed a humanoid robot system with a remote brain as well. The body of the biped robot was equipped with several sensors, actuators and a power source. The brain provides real-time parallel processing as well as a software mechanism for sensor monitoring and motion control. Related to the following quote:

> “For robotics research, this approach opens the way to use of large-scale powerful parallel computers." Inaba (1993)

These works have established the way to connect robots with the cloud.

3.3 Cloud Robotics

Kuffner (2011) and Quintas et al. (2011) have introduced the topic Cloud Robotics. This idea provides a physical separation between the hardware and software components of the robot. The conventional hardware devices of a robot, such as sensors, actuators, cameras and loudspeakers are still on the robot. The difference to the usual approach, which designs the software on the robot, is that the brain of a robot is outsourced to the cloud. Related to the idea of Inaba (1993), the mother environment is similar to the cloud. Since wireless communication is established as a high-performance and secure communication medium which provides huge bandwidths, Cloud Robotics pushes the idea of Inaba (1993) with the aid of unbounded computation power on demand to the next level. Besides, web services with smart interfaces as well as standard protocols like SOAP[10] and HTTP[11] permitting an easier connection to remote servers, compared to the research of Inaba. According to the next quote:

> “What if robots and automation systems were not limited by onboard computation, memory, or programming? This is now possible with wireless networking and rapidly expanding Internet resources. " Kuffner (2011)

Cloud Robotics allows robots to offload compute-intensive tasks like image processing, voice recognition and even downloading new skills instantly. Robots communicate with the cloud via standard protocols like HTTP or HTTPS[12]. This approach provides the following two possible usages.

[10]Simple Object Access Protocol
[11]Hypertext Markup Language
[12]Hypertext Markup Language Secure

3.3.1 Knowledge sharing for robots

The idea of knowledge sharing aims to exchange information among heterogeneous robots as presented in figure 3.2. Robots are heterogeneous if they have different capabilities and hardware resources. Homogenous means that robots belong to the same platform. Quintas et al. (2011) described that robots benefit from the experiences of others and improve their learning mechanism through this concept. They investigated the possibility of how cloud-enabled robots upload recently new-learned information (see figure 3.2, part *a*) to the cloud. Furthermore, robots are able to download knowledge from the cloud (see figure 3.2, part b) to solve tasks in foreign environments.

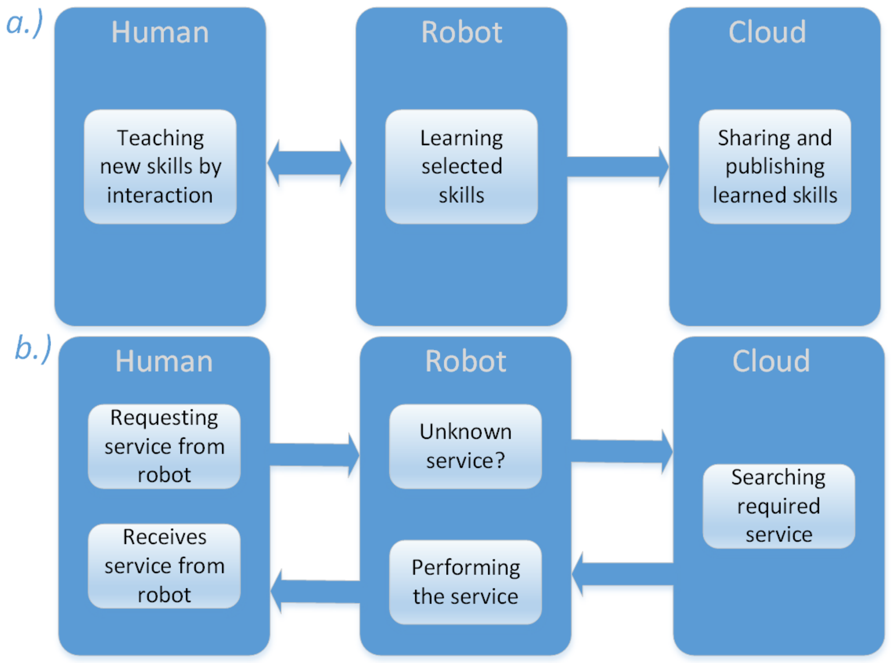

a.)
Human
Teaching new skills by interaction
Robot
Learning selected skills
Cloud
Sharing and publishing learned skills
b.)
Human
Requesting service from robot
Receives service from robot
Robot
Unknown service?
Performing the service
Cloud
Searching required service

Figure 3.2: Knowledge sharing for robots

Cloud-connected robots are not longer developed for a few specific tasks, as it used to be in the past. If a robot shall perform an unknown task, this design approach allows the robot to download the required new knowledge.

3.3.2 Outsourcing compute-intensive tasks

Cloud Robotics also provides the possibility to offload heavy robotic tasks, like speech recognition, face detection, inverse kinematic, grasping policies, image processing, SLAM[13] algo-

[13] `SimultaneousLocalizationandMapping`

rithm and many more to remote servers. These services provide massive parallel computation on demand for cloud-enabled robots. Thanks to this idea robots don‘t have to be equipped with heavy computation power and big power sources that gives manufacturers the opportunity to construct cheap, smart and lightweight robots as shown in subsection 3.5.8.

3.4 Challenges

The concept of Cloud Robotics brings along many problems which have to be solved by developers of cloud-based robotic systems. This section demonstrates opportunities to counter typical challenges for developers who want to create platforms either to provide efficiently implemented robotic algorithms as a service or systems for sharing robotic-specific knowledge.

Communication

Once computation and knowledge is swapped out to remote servers, the communication between robots and a cloud service plays a very important role, related to performance and functionality in cloud-based robotic systems. It is necessary to decide which application layer protocol has to be used. The appropriate communication protocol depends on the cloud service. If a cloud service provides a mail server, the robot has to interact via an application layer protocol like SMTP[14]. FTP[15] will be used to transfer files between the client and the cloud service. The most significant application layer protocol represents HTTP[16] which is used to transport HTML documents for layout definitions and content in different description formats like XML[17] , JSON[18] or plain text between client and server. A cloud service which is accessible via HTTP, is called web service. To send robotic-specific information to a cloud service, it is useful to encode these data as XML or JSON format which can be communicated by support of HTTP. On the one hand, HTTP is a none-real-time protocol. Therefore, a robot-cloud communication with HTTP is not useful for robots which have to operate in time critical environments where latencies are not permitted. On the other hand this protocol is very useful to send knowledge of heterogeneous robots in XML or Json format to a cloud service because of its standardization. Therefore, every internet-enabled robot platform which provides a software developer kit allows to establish a HTTP-Connection to a cloud service. Another benefit of using HTTP is that the transferred data will not be blocked by a firewall. Finally, the Web RTC[19] project permits a real time communication for web browsers which is supported by Google Chrome, Mozilla Firefox and Opera.

[14]Simple Mail Transfer Protocol
[15]File Transfer Protocol
[16]Hypertext Markup Language
[17]Extensible Markup Language
[18]JavaScript Object Notation
[19]Web Real-Time Communication: `http://www.webrtc.org`

Heterogeneity of robots

Probably the hardest challenge in this research area is to try knowledge sharing among heterogeneous robots which are equipped with different hardware properties as well as other capabilities. The monkey and banana problem is a popular challenge in artificial intelligence. A monkey has to reach a bunch of bananas which is fixed at the ceiling. The monkey is equipped with a chair and a stick to reach the bunch. This is a common planning problem where the monkey has to find the ideal sequence of instructions to solve this task. Related to knowledge sharing for robots, if a robot wants to share its ideal plan with a heterogeneous robot, it might be that this plan is not useful for another robot. To solve this challenge, intelligent reasoning mechanisms are needed to decide if a plan is practicable. Besides, a biped robot performs tasks in another way than a drone or a wheeled robot. Furthermore, through different hardware equipment and capabilities, heterogeneous robots are not able to communicate directly with each other. A middleware can assist to abstract from the robot specific hardware properties. Such an abstraction layer like ROSQuigley et al. (2009b), Yarp Fitzpatrick et al. (2008), Orocus Smits et al. (2008) and Player Gerkey et al. (2003) hides and encapsulates all robotic-specific capabilities like how the motion component of a robot moves the arm. With the aid of a middleware it is possible to instruct the robot to move the arm to a specific position in an abstract way without thinking about how the robot executes this task.

Knowledge representation

Moreover, knowledge which shall be exchanged among different robot platforms needs to be defined in a standard semantic representation, which every robot is able to process. A knowledge description language is required to provide robots with all information they need from the knowledge base. Earlier research on knowledge representation for actions or objects usually did not deal with this kind of meta-information, but rather focused on the representation of the information itself, for example in Hierarchical Task Networks Kutluhan et al. (1994) and related languages for plan representation Myers and Wilkins (1997) or the Planning Domain Definition Language developed by Ghallab et al. (1998). Exchange formats like the Knowledge Interchange Format KIF by Genesereth and Fikes (1992) are very expressive and generic languages, but have limited reasoning support. In particular, a representation languages has to provide techniques for describing:

- Actions and object positions inside an environment
- Meta-information about physical properties of objects
- Self-models of a robot's components which describes its capabilities

Knowledge reasoning

Common reasoning mechanisms validate a plan if the robot already has all information which is needed to execute a specific task successfully. Thereby, inference systems are required which give a robot helpful advices how to handle plans with missing components.

Robot descriptions

The next challenge describes the problem of the gap between high-level plans like „Pick up the cup with the right hand" and the corresponding low-level robot commands which need to know the physical properties of the cup object. To fill this gap, a description language is required which specifies the skills of a robot, including actuators, sensors and control algorithms to match these capabilities against the requirements of an abstract plan. Through this progress it is possible to check if knowledge is practicable before the robot downloads the information. Robot description languages increase the reusability of robot control programs and offer developers the possibility to implement software that works on such description models and not only on the particular robot instance.
For example, the recent work by Lu (2012) has presented URDF[20] which can be used as description language to specify the kinematics, a visual representation as well as a collision model. Developers can use this concept to calculate the 3D position of a robot and detect potential collisions between the robot and its environment. The description contains information about the components of the robot, like left-gripper or right-gripper but the robot does not know the semantic of it. URDF is not designed for specifying robot components such as sensors, actuators and matching those against task requirements.
COLLADA[21] developed by Barnes et al. (2008), is a XML[22] schema which is designed to describe 3D objects including their kinematics. It mainly focuses on modeling information about scenes, geometry, physics, animations and effects. But similar to URDF, it lacks elements for describing sensors, actuators and software.
Furthermore, Kunze et al. (2011) created SRDL[23] to specify robots, capabilities and everyday actions. Inference algorithms are used to match action descriptions to components of robots via the concept of capabilities. Actions and capabilities are represented by the means of ontologies. SRDL includes also inference algorithms.

Execution of abstract plans

To process downloaded knowledge from the internet, the robot has to provide an execution engine to perform instructions. This challenge describes the problem of how to execute an

[20]Unified Robot Description Format
[21]COLLAborative Design Activity
[22]Extensible Markup Language
[23]Semantic Robot Description Language

abstract plan on the robot platform. There are several opportunities to solve this problem. McDermott (1991) developed the Reactive Plan Language as LISP-based notation for abstract robot planning. It provides high-level concepts like monitors, interrupts, concurrency, failure handling and top-level task descriptions. Another approach is using the CRAM[24] Plan Language (CPL) which is created by Beetz et al. (2010) to execute high-level robot programs. It is based on the ideas of McDermott (1991). This language allows to profit from techniques like failure monitoring, creating goals and defining locations where objects shall be placed. Such locations are usually described using abstract specifications like 'in reach of the patient', and need to be translated into actual metric positions. The goal concept is attributabled to the work of Beetz and McDermott (1992). By defining goals, it is possible to command a robot in an abstract way without thinking about which plan is used to reach the goal. Additionally, SMACHBohren (2010) represents a hierarchical concurrent state machine to rapidly create complex robot behaviour. SMACH provides a Python library that takes advantage of very old concepts in order to create robust robot behaviour with maintainable and modular code very quickly.

Brainless robots

Another problem occurs if the wireless communication to the cloud breaks down. Without a connection to the cloud, the robot becomes brainless. So it could be useful to place default behaviour directly on the robot. One opportunity could be to store all knowledge which the robot has downloaded from the cloud in its own database to be limited independent of the cloud. Provided that the memory resource of the robot is not overworked, it could be helpful to replace older plans which the robot normally does not use via common caching policies. If the cloud connection is broken down in the approach of an intelligent gateway which is demonstrated in subsection 4.2.1, all knowledge is gone. Default behaviour for manipulation, navigation, perception and text-to-speech could be placed directly on the robot to counteract this challenge.

Security

Through the combination of robotics and cloud computing, robots inherit the advantages as well as the disadvantages of cloud computing. Robots which are operating in a hospital store private information about patients in a cloud to share these with other robots. So, it is important to consider well-known security issues like data loss, botnets, phishing and manipulation of information. Besides, if robots download knowledge from unknown cloud services it cannot be assured that these information have not been manipulated. Therefore, verifying mechanisms are required.

[24]Cognitive Robotic Abstract Machine

Symbol grounding problem

This challenge describes the problem of how a robot can process information whereas it does not know the semantic of it. This issue includes the well known grounding problem, introduced by Harnad (1990) from the psychology research area. Before the robot can execute data, it has to assign semantic to these meaningless symbols.

3.5 Related Work

Recently, more and more companies and universities are interested in research on the topic of cloud robotics. The following projects are some that are currently working on this topic.

3.5.1 RoboEarth

The project of Waibel et al. (2011) introduces RoboEarth which represents a World Wide Web for robots. This cloud service can be used by heterogeneous robots to share information among each other. Through RoboEarth robots are able to share information and experiences about objects, actions and environments. This project is part of the Cognitive Systems and Robotics Initiative[25] founded by the European Union. The RoboEarth platform is divided into the following segments.

RoboEarth storage

RoboEarth provides a distributed database to store information like object descriptions, maps and task specifications which is accessible by using a web interface. Graphic 3.3 shows the core idea of RoboEarth. Related to challenge 3.4, this project uses the RoboEarth language which represents an extension of the Ontology Web Language. This description format, developed by Tenorth et al. (2012), is used to describe the semantic relations between the objects, actions and environments.
The knowledge processing system KnowRob, created by Beetz and Tenorth (2009), aims to explore the RoboEarth database for a plan like „Serve a drink" with the aid of its internal Prolog engine. If KnowRob finds a plan which satisfies the request, it checks the capabilities of the robot against the requirements of the plan. If successful, KnowRob downloads the plan and translates it to the CRAM Plan Language, which is a Lisp-like language for the implementation of abstract plans. Besides, the KnowRob component provides inference mechanisms. For example, it is able to reason that milk is drinkable and can be found in the fridge. If a robot does not find milk in the kitchen, it can request KnowRob for assistance.

[25] `http://cordis.europa.eu/projects/rcn/93278_en.html`

Beetz et al. (2010) have developed the Cognitive Robotic Abstract Machine Plan Language for high-level plan execution on heterogeneous platforms.

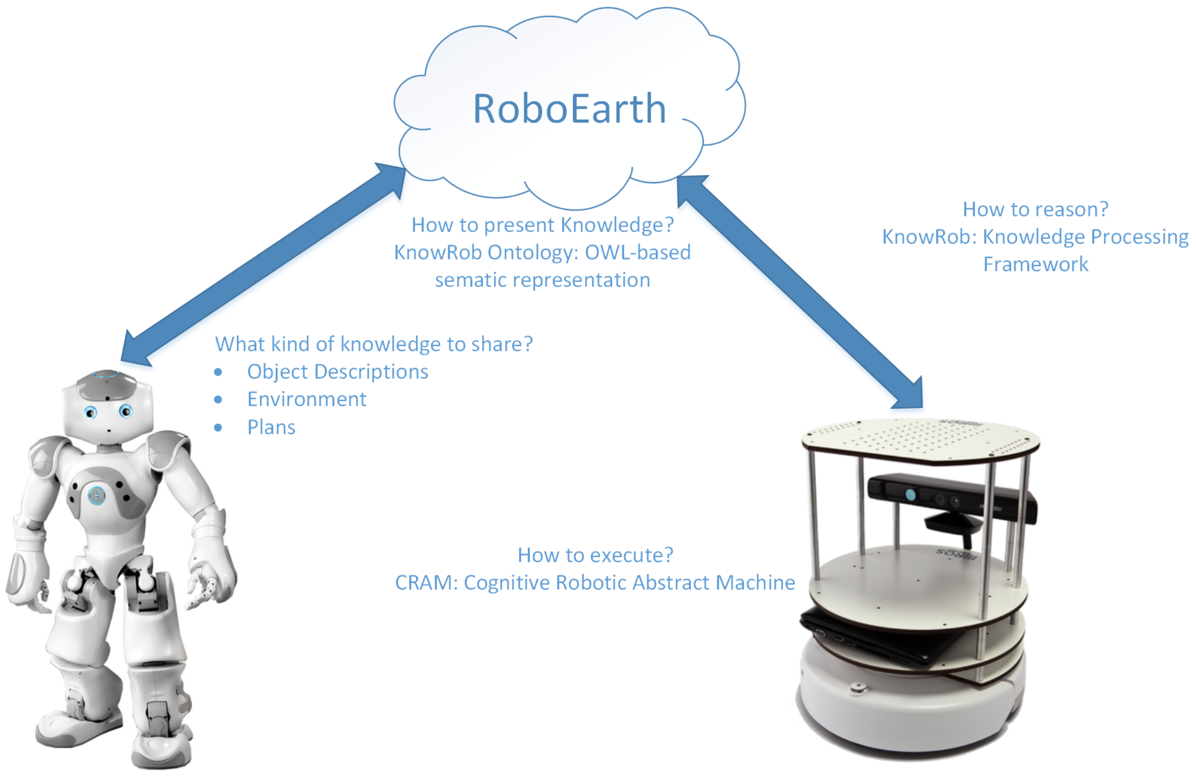

RoboEarth
How to present Knowledge?
KnowRob Ontology: OWL-based
sematic representation
How to reason?
KnowRob: Knowledge Processing
Framework
What kind of knowledge to share?
• Object Descriptions
• Environment
• Plans
How to execute?
CRAM: Cognitive Robotic Abstract Machine

Figure 3.3: RoboEarth (Waibel et al. (2011); Nao (2011); TurtleBot (2014))

The RoboEarth team demonstrated in some experiments how heterogeneous robots are able to share executable plans among each other. Note that all components are integrated into the ROS middleware. Thereby, the robots have to be compatible with the ROS abstraction layer. Trials have shown knowledge exchange among the Amigo[26] robot and the PR2[27].

[26]`http://www.techunited.nl/en/amigo-robot`
[27]`http://www.willowgarage.com/pages/pr2/overview`

Rapyuta

With their RoboEarth database, the RoboEarth team provides a very efficient approach to store robot-specific information in the cloud. On the one hand they solved the problem that robots were not able to access remotely stored knowledge so far. On the other hand the processing elements of this information happen locally on the robot. Therefore, the cloud engine Rapyuta was developed by Hunziker et al. (2013) to outsource robotic algorithms to the cloud as well. Figure 3.4 shows that by extending the RoboEarth database, robotic-specific information and processing units are both settled in the cloud.

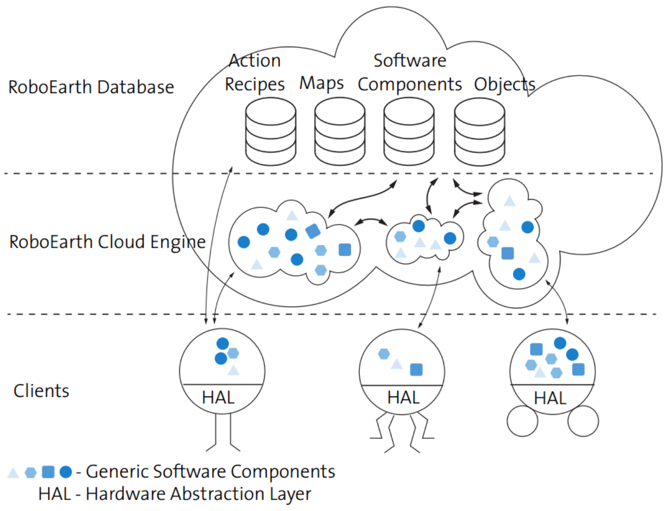


Figure 3.4: Rapyuata Gajamohan (2013)

Conclusion

According to the relevance of this project to the master thesis, RoboEarth plays an important role because it presents a very attractive approach to share knowledge between heterogeneous robots with the aid of elegant as well as powerful technologies. On the one hand it brings along a lot of functionality to store robotic-specific knowledge in an efficient way. On the other hand if robots want to process downloaded knowledge they have to provide a specific interface to transform abstract commands to low-level instructions. Furthermore, this project created huge attention in the research area of cloud robotics and is still in a continual advancement process.

3.5.2 A knowledge sharing Peer-To-Peer system

The design of an efficient collaborative multirobot framework that ensures the autonomy and the individual requirements of the involved robots is a very challenging task. This requires designing an efficient platform for inter-robot communication. P2P is a good approach to achieve this goal. P2P aims to make the communication ubiquitous by crossing the communication boundary and has many attractive features to be used as a platform for collaborative multi-robot environments. Ogata et al. (2011) has implemented a P2P system based on JXTA Overlay. In this paper, a JXTA Overlay is used as a platform for robot collaboration and knowledge sharing. They evaluated the knowledge sharing system by many experiments. Results have shown that the proposed system has a good performance and can be used successfully for knowledge sharing between robots.

Conclusion

Compared to RobotEarth, where robots cooperate with each other via a cloud infrastructure, this approach demonstrates a knowledge sharing by using a different communication pattern. Since this master thesis pursues the target to provide a knowledge sharing via a central point of communication, this project has only a limited relevance for the thesis.

3.5.3 RobotShare

Another approach by Fan and Henderson (2008) has brought out a search engine like Google for robots. RobotShare represents another web based approach for knowledge sharing among robots, similar to RoboEarth. Concerning this challenge they developed a search engine permitting robots to acquire and publish information about objects and plans. The knowledge is efficiently indexed. When robots request the web repository for something like how to wash cookware, they get the URL-reference of the knowledge as shown in 3.5.

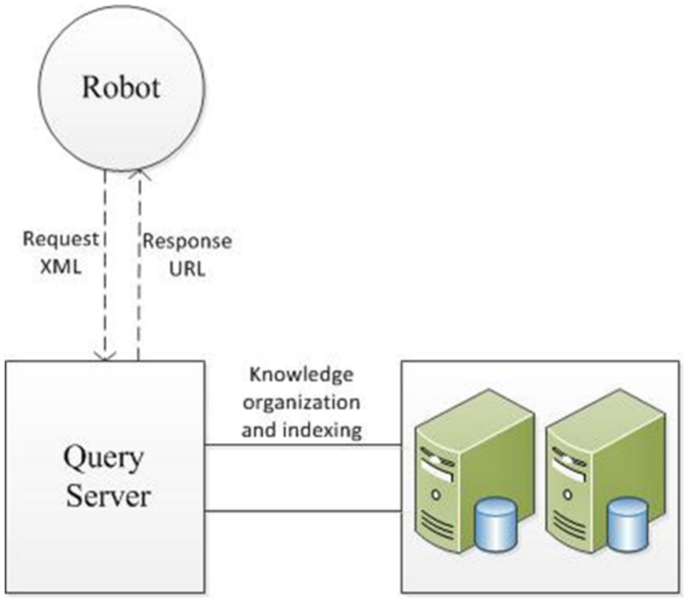


Figure 3.5: RobotShare

Conclusion

On the one hand RobotShare shows an efficient concept to index robot-specific knowledge which allows robots to acquire information very quickly. But on the other hand the received information is defined in natural language and is not executable. Because of this missing feature, RoboShare has no bearing on this work.

3.5.4 DaVinci

The work of Arumugam et al. (2010) presents DaVinci, a cloud service for robots to swap out computationally intensive robot algorithms like path planning, sensor fusion or simultaneous localization and mapping to remote servers.

Figure 3.6 shows the architecture which includes mainly a Hadoop[28] cluster and a DaVinci server. The Hadoop cluster includes the data storage as well as computation nodes to provide parallel processing on this information. Through Hadoop's HDFS[29] filesystem, the information can be stored efficiently. DaVinci uses the Map/Reduce framework to facilitate parallel execution of tasks. Compared to running the same task on a single server, this reduces the processing time of computationally intensive tasks significantly. The ROS middleware is used to provide communication between the cloud service and the robots. Through the ROS abstraction layer it is possible that heterogeneous robots can request the DaVinci cloud service. Robots can share their sensor information by uploading them to the DaVinci server which operates as a central point of communication for the robots. Therefore, ROS nodes are running directly on the robot to send this information to the DaVinci server which includes a running ROS master node to collect the sensor data. This service can be accessed over an intranet connection as a private cloud or over the internet via ROS messages which are wrapped into HTTP requests and responses. DaVinci opens the way to aggregate sensor data from heterogeneous robots to create global maps. Hence, the more robots share information, the better robots can navigate in an unknown environment.

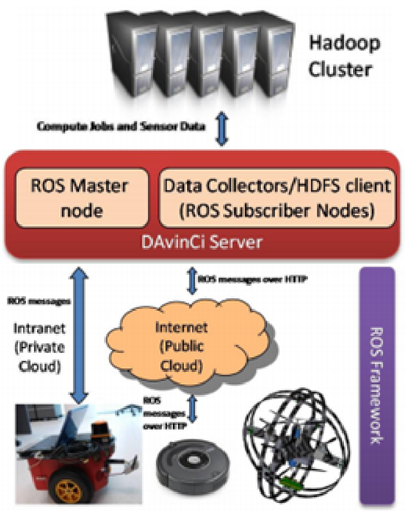


Figure 3.6: DaVinci Arumugam et al. (2010)

Conclusion

Similar to RoboEarth, this project provides exchange of sensor information only for ROS-enabled robots. This master thesis considers especially projects which aim to share programs among robots.

[28]`http://hadoop.apache.org/`
[29]Hadoop Distributed File System

DaVinci is focused on the aggregation of sensor information which are generated by heterogeneous robots to create global maps. Because of that, this project will not be considered much while creating this thesis. Nonetheless, the communication between the ROS-based robots and the DaVinci server is interesting for this master thesis.

3.5.5 Cloud-based robot grasping

Another project of Kehoe et al. (2013) uses the cloud robotics approach to provide a cloud-based grasping service with the aid of the Google object recognition engine, which is established by the Google Goggles project. Google Goggles represents an image recognition service for mobile devices. Usually a Smartphone sends an unknown picture about an object or landmark to this cloud service. The recognition service analyzes the snapshot and returns a generated description back to the user. This project aims to support a robot finding the right grasping policy efficiently for an unknown object. The project is structured into two phases. Figure 3.7 shows the offline phase which aims to train the recognition server by storing semantic information about objects like description, identifier, weight, surface properties and 3D model in the Google Cloud Storage. These work has to be done by humans. Through the GraspIt! toolkit, the stored CAD[30] model is used to pre-compute a set of grasping strategies. Finally, the service stores these policies in its cloud storage.

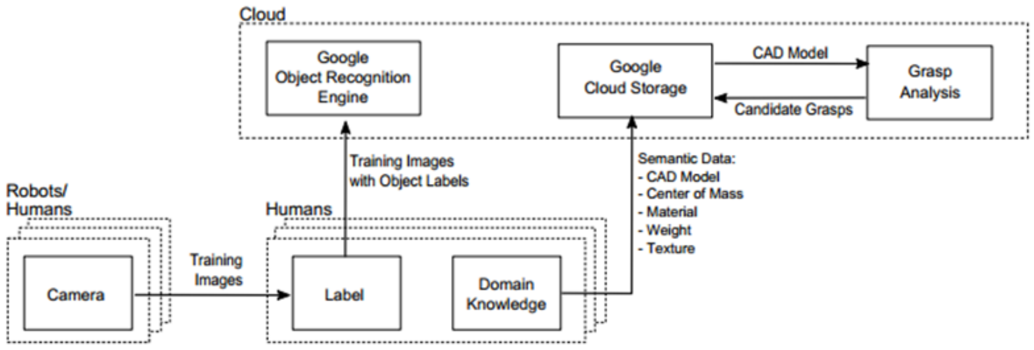


Figure 3.7: Cloud-Based Robot Grasping - Offline Phase Kehoe et al. (2013)

The following graphic 3.8 demonstrates the online phase where robots upload images about objects to receive a possible grasping policy.

[30]Computer-Aided Design

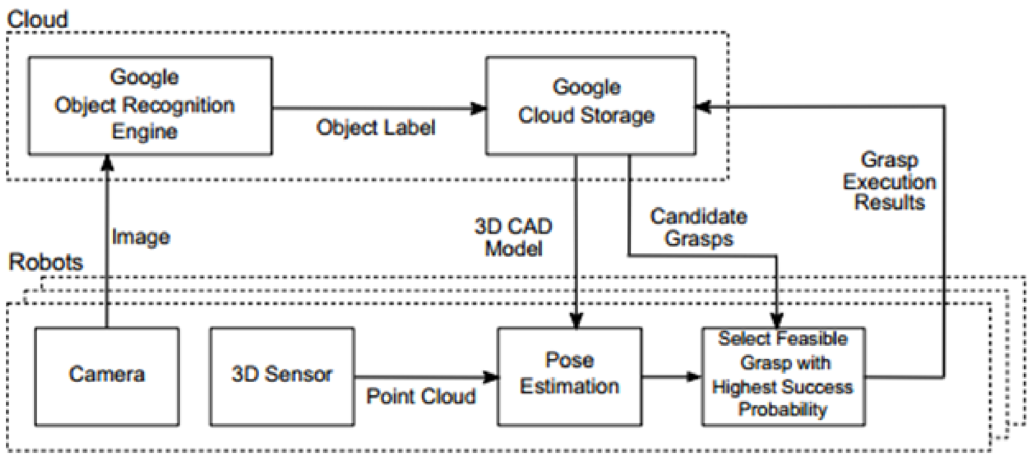


Figure 3.8: Cloud-Based Robot Grasping - Online Phase Kehoe et al. (2013)

A Robot uses its onboard camera to create a 2D image of the unknown object and sends this picture to the cloud service for analyzing. The robot receives the 3D CAD model and a set of grasping candidates, as shown in figure 3.8. After the estimation of the object position where the robot calculates the reference point to grasp, it selects the grasping strategy with the highest success probability. Finally, the execution results will be stored to improve this process. Kehoe et al. (2013) executed experiments with the two-arm mobile manipulator PR2 robot produced by Williow Garage and six objects (air freshener, candy, juice, mustard, peanut butter and soap) which are stored in the cloud storage with pre-computed grasping candidates. The results showed a failure rate between 0 - 23 %.

Conclusion

This project demonstrates a valuable approach to use cloud-based infrastructure to analyze possible grasping policies. Especially, the usage of the Google Cloud Platform[31] presents an interesting option to connect robots with the internet for further work of this thesis.

3.5.6 MyRobots.com

The MyRobots.com project[32] proposes a social network for robots which is launched by the company RobotShop[33]. Users can create profiles for their robots, similar to Facebook, which allow robots to publish updates like status information about their power source or their actual position.

[31]https://cloud.google.com/
[32]http://www.myrobots.com
[33]http://www.robotshop.com

In the same way humans benefit from socializing, collaborating and sharing information, robots can also benefit from those interactions by sharing their sensor information and giving an insight into their current state. MyRobots.com aims to connect all robots and intelligent devices to the internet as shown in 3.9. This approach improves the capabilities of the robots by online monitoring and remote controlling. This project strives to provide cloud robotics accessible to everyone and everything and aims to increase the involvement of robotic developers. Internet-enabled robots can connect directly to the MyRobots.com cloud service. For robots without internet access, the MyRobots-Connect adapter can be used as Serial-to-Ethernet gateway to be connected with the platform. Through MyRobots.com users are able to monitor and control their robots via a web interface remotely as well as to receive alerts and problem reports. Furthermore, users obtain support from the robot hospital service which provides algorithms to diagnose defective robots. Users are also able to develop custom robotic applications by using the API[34] of MyRobots.com and upload these to the platform. Besides, it is possible to download applications directly on the robot, similar to 3.5.7.

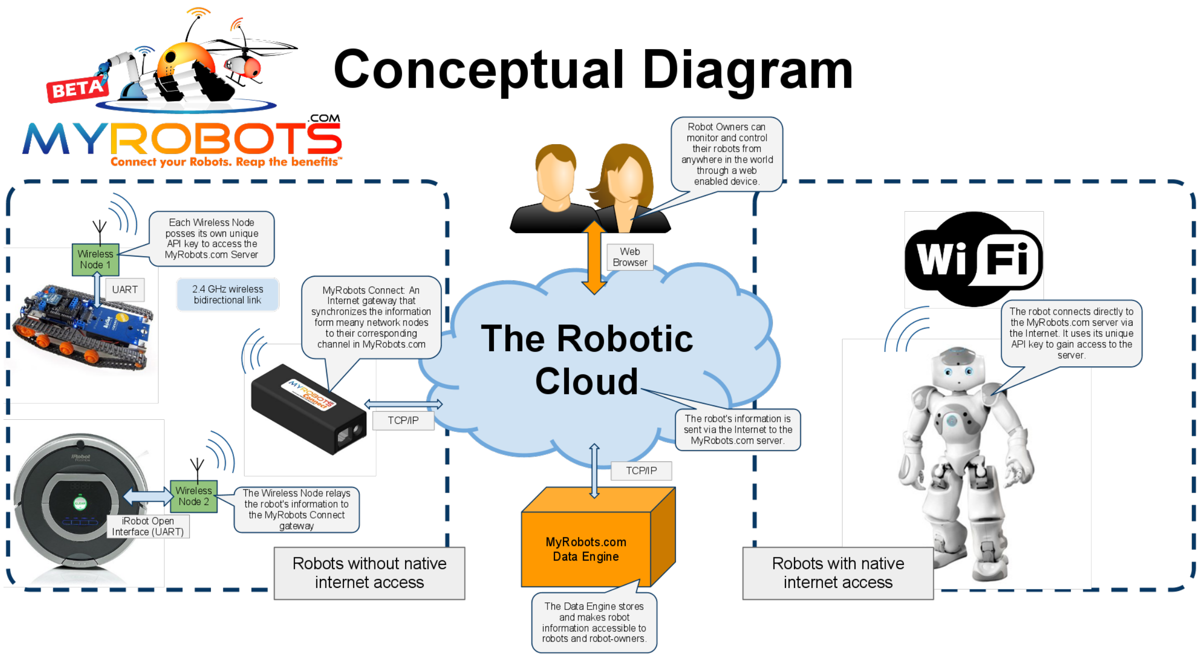

Conceptual Diagram
BETA
MYROBOTS.COM
Connect your Robots. Reap the benefits™
Wireless Node 1
Each Wireless Node posses its own unique API key to access the MyRobots.com Server
UART
2.4 GHz wireless bidirectional link
MyRobots Connect: An Internet gateway that synchronizes the information form meany network nodes to their corresponding channel in MyRobots.com
TCP/IP
Wireless Node 2
iRobot Open Interface (UART)
The Wireless Node relays the robot's information to the MyRobots Connect gateway
Robots without native internet access
Robot Owners can monitor and control their robots from anywhere in the world through a web enabled device.
Web Browser
The Robotic Cloud
The robot's information is sent via the Internet to the MyRobots.com server.
TCP/IP
MyRobots.com Data Engine
The Data Engine stores and makes robot information accessible to robots and robot-owners.
Wi Fi
The robot connects directly to the MyRobots.com server via the Internet. It uses its unique API key to gain access to the server.
Robots with native internet access

Figure 3.9: MyRobots.com MyRobots.com (2013)

Connecting robots to the internet can have many benefits for humans and robots. Humans get better user-experiences with handling robots. Besides, it is possible to collect statistic information to improve the behaviour of the robot.

[34] Application Programming Interface

Conclusion

This approach achieves an easy connection of robots to a cloud service. It allows developers to download applications for their robots and provides monitoring mechanisms. It provides no knowledge sharing among different robot platforms like RoboEarth or DaVinci. Robots are only able to execute programs which are designed for their specific platform. Due to this restriction, this project has not much relevance for this master thesis.

3.5.7 Nao application store

Aldebaran Robotics, the manufacturer of the humanoid Nao robot, has created a store for Nao-specific applications. Users who joined the Nao community[35] are able to share their Nao programs. Furthermore, users can download a Nao application through the web interface directly on the Nao robot.

Conclusion

Similar to MyRobots.com, the Nao robots cannot perform applications which are generated by heterogeneous robots as in RoboEarth. Since this thesis aims to create a knowledge sharing system where the information can be generated by heterogeneous robots, this project will not take much influence. On the other hand the Nao application store represents a necessary comparison project.

3.5.8 Smartphone robots

Recently, another trend is observed to build smart robots. Because of the rapid and continuous spreading of smartphones, many startups have focused on the development of lightweight robots which are connected with a smartphone to use its computation power and services like voice recognition and face detection. The concept of this approach is that the user has to download the robot-specific applications from the appstore and integrates his smartphone into the robot. Recently, more and more smartphone robots are available like ROMO[36], Oddverx[37] and SmartBot[38] which are presented in the next picture 3.10.

[35] `https://community.aldebaran-robotics.com`
[36] `http://romotive.com`
[37] `http://www.oddwerx.com`
[38] `http://www.overdriverobotics.com`

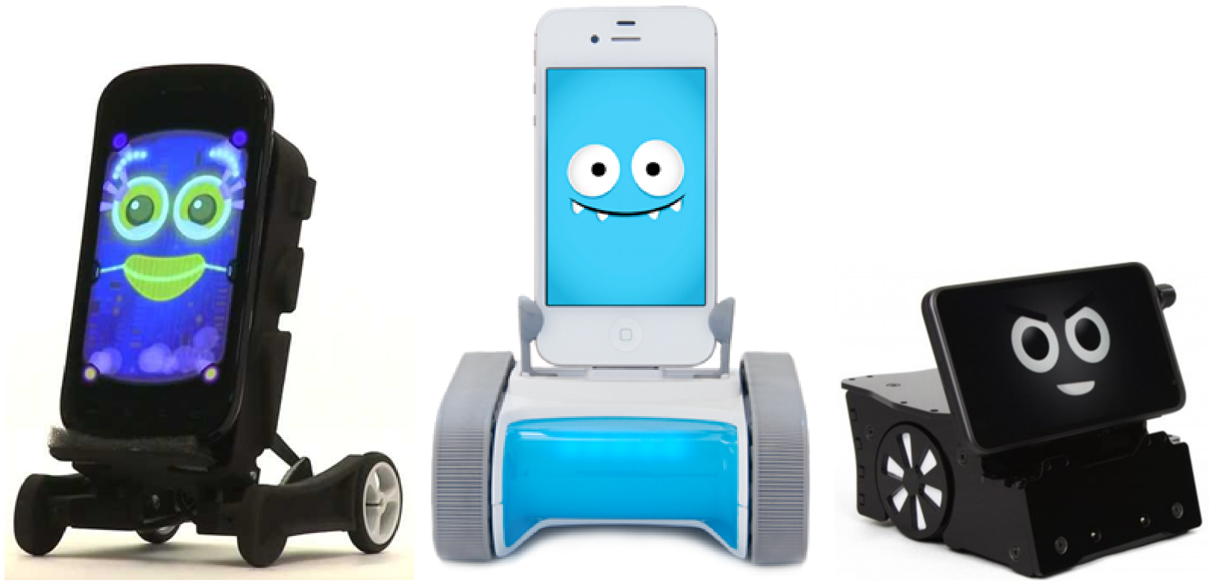


Figure 3.10: Smartphone Robots (Oddwerx (2012); Roméo (2014); SmartBot (2013))

Conclusion

Smartphone Robots presents a nice application to outsource the brain to remote devices. In contrast to the other projects, this approach is only useful for robots whose hardware components do not require much computation power. Therefore, this idea is not practicable for robots like the PR2, TurtleBot or Nao.

3.5.9 3D CAD browser

The 3D CAD Browser is an online 3D model exchange platform for graphic designers and CAD engineers. With the aid of this cloud service users are able to upload as well as to download 3D models in different formats. It presents an approach where users can upload a 3D model in a specific format and the cloud service translates this object format into another one.

Conclusion

Related to the topic Cloud Robotics this project takes relevance because the cloud service undertakes the challenge to transform one 3D model into another format. According to the approach of plan sharing for robots, such a cloud service can be used to provide uploading as well as downloading mechanisms of plans for different types of robots.

3.6 Objectives and requirements

Although the research area of Cloud Robotics is just at the beginning, the analysis in 3.5 has shown many different applications for robots. The related works have given a detailed insight into some current projects. These works can be mainly divided into the two application areas of knowledge sharing for robots and outsourcing computational intensive tasks via cloud computing concepts. Section 3.4 has already presented the common challenges in this research area. This section aims to concrete the targets of the master thesis. It demonstrates some use cases to deduce requirements and objectives. Besides, it will be used to define a concrete research hypothesis and to delineate this work from other projects.
Firstly, this thesis is focused on knowledge sharing among heterogeneous robots. There are different communication models like the Peer-to-Peer architecture and the blackboard architecture. Both patterns support a solution of the knowledge sharing problem for robots. According to Nii (1986) the blackboard architecture pattern is used to solve typical AI-problems which do not provide a unique solution. Generally, the blackboard describes a shared repository of problems, partial solutions, suggestions, and contributed information. Autonomous entities like an agent or a robot communicate with the blackboard as knowledge source by selecting a problem, solving it with its expertise and putting the proposed solution back to the board. Related to this research area, the blackboard architecture designs a central point of communication where robots are able to share information among each other as shown in subsections 3.5.1 and 3.5.4.
The Peer-to-Peer architecture provides a direct communication between the robots. Both concepts have advantages as well as disadvantages. On the one hand subsection 3.5.2 has shown, that P2P systems overcome several security methods like firewalls, routers and NAT‘s. In contrast to scenario 3.1, a robot sends requests to other robots directly to ask if they have experiences for a specific job. On the other hand the Peer-to-Peer approach requires that, if robot A wants to download knowledge which is generated by the source robot C, C has to be online. In the blackboard architecture, the robots interact indirect with each other. So, they do not care about the knowledge source.
This master thesis aims to implement the knowledge sharing approach through an indirect communication where all participating robots do not know each other as well as do not take care about the availability of the source robot. Those are the reasons why this work uses the blackboard architecture. As this master thesis uses cloud computing technologies to tackle the problem of knowledge sharing for robots, a cloud service operates as representative for a blackboard. Figure 3.11 provides a sketch of knowledge sharing among different robot platforms with the aid of a cloud service.

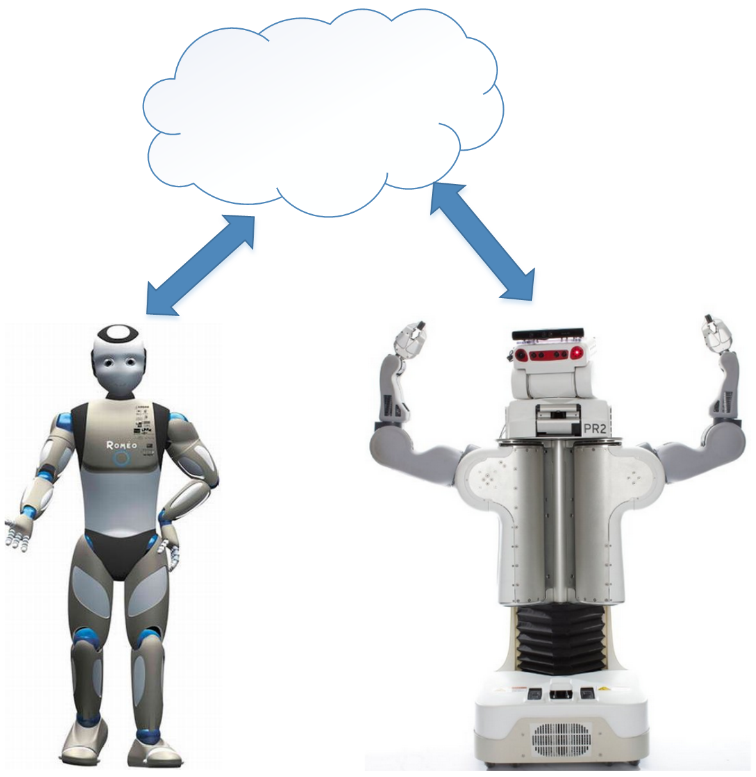


Figure 3.11: Knowledge sharing among heterogeneous robots via cloud services (Roméo (2014); PR2 (2013))

The next question to be answered is, which kind of knowledge should be exchanged between the robots. Compared to Prasad (2013) and 3.5.4 which are focused on multi-robot map data merging in Cloud Robotics systems, this thesis does not investigate the challenge of how heterogeneous robots can share information about environments among each other. Besides, it does not use robots to create 3D models and send this information to the cloud for grasping analysis like Kehoe et al. (2013). Because of the fact that the considered robots are equipped with enough computation power, this thesis does not investigate the approach of outsourcing robotic tasks like face detection or speech recognition as in 3.5.1. Knowledge generation from the WWW by translating abstract instructions from websites to robot-commands as presented in Tenorth (2011) will not be discussed in this master thesis. Instead of working on these topics, this work permits robots to exchange abstract plans among each other over a cloud service which is accessible via the World Wide Web. Since the communication will be carried via WWW-Protocols, it will not be assumed that the robots have real-time requirements. These abstract plans are defined as sequences of actions and are stored in the cloud service. In contrast to the Monkey and Banana problem from section 3.4, this work assumes that the instructions which will be shared between the robots, are already in the right sequence.

3.6.1 Use cases

This master thesis applies the approach of knowledge sharing for robots via cloud services in a scenario where two heterogeneous robots and one human participate. The scenario is divided into the upload use case and the download use case. The target is to show how an inexperienced robot is able to solve an unknown task by processing knowledge which is generated by another robot. The following use case diagram shows that the experienced robot A uploads its high-level information about how to bring the remote control back to the instructor to a cloud service. Thereby, abstract plans are stored in a database which is globally accessible via the internet. The scenario shows that when an experienced robot uploads its new generated knowledge, this information has to be transformed to an abstract format.

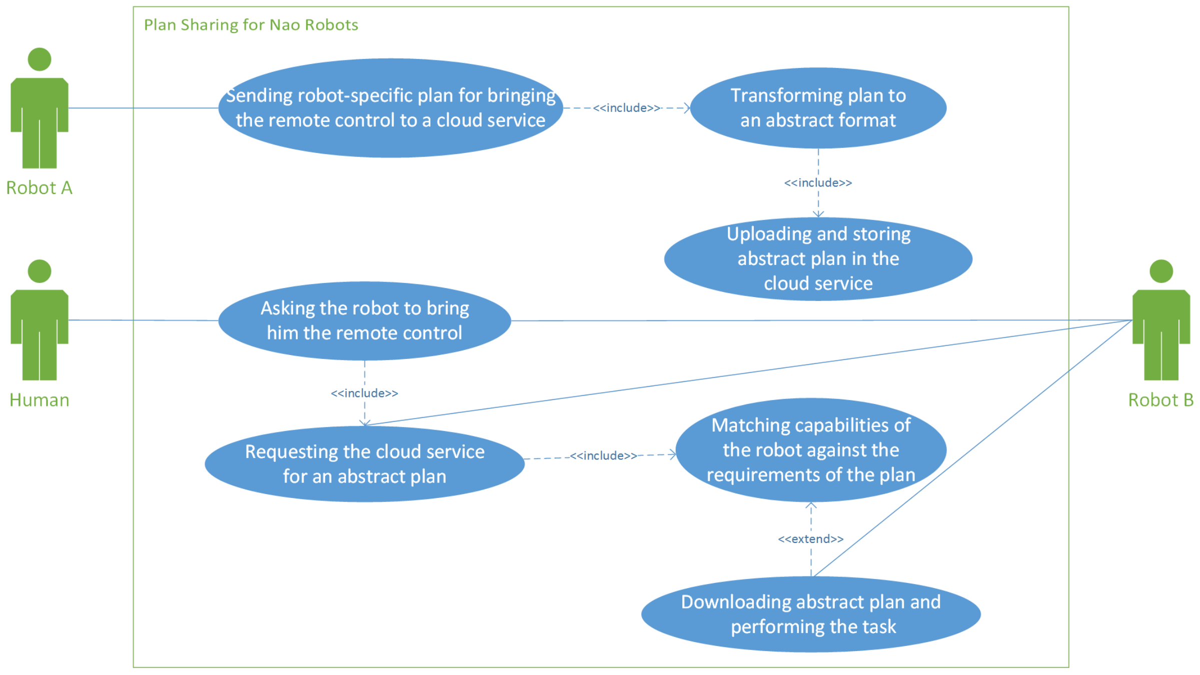

Plan Sharing for Nao Robots
Robot A
Human
Robot B
Sending robot-specific plan for bringing the remote control to a cloud service
<<include>>
Transforming plan to an abstract format
<<include>>
Uploading and storing abstract plan in the cloud service
Asking the robot to bring him the remote control
<<include>>
Requesting the cloud service for an abstract plan
<<include>>
Matching capabilities of the robot against the requirements of the plan
<<extend>>
Downloading abstract plan and performing the task

Figure 3.12: Use cases - Nao in the cloud

The use cases have shown that some software is required to implement the communication between two robots and the cloud service. This thesis organizes these software components inside a robot-specific interface which permits the robot to communicate with a cloud service. Note that every robot which aims to participate in a plan sharing system has to provide such kind of interface.

3.6.2 Functional requirements

The interface to provide a plan sharing between robots via cloud services must meet the following conditions:

1. The interface should be easy portable to further robot platforms.
2. Robots are able to send requests for robotic plans. The interface receives a plan request and ensures to find a solution.
3. The interface translates an abstract plan to the robot-specific execution language.
4. In contrast to the RobotShare project presented in subsection 3.5.3 this interface facilitates robots to execute the requested knowledge.
5. The robots should be able to receive plans which are generated by heterogeneous robot platforms: The difference to the Nao Application Store in 3.5.7 is that the robots can download programs which have not been produced by the same robot type. So, the provided design in chapter 4 opens the way for robots to process plans and knowledge which are uploaded by heterogeneous robots, not only by a specific one.
6. The interface is able to receive a robot-specific plan and transforms these instructions to an abstract program. After the translation process, the high-level plan will be uploaded to a cloud service.

3.6.3 Conclusion

The target of this master thesis deals with the implementation of a plan sharing system for heterogeneous robots. It connects robots to a cloud service via an interface to exchange abstract plans. The research hypothesis assumes that Nao robots are able to transform and execute high-level plans from a cloud service. This includes the challenge of how it is possible to control a robot on an abstract level without thinking about low-level tasks like path planning. Besides, the master thesis introduces another way to give robots the possibility to share information among each other. It includes the implementation of an interface for robots as well as the execution of a scenario in which robots download and execute information from the cloud service.

Chapter 4

Design

Chapter 3 has introduced the topic of Cloud Robotics. It presented a general specification and use cases for a plan sharing system permitting heterogeneous robots to upload new acquired skills to a cloud service as well as to download information. Furthermore, the analysis has shown that robots need an interface to share executable abstract plans among each other. As the Nao robot is the most applied humanoid robot related to education and research as well as it is the only robot which is provided by the labor of the University of Applied Science Hamburg, the Nao platform is selected to carry out a plan sharing system. Note that the following concepts do not take notice of all requirements of subsection 3.6.2. Since this work is focused on the downloading use case of subsection 3.6.1, an uploading mechanism will not be considered in the following design suggestions.
This chapter introduces design opportunities of an interface to provide a plan sharing among Nao robots via cloud services. Furthermore, it discusses several patterns to provide a communication between the Nao robot, the interface and the cloud service. Note that the following presented ideas and concepts are useful for other robots as well. It is more demonstrative to apply the following design approaches on one specific platform. The process how to port this concept to other robot platforms will be described in the outlook chapter 8.2. Finally, the conclusion shows the advantages as well as disadvantages of these approaches and argues why the chosen concept is implemented in the realization chapter.

4.1 Nao interface

This section designs an interface for the Nao robot platform to download and to execute abstract plans from a cloud service. It tries to satisfy the requirements 1-5 as listed in subsection 3.6.2. Diagram 4.1 describes the components of the interface providing a plan sharing system for the Nao platform.

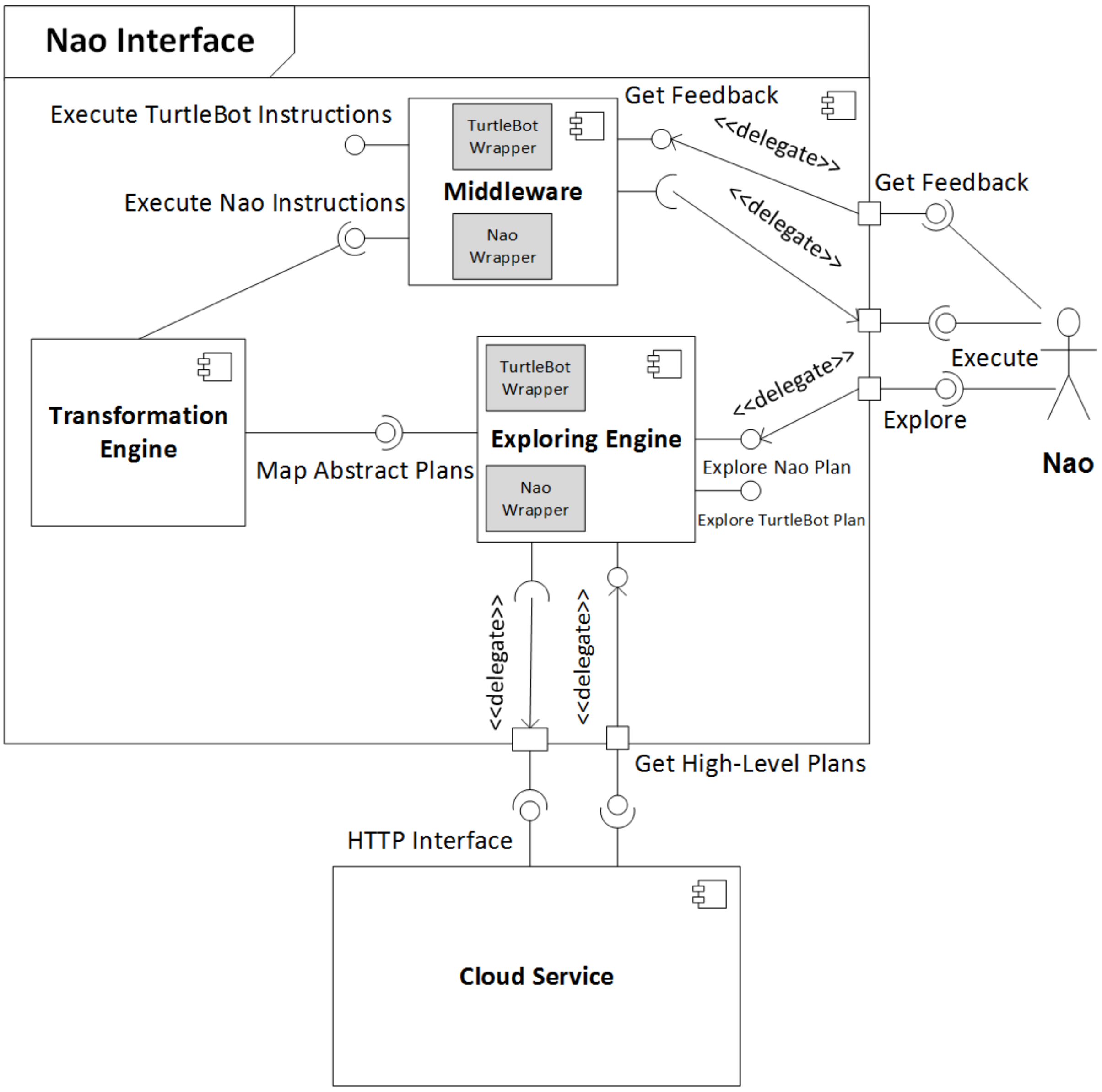


Figure 4.1: Component diagram of the Nao interface

The interface consists of the exploring engine, transformation engine and a middleware.

It provides five interaction points to communicate with the Nao robot as well as with the cloud service. The Nao robot sends a plan request to the exploring interaction point of the interface and receives Nao-specific instructions. During the execution, the interface receives feedback information from the robot. The cloud service sends abstract plans to the Nao interface by accessing the *Get High-Level Plans* interaction point. The <<delegate>> stereo type in the component diagram presents a forwarding procedure call to another component.

Middleware

Generally, a middleware represents a software layer to hide technical system-specific details from the application. In this thesis the middleware term will be used as abstraction layer, which encapsulates robot platform-depending challenges like navigating, object recognition, inverse kinematic or path planning algorithms for several robot platforms. Besides, the middleware permits controlling several robots in an abstract way by providing wrappers for different platforms. As presented in figure 4.1, the middleware contains wrappers for the Nao platform, TurtleBot[39] robot and much more. The middleware is used to send low-level commands to the Nao robot. As shown in diagram 4.1, the transformation engine accesses the *Execute Nao Instructions* socket to trigger the execution of the plan. This component provides a wrapper over the NaoQi SDK[40]. which is described in section 2.4. The wrapper forwards the commands to the Nao robot for execution by connecting to the NaoQi broker. This process runs onboard on the Nao robot. During the plan execution the middleware receives feedback from the Nao robot which can be used for failure analysis and logging. By using a middleware, it is possible to port this interface to other robot platforms.

Exploring Engine

As shown in figure 4.1 the exploring engine aids the Nao robot to search a special high-level plan description. Therefore, the interface delegates a plan request to this component to explore the cloud service via a HTTP interface. After downloading the abstract program, it matches the capabilities of the robot against the requirements of the high-level plan. To arrange this, it needs a semantic description of the robot. Subsection 3.4 already has presented some description formats like URDF[41] or SRDL[42]. Moreover, it translates the plan from the abstract format like OWL[43] into an executable even so generic language. Its important that the exploring engine and the cloud service are geared to each other.

[39]`http://www.turtlebot.com`
[40]Software Development Kit
[41]Unified Robot Description Format
[42]Semantic Robot Description Language
[43]Web Ontology Language

The following graphic 4.2 demonstrates the communication process between the exploring engine, transformation engine, the middleware and the Nao robot. Note that this component is not only accessible for Nao robts as presented in diagram 4.1. Since, for this component already exists a software solution, this chapter is not going into detail. Instead, this engine will be explained more in the realization chapter 5.

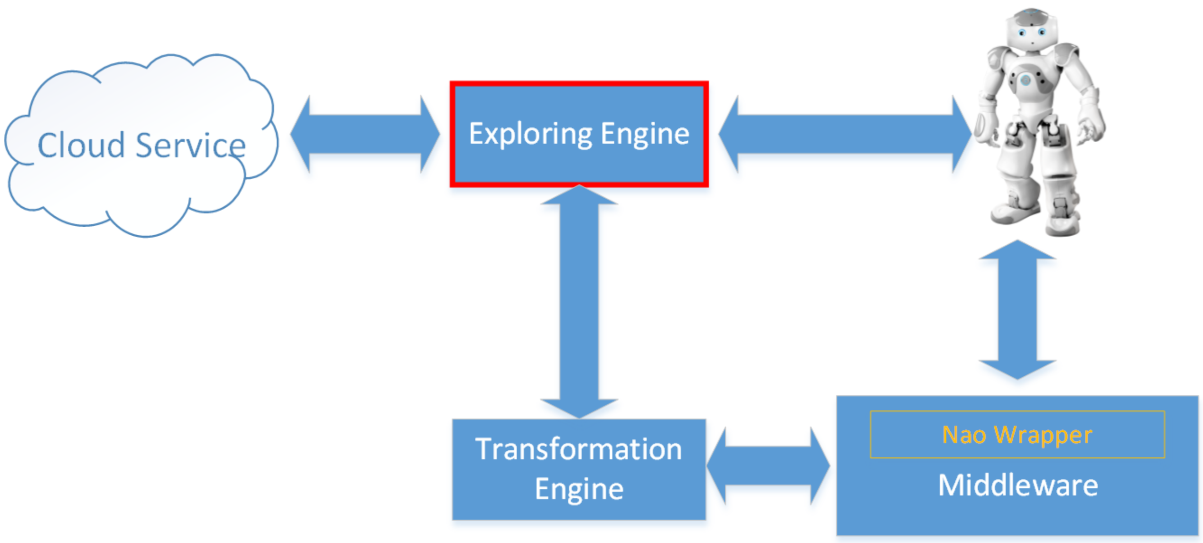


Figure 4.2: Communication process between all related components Nao (2011)

Transformation Engine

Graphic 4.1 shows that the exploring engine sends an executable abstract plan to the transformation engine by accessing the *Map Abstract Plans* socket. To perform these plans on the Nao platform, an abstraction layer is required to execute high-level commands. This component aims to execute robot-independent applications by mapping abstract commands to low-level Nao-specific instructions. The next graphic shows a more detailed view into the transformation engine. It introduced modules for manipulation, navigation, perception and text-to-speech. Each of these modules contains specific functions like moving an arm or walking to a particular position. Hence, high-level instructions of a plan like *Serve a drink* can call these functions to execute the Nao-specific low-level commands which are provided by the Nao wrapper of the middleware. Note that this work designs the modules presented in 4.3 for manipulation, navigation, perception and text-to-speech as so-called process modules. The concept of process modules will be used inside the transformation engine.

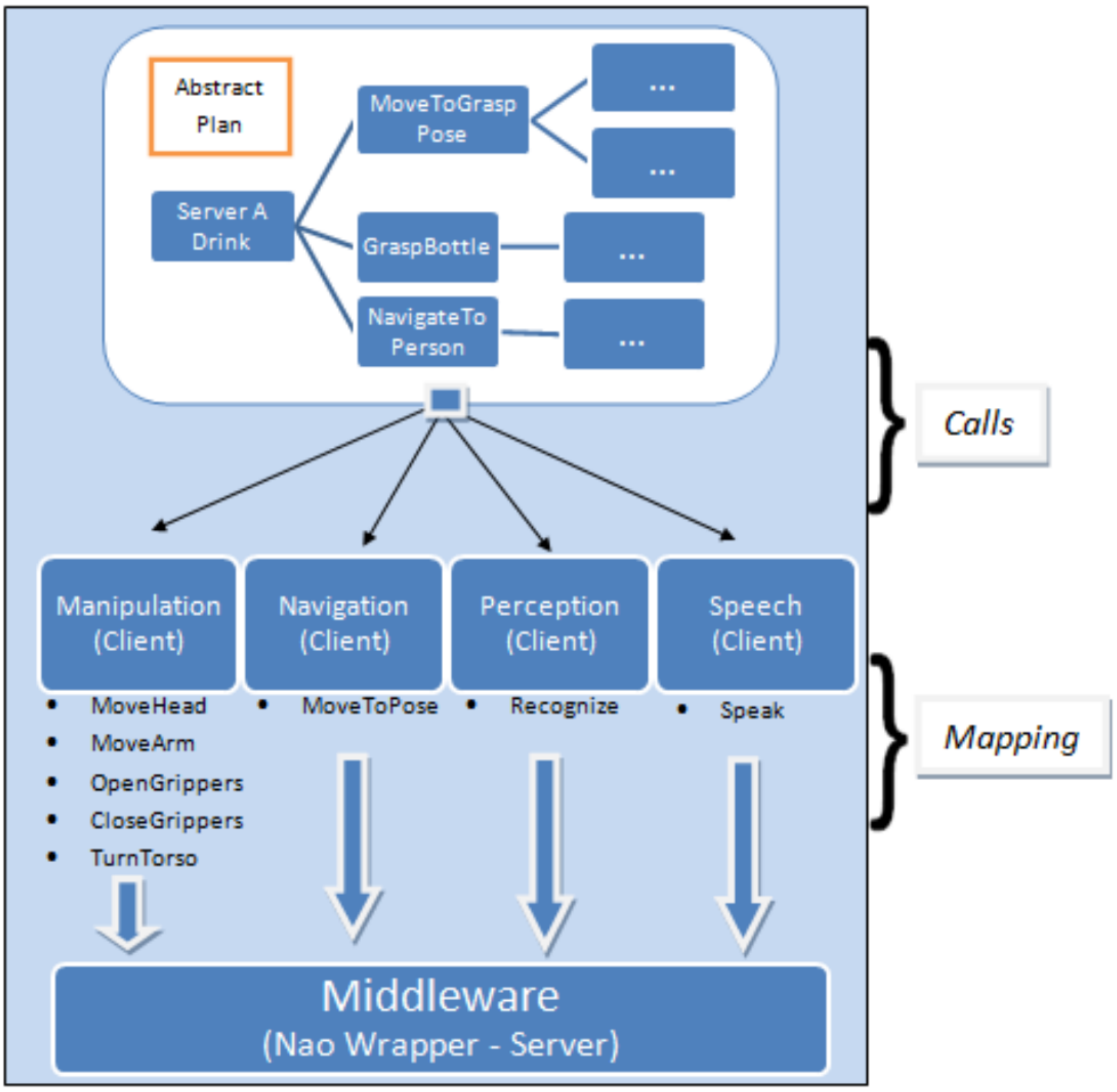


Figure 4.3: Transformation of high-level plans to Nao-depending instructions

Process modules

This transformation engine uses the idea of process modules which is introduced by Beetz et al. (2010) to transform high-level commands to robot-specific instructions. Generally, this concept aims to execute the same high level plan like bringing the remote control on heterogeneous robot platforms. A process module triggers the required hardware actions to control the robot if an abstract task should be executed.

The number of process modules depends on the robot and its application domain. Household robots have to provide at least process modules for manipulation, navigation, perception and speech processing. A high-level plan interacts with the robot by accessing only these modules. The main requirement of a process module is to map high-level commands to low-level instructions. As shown in figure 4.3, the process modules encapsulate low-level robot functions. In this work process modules communicate directly with the Nao wrapper as shown in figure 4.4.

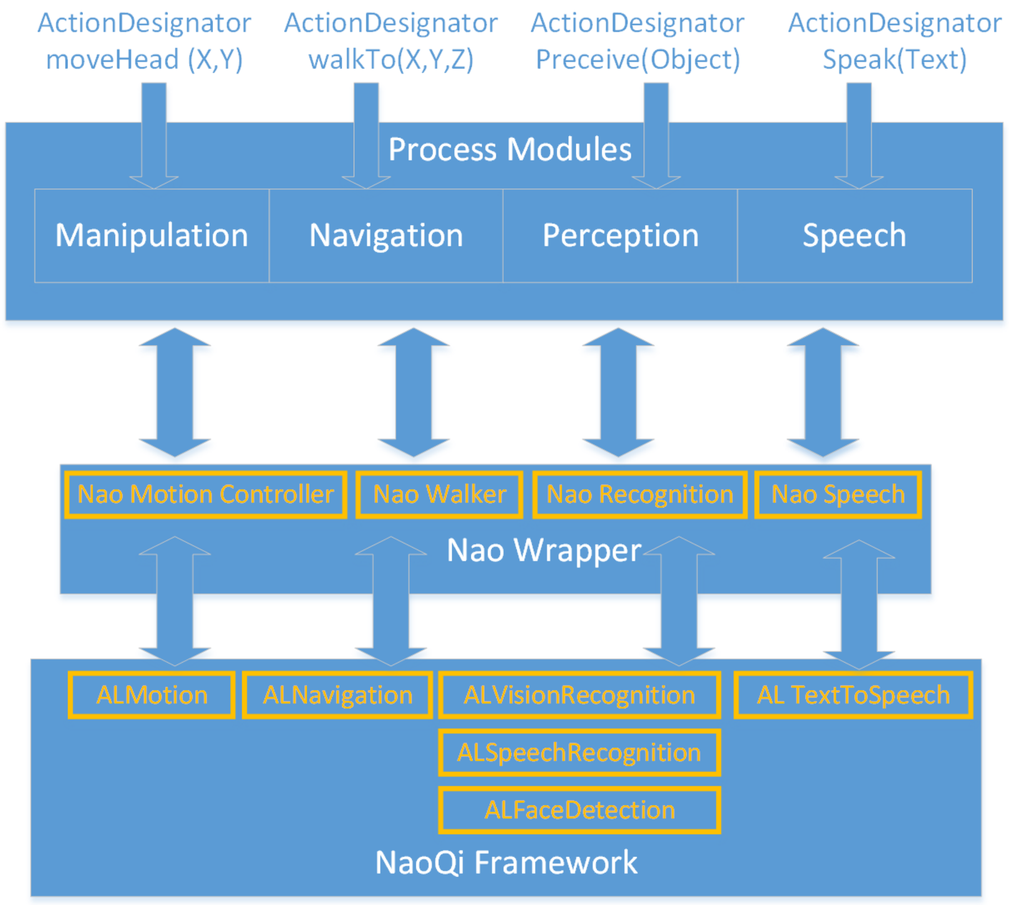


Figure 4.4: Interaction between process modules and Nao wrapper

The input always is a designator which contains the meta information from the environment like a target position to navigate the robot or the physical properties of an object which should be grasped via the manipulation interface. The designator concept already was introduced by Beetz and McDermott (1994) to supports dynamic plan parameterization.

Beetz and McDermott (1994) used this approach to describe objects in an abstract way within the Reactive Planning Language McDermott (1991). Generally, plans address objects by unique identifiers like “put object 0815 onto object 0955”. If the robot cannot find this object, it is not able to solve the task. This concept tackles the problem of how to handle objects for robots by describing objects in a high-level way. Through this concept the robot can search for a simliar kind of object which satisfies the same designator. Müller (2008) has extended this approach by adding the location designator to specify positions of objects. Action designators were established by Beetz et al. (2010). Their work contain all relevant information to execute a task successfully.
With the aid of the designator concept, robots are able to handle dynamic and flexible plans. The execution of a plan can be changed at runtime without restarting the plan. Generally, a process module receives an action designator from high-level plans as parameter and resolves these high-level descriptions to numeric control values. As presented in 4.4, it calls the functions of the Nao wrapper, which forward these resolved numeric values to the low-level componentes of the NaoQi SDK. The Nao wrapper includes modules to encapsulate Nao-specific low-level functionality. The process modules use three middleware-depending communication approaches which are described in 2.2 to call the low-level controllers of the Nao wrapper.
Each module operates as a server to receive commands from the process module clients as shown in 4.3. After executing the action, the process module creates events to influence the belief state of the robot. It includes all information about the execution environment which the robot assumes to be correct. This internal state also contains the set of known objects as well as its locations and properties. A successfully executed action of the navigation module creates an event like *moveTo(position)* which will be archived.

Moreover, the Nao interface represents a self-contained system which interacts with the Nao robot as well as with the cloud service. In the following, several design ideas are presented to place the Nao interface inside a plan sharing system.

4.2 Approaches of communication

This section shows several design approaches to create a communication between the Nao robot, the interface and the cloud service. These concepts allow Nao robots to download and execute abstract plans from a cloud service. There are different possibilities to place the interface. The gateway term will be used as access point for the Nao robot to connect itself to a cloud service. Generally, a gateway is a link between two computer programs or systems. It is used to provide a transformation among different communication protocols. A gateway acts as an interface between two programs or systems allowing them to share information among each other. Note that the following approaches are implementable for every type of robot platform, not only for the Nao robot.

4.2.1 Intelligent gateway

This approach shows a concept where Nao robots are able to download executable plans with the aid of a gateway, which represents an interface between the Nao robot and the cloud service. Nao robots communicate indirectly with the cloud service via accessing the gateway. The next figure shows a rough design of an intelligent gateway. Intelligent means, that the gateway represents the Nao-specific interface which is designed in section 4.1.

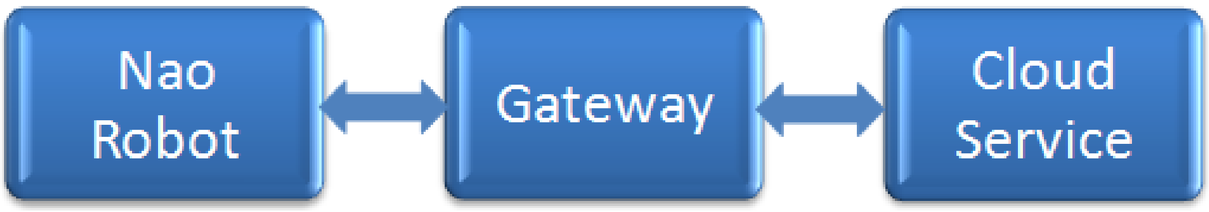


Figure 4.5: Sketch of an intelligent gateway

Components

This subsection describes the responsibilities of the components which are used to design a knowledge sharing for Nao robots via an intelligent gateway.

Nao robot

The Nao robot operates in a human living environment and represents a thin client which means that not much extra software has to run on the robot. Related to scenario 3.1, Nao only has to receive instructions by a human via speech recognition and delegate these instructions to the gateway to access the cloud service, looking forward to get a high-level plan. The Nao robots have to provide a NaoQi broker server process to communicate with the gateway over a wireless connection. So, Nao has to know the IP[44] address of the gateway. The Nao platform, including the architecture and NaoQi SDK is already described in section 2.3. On the robot itself are no more extra components required.

Gateway

The gateway represents an interface between the Nao robot and the cloud service. So, the gateway has to know the IP address of the NaoQi-Broker server as well as the URL of the

[44] Internet Protocol

cloud service. All intelligent software components of the Nao-specific interface are placed in the gateway. The gateway operates as server and delegates the received plan requests from the Nao robot to the cloud service. After exploring the cloud service and matching of Nao capabilities against the requirements of the high-level plan the gateway downloads the abstract commands. Through the Nao-specific interface which contains an abstraction layer for manipulation, navigation, perception and text-to-speech translation the gateway is able to execute the high-level instructions. During the execution of abstract plans, the gateway sends low-level commands to the NaoQi broker to perform the program. It is not important that the Nao robot and the gateway are connected in the same WLAN domain. But, the closer the gateway is placed to the Nao robot the better the performance.

Cloud service

The cloud service stores all required information like descriptions of plans and objects in an abstract definition format. These plans contain sequences of actions. The cloud service is accessible through the HTTP[45] interface and has to store all knowledge which is required to perform high-level plans. As mentioned in 3.4, a cloud service which is accessible via HTTP is usually called web service. So, a web service is intended as kind of cloud service to interact with the gateway. In this approach the web service has the responsibility to store robot programs in an abstract way which are preprocessed to be downloadable by robots. Note that this kind of robot plan is not Nao-specific. The web service in this approach represents a RESTful API. The web service provides the common HTTP request methods GET, POST, PUT and DELETE. Through these operations the gateway is able to send a plan request via the GET command to the web service. The GET instruction requests a resource like an abstract plan from the cloud service. Thereby, the web service does not change its status. This approach uses only the GET command because the design of the plan sharing system does not consider an upload of robot plans.

Communication

Diagram 4.6 demonstrates the communication between the human, the Nao robot and a cloud service by using the intelligent gateway approach. The sequence diagram shows the use case when a human requests the Nao robot to perform an unknown task, similar to the scenario 3.1. Firstly, the gateway activates the Nao-specific interface. With the aid of this interface it is possible to control the Nao in an abstract way. It presents an abstraction layer for manipulation, navigation, speech and perception. After that, it is possible to execute abstract instructions from the web service which stores all needed information to solve the task. So, the Nao robot can assume that no more knowledge has to be generated. Related to figure 3.2 instead of answering with „Sorry, I can‘t perform what you command me”, Nao connects

[45]Hypertext Transfer Protocol

itself to a web service by accessing the gateway. It is important that the Nao robot and the gateway are configured. The Nao robot sends a plan request to the gateway which forwards the request to the web service. If a plan is found in the database of the web service, the gateway receives it and maps the high-level plan to Nao-specific commands. This mapping is only possible through the Nao-specific interface. The gateway accesses the NaoQi broker server remotely and calls the particular low-level actuators of the Nao robot. During the execution of the task, the gateway receives feedback and status information.

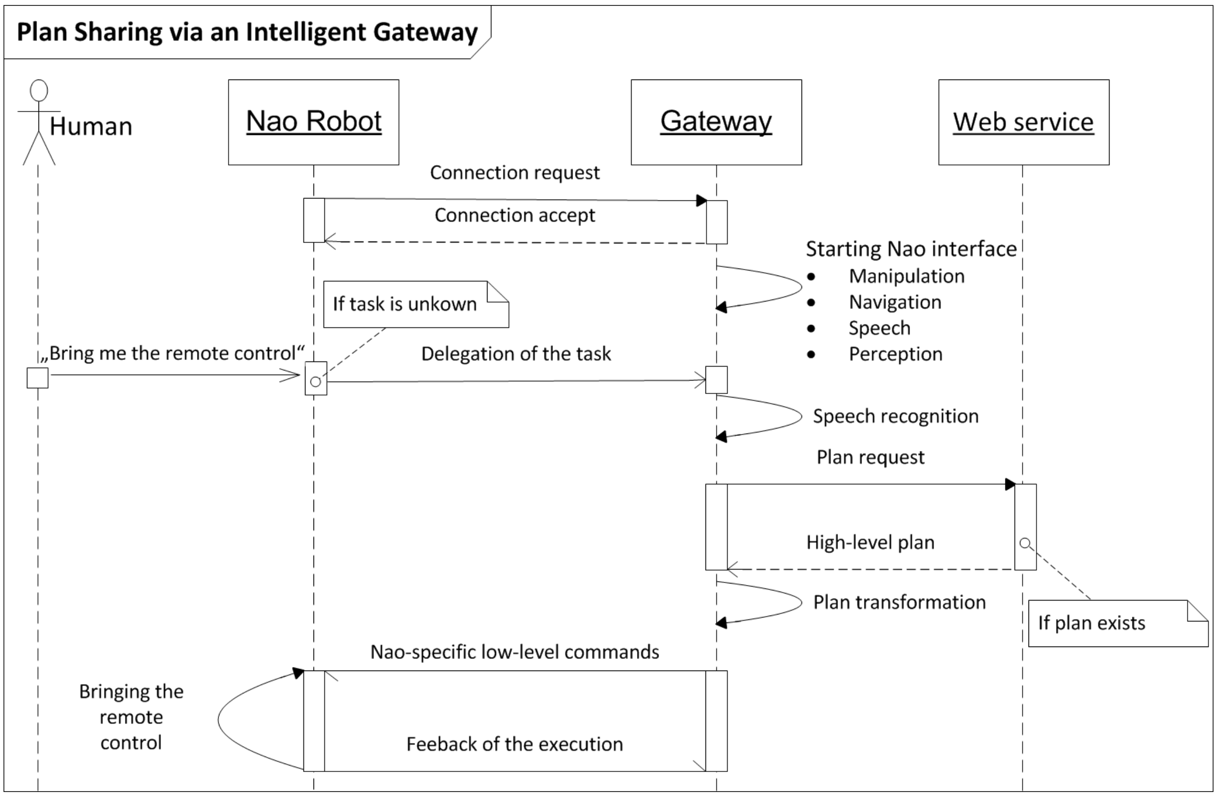


Figure 4.6: Communication among the Nao robot, the intelligent gateway and the web service

4.2.2 Transparent gateway

In this design concept the Nao robots are able to download executable plans by accessing a transparent gateway. Similar to the first design approach, the communication happens indirectly between the Nao robot and the cloud service by accessing the gateway. In contrast to the first concept this approach places the interface as part of the cloud service. Thus, the cloud service undertakes the challenge to provide all needed software components to implement a plan sharing among Nao robots.

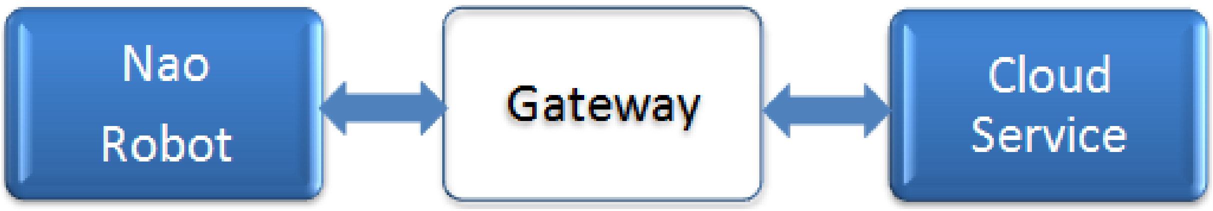


Figure 4.7: Sketch of a transparent gateway

Components

In the following the roles of the used components are described. It shows that this approach applies these components in a different way as in the first approach.

Nao robot

The Nao robot has the same role as in the first design approach. The Nao robot also does not provide software to translate abstract plans to low-level commands locally on the robot. Similar to the first concept, it sends a plan request to the gateway and receives robot-depending instructions from the gateway. After execution Nao is able to store the low-level commands to increase the knowledge in its database.

Gateway

In this approach, the gateway is designed as forwarding component. Transparency means that the gateway does not provide software components to support a plan sharing among the Nao robots. It only forwards the plan request from the Nao robot to the cloud service as well as delegates Nao-depending commands back to the robot. Different to the first approach, this gateway does not carry out a mapping of high-level plans to Nao-specific low-level instructions and it does not provide a search engine to explore the cloud service for a specific robot plan.

Cloud service

This component has to provide the whole functionality for plan sharing among Nao robots. Equally to the first concept, the cloud service is accessible via an HTTP interface as web service and receives plan requests from the gateway. This design concept is similar to the project 3.5.9 where users are able to upload 3D models in several different formats to the web service which provides a mechanism to translate a user-specific 3D model into its own internal description format. The web service of this project has a similar responsibility as this web service. This approach permits every robot to upload a robot plan in its own language.

The service receives the platform-specific plan and translates the program to its generic description language. If a heterogeneous robot wants to download the plan, the cloud service transforms the generic plan to the language of the platform. The robot-depending commands will be sent back to the gateway which delegates the instructions to the robot.

Communication

The following sequence diagram shows the interaction between the Nao robot, the transparent gateway and the cloud service. As in the last approach the Nao robot and the cloud service do not known each other. Nao sends a plan request to the gateway which forwards the request to the cloud service. The cloud service starts the interfaces of all considered robots. After exploring the database the service maps high-level commands to low-level instructions by using the platform-specific translation method. Furthermore, these resolved instructions will be sent to the gateway which delegates the commands to the Nao for execution.

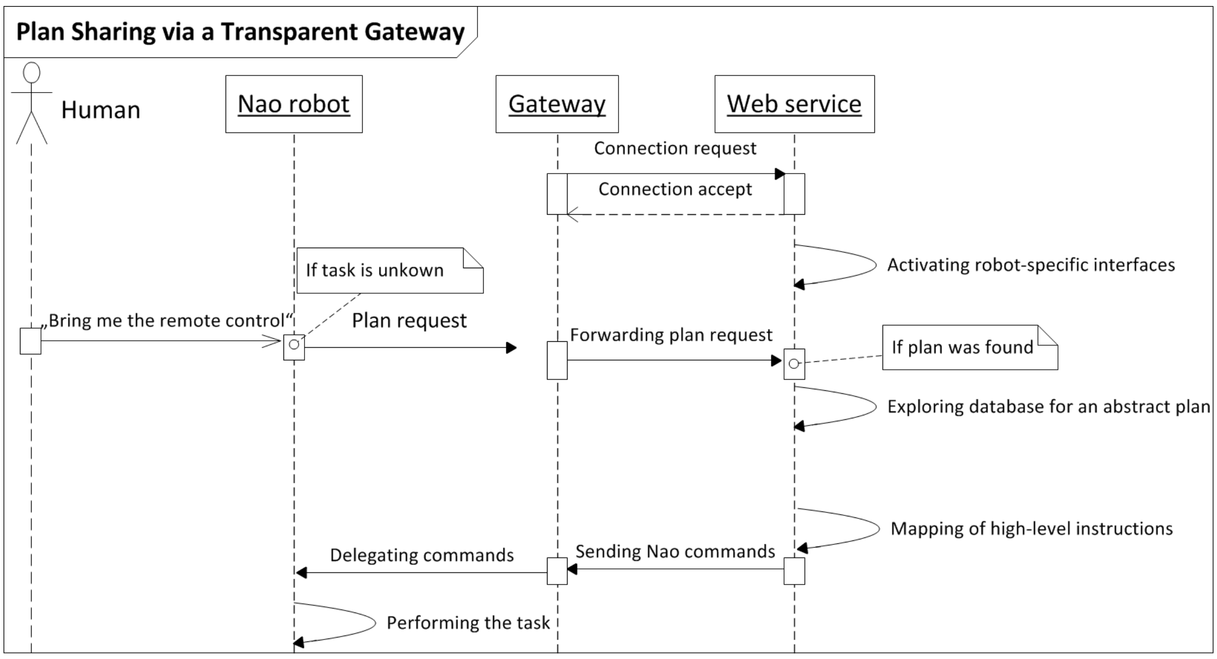


Figure 4.8: Sequence diagram for the transparent gateway approach

4.2.3 Without Gateway

This approach aims to provide a plan sharing for Nao robots with a direct connection to the cloud service without a gateway. The main difference to both of the other concepts is that the interface for interaction with the cloud service is placed directly on the robot.

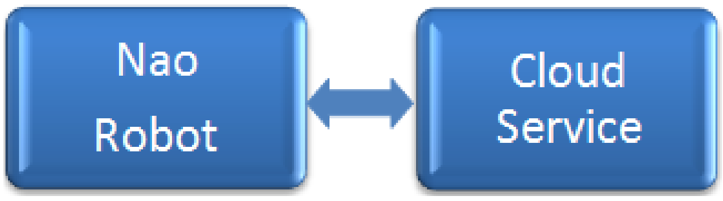


Figure 4.9: Sketch of a plan sharing system without a gateway

Components

In the following the responsibilities of the Nao robot and the cloud service will described.

Nao robot

This approach designs the robot as fat client where all needed software for plan transformation and downloading are installed locally on the robot. The Nao robot has to know the URL address of the cloud service. Nao explores directly the cloud service for a specific plan. If the Nao finds a satisfying plan, it downloads the program. As this program is defined in an abstract format the Nao robot has to map these high-level commands to its own platform-depending language. After the translation process Nao is able to execute the abstract plan.

Cloud service

As well as in the both last approaches the cloud service represents a web service and provides an HTTP interface to download a plan. Besides, these programs are stored in an abstract description language.

Communication

As presented in the next sequence diagram, the Nao robot and the cloud service communicate without a gateway. When the Nao receives instructions to solve an unknown task it explores the web service directly for a special robot program. After finding a satisfying plan the cloud sends a high-level plan back to the robot. As already mentioned the Nao robot has to map the high-level commands to its own platform-depending NaoQi SDK. If the robot has finished translating the whole abstract plan the Nao is able to execute the plan.

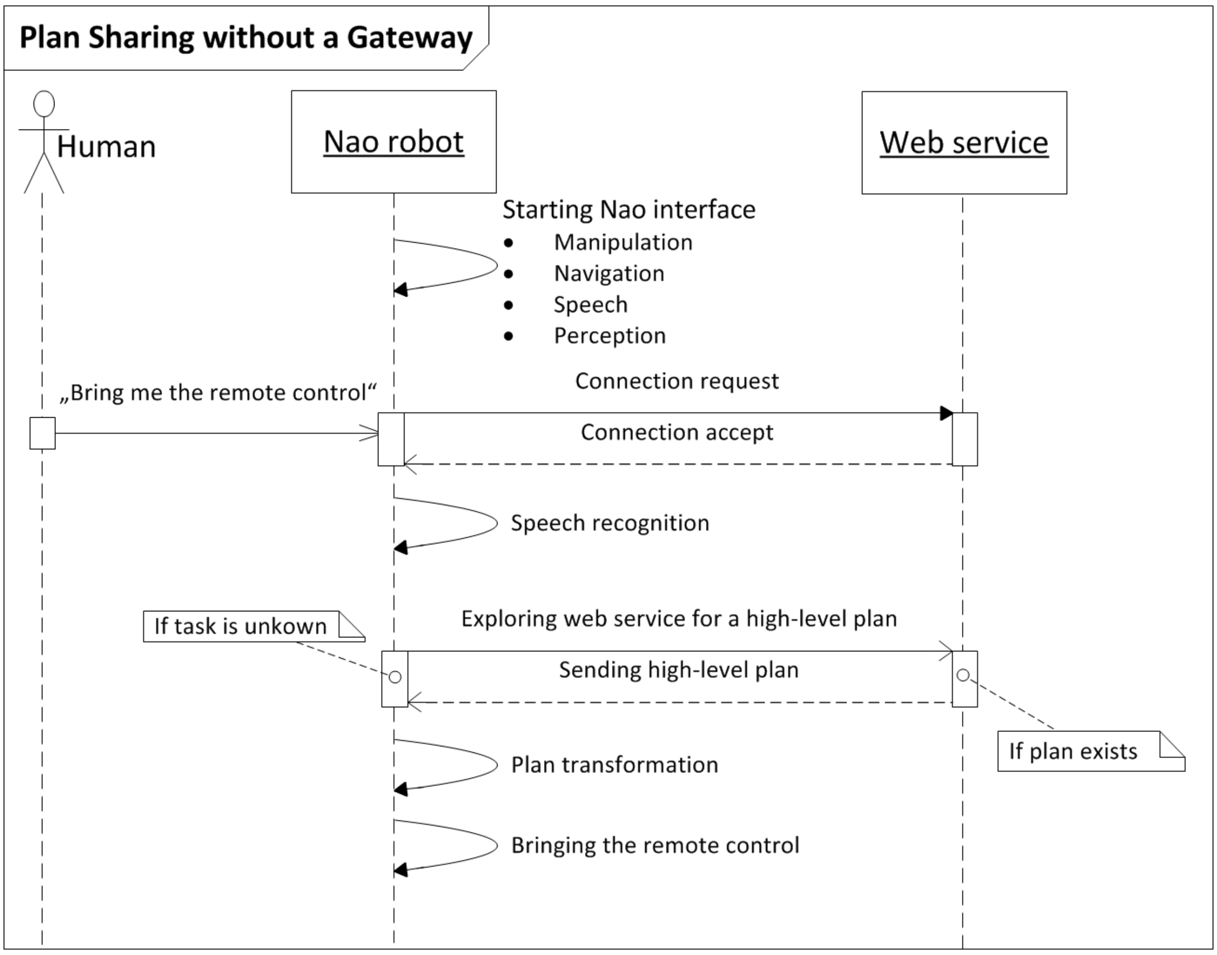


Figure 4.10: Communication between the Nao robot and the web service

4.2.4 Conclusion

The previously presented approaches to organize a plan sharing system for Nao robots have different advantages as well as disadvantages. On the one hand by using a gateway as interaction bridge the Nao robots do not need to be preconfigured which means the robot does not have to provide components to connect itself with a cloud service. On the other hand both of these ideas bring along the disadvantage of the lower performance, compared to designing a plan sharing system without a gateway. The crucial reason to place the Nao-specific interface not directly on the robot is that the developer has no chance to intervene in case of runtime errors. Such critical behaviour is very risky because the Nao robot could be damaged. So, it is recommended to place as few software as possible onboard. The transparent gateway would be useful if not much message traffic occurs between the Nao robot and the cloud service. Furthermore, designing an intelligent gateway makes sense because of the enhanced exchangeability. Through this concept, it is easier to exchange the cloud service or the robot than in both of other approaches. This thesis carries out the design of a knowledge sharing system for Nao robots with the aid of an intelligent gateway. It implements the Nao-specific interface from section 4.1 to a cloud service to download and transforms high-level plans.

Chapter 5

Realization

This chapter implements a Nao-specific interface which is designed in section 4.1. This interface permits Nao robots to download and execute abstract plans from a cloud service. Moreover, it creates a plan sharing system by using the design approach of an intelligent gateway as described in 4.2.1. This design concept allows an indirect communication between the Nao robots and a web service via an intelligent gateway that provides all intelligent software to create a plan sharing among Nao robots. This realization uses the following components which are already implemented by different research groups.

- ***RoboEarth*** Waibel et al. (2011): According to 3.5.1, the analysis revealed that the RoboEarth web service is the best option because of its huge dissemination, scalability and popularity as an internet based storage for robot-specific knowledge. Related to 4.2.1, RoboEarth is selected as cloud service inside the plan sharing system.

- ***KnowRob*** Beetz and Tenorth (2009): This knowledge processing system represents the exploring engine designed in section 4.1 and is used to explore the web service for abstract robot plans.

- ***CRAM Plan Language*** Beetz et al. (2010): This language provides domain specific functions to describe abstract control programs for robots, which are used to create high-level plans.

- ***ROS***[46] Quigley et al. (2009b): This framework fulfils the part of the middleware described in subsection 4.1 to send low-level commands to the Nao robot. ROS was chosen as preferred abstraction layer because of its popularity as well as its support of many different robot platforms.

- ***Nao-ROS-stack***Hornung (2009): Provides a wrapper over the NaoQI SDK[47] which is integrated in to the ROS middleware.

[46]Robot Operating System
[47]Software Developement Kit

3.5.1 also showed that every robot has to provide an own mapping of abstract commands to low-level instructions to execute RoboEarth-stored knowledge. So far, Nao robots have not been able to process robot plans from the RoboEarth database. To counter this challenge, the ***Transformation engine*** is created during this master thesis to map high-level plans to Nao-specific commands. Therefore, the concept of process modules is implemented which represents the core of the transformation engine. This work has implemented process modules to manipulate and navigate the Nao robot as well as to process text to speech. Since the Nao robot and the web service RobotEarth are already presented in sections 2.3 and 3.5.1, this chapter is focused on introducing the components and middlewares used for the implementation of the Nao-specific interface.

5.1 Interface

The Nao interface is based on recent projects Johannßen (2013a) and Johannßen (2013b). This section shows a possible software solution to satisfy the design requirements of the interface between the Nao-ROS wrapper and the RoboEarth web service as presented in section 4.1. Figure 4.2 has shown that the interface includes several layers to abstract from the Nao-specific hardware components. A middleware and a transformation engine will be used to control the Nao robot in an abstract way as well as to transform high-level commands to Nao-depending instructions. It also provides an exploring engine to browse the RoboEarth web service for a requested abstract plan. The interface mainly contains components for exploring and transforming high-level plans running on top of the ROS[48] middleware. This section demonstrates how these components are implemented.

5.1.1 ROS middleware

As described in the requirements of section 3.6, the interface has to provide the possibility for Nao robots to process knowledge, which is generated by heterogeneous robots. To tackle this challenge, robotic-specific middlewares can be used as abstraction layer to execute high-level instructions that are produced by heterogeneous robots with different capabilities and hardware resources. An abstraction layer over the Nao specific hardware is needed to satisfy this requirement. With the aid of ROS, it is possible to command the robot to walk from place A to place B without thinking about which kind of walking algorithm will be used. As mentioned in section 2.2, this framework is extendable with modules, called stacks. There are stacks for many robots like Nao, TurtleBot, Husky or PR2 as presented in `http://wiki.ros.org/Robots`. Furthermore, ROS has released several distributions like Fuerte[49],

[48]Robot Operating System

[49]`http://wiki.ros.org/fuerte`

Electric[50], Groovy[51] and Hydro[52]. This master thesis is focused on using Fuerte because this version is the most stable one. For more detailed information about the ROS framework see section 2.2.

5.1.2 Exploring engine

This component fulfills the responsibility to search a satisfying abstract plan inside the RoboEarth storage as well as to translate this plan to an executable format. To implement this functionality the KnowRob ROS-stack[53] is used, which is integrated into the ROS middleware. More information about the KnowRob framework can be found in subsection 3.5.1. This component uses the ROS-package *re_comm*[54] which is part of the RoboEarth ROS-stack to communicate with the web service. The following two subsections demonstrate how the KnowRob framework is used.

Download abstract plan

With the aid of the KnowRob stack, robots are able to request the RoboEarth database for a plan like *grasping a bottle*. The following algorithm 5.1 shows how KnowRob searches a high-level plan.

Algorithm 5.1 Downloading OWL-description of a high-level plan

```
1: procedure DOWNLOADPLAN($Command$)          ▷ Command: Name of the plan as string
2:     $PlanURL \leftarrow searchURL(Command)$
3:     $OWLplan \leftarrow downloadOWLplan(PlanURL)$   ▷ Downloads the OWL-description of
   the plan
4:     $ActionClass \leftarrow getActionClass(OWLplan)$
5:     return $ActionClass$                    ▷ Returns the OWL-class of the plan
6: end procedure
```

Furthermore, it checks if the robot already knows all needed object models to perform the task successfully. If not KnowRob downloads the missing models defined in OWL format as presented in the algorithm 5.2.

[50] http://wiki.ros.org/electric
[51] http://wiki.ros.org/groovy
[52] http://wiki.ros.org/hydro
[53] http://wiki.ros.org/knowrob
[54] *http://wiki.ros.org/re_comm*

Algorithm 5.2 Downloading missing object models

```
1: procedure DownloadMissingComponents($ActionClass, Robot$)    ▷ (ActionClass: OWL-class of the plan; Robot: Semantic description of the robot)
2:     $ListofMissingObjectModels[URL] \leftarrow searchMissingObjectModels(Name, Robot)$
3:     $ListOfModels[OWL] \leftarrow downloadMissingModels(ListMissingObjectModels)$
4: end procedure
```

Generate executable plan

After downloading the requested high-level plan, the next algorithm translates the OWL-based plan description to an executable format which can be processed on the Nao platform.

Algorithm 5.3 Translating OWL-Plan to CPL

```
1: procedure GenerateCPLplan($Plan$)    ▷ Plan: OWL-class description
2:     $CPLplan \leftarrow exportPlanToCPL(Plan)$
3:     return $CPLplan$
4: end procedure
```

Putting all together

The following algorithm presents the functionality of the exploring engine including the just described algorithms 5.1, 5.2 and 5.3.

Algorithm 5.4 Exploring engine

```
1: procedure ExploringEngine($Command, Robot$)    ▷ (Command: Name of the plan; Robot: Sematic description of the robot)
2:     $ActionClass \leftarrow DownloadPlan(Command)$
3:     $DownloadMissingComponents(ActionClass, Robot)$
4:     $CPLplan \leftarrow GenerateCPLplan(ActionClass)$
5:     return $CPLplan$
6: end procedure
```

5.1.3 Nao platform

The implementation of the transformation engine is based on using the NaoQi SDK 1.14.5 Aldebaran-Robotics (2013a).

5.1.4 Transformation engine

As in section 4.1 designed, this part of the interface carries out a mapping of high-level plans to Nao-specific low-level commands. To solve this problem the following ROS-stacks and concepts are used.

5.1.4.1 Nao-ROS-stack

This stack provides a ROS-specific abstraction layer over the NaoQi SDK. It is developed by the University of Freiburg Hornung (2009) and wraps the NaoQi API[55] in the high-level programming language Python to control the Nao robot in an abstract way. The current version of the Nao-Stack is compatible with the NaoQI version 1.12 or newer. The Nao-Stack provides access to sensors, odometry, cameras, teleoperation and speech recognition via ROS nodes. Besides, it is possible to manipulate the joints of the Nao without thinking about low-level problems like inverse kinematic or self collision. Usually, stacks are structured into several packages whereby each of them provides support for one specific task like navigation or face detection. The Nao-Stack mainly includes the following packages:

Package	***Description***
nao_robot	Provides basic functionality to access odometry, cameras, sensors and joints
humanoid_msg	Includes some basic services for humanoid robot navigation
nao_extra	Contains tools to run the Nao robot remotely on the PC. It provides mechanisms to monitor the joints and odometry remotely.

Table 5.1: Packages of the Nao-stack

The Nao-ROS-stack provides support for developers to program applications for the Nao platform more easily. As explained in section 2.2, ROS nodes represent processes to perform tasks. They communicate with each other by sending messages inside the ROS network. The Nao-ROS-stack contains several nodes for particular low-level tasks like solving path planning or manipulating the motion of the Nao robot. This stack basically provides the following ROS-nodes:

[55] Application Programming Interface

Node	***Description***
nao_controller	Contains basic functionality to access odometry, cameras, sensors and joints
nao_speech	Provides access to translate text to speech
nao_path_follower	Represents the navigation module
nao_walker	Includes some basic services for humanoid robot navigation
nao_sensors	Contains tools to run the Nao robot remotely on the PC. It provides mechanisms to monitor the joints and odometry remotely.
nao_tactile	Publishes data of Nao‘s tactile sensor
nao_camera	Wrapps the NaoQI library ALCamera
nao_leds	Provides controlling of Nao‘s LEDs

Table 5.2: Nodes of the Nao-ROS-stacks

5.1.4.2 CRAM Plan Language

As already mentioned in section 3.4, the Cognitive Robotic Abstract Machine Plan Language extends the functional programming language Common Lisp. This language is based on the concepts of the Reactive Plan Language (RPL) by McDermott (1991). The ROS-Stack of the CRAM Plan Language (CPL) so-called *cram_core*[56] is created by Moesenlechner (2009) and includes these packages:

Package	***Description***
cram_language	Represents a Common Lisp extension to program abstract robot plans
cram_reasoning	Includes a full-featured Prolog interpreter which is implemented in Common Lisp and provides algorithms for pattern recognition
cram_designator	Provides meta information about objects and actions
cram_process_modules	Represents the interface to a specific robot
cram_utilities	Utility methods and functions to support implementing high-level robot programs

Table 5.3: CRAM-Core packages

CPL supports creating ROS-nodes which can communicate with other nodes of the Nao-ROS-stack by sending ROS messages.

[56]`http://wiki.ros.org/cram_core`

5.1.4.3 Process modules

Section 4.1 has already introduced the concept of process modules to encapsulate the robot-specific low-level components. As presented in table 5.3, this concept is supported by the ROS middleware as part of the *cram_core* ROS-Stack. The package *cram_process_modules*[57] provides libraries to define process modules in CPL. This master thesis has developed process modules to manipulate and navigate the Nao robot as well as to process text to speech. As presented in section 4.1, process modules map high-level commands to low-level instructions. A process module receives a designator as parameter from abstract plans and resolves these high-level descriptions to numeric control values. Furthermore, these resolved values will be forwarded to the specialized low-level of the Nao-stack. The following algorithms are implemented using the CRAM Plan Language. As presented in listing 5.5, each process module operates as autonomous thread. The function *StartProcessModules()* will be triggered by the transformation engine.

Algorithm 5.5 Main function of the transformation engine

```
1: procedure StartProcessModules()
2:     Parallel
3:         $runThread(ManipulationProcessModule)$
4:         $runThread(NavigationProcessModule)$
5:         $runThread(SpeechProcessModule)$
6:     EndParallel
7: end procedure
```

A process module always receives an action designator to perform a special task. It resolves the designator by extracting key-value pairs like (left, open) for the grasping action through prolog-rules. Afterwards, the process module calls the particular action with the resolved values as parameters. Pseudocode 5.6 shows the functionality of a process module.

Algorithm 5.6 Functionality of a process module

```
Require: $Designator \neq null$
1: loop
2:     procedure ProcessModule($ActionDesignator$)    ▷ Designator: List[(Key,Value)]
3:         $ResolvedDesignator \leftarrow resolve(ActionDesignator)$    ▷ Mapping of High-Level
       descriptions to numeric values
4:         $callAction(ResolvedDesignator)$    ▷ Execution of the action
5:     end procedure
6: end loop
```

[57] `http://wiki.ros.org/cram_process_modules`

Manipulation process module

This interface should provide the following actions to manipulate the low-level components of the robot:

- Moving the arms to a target position
- Turning the head
- Opening and closing the grippers

The manipulation process module receives an action designator which describes what kind of action shall be executed as well as how this action should be performed. Moreover, the resolving function defines prolog rules to transform the meta information to numeric values. Such an action designator to open the gripper can be described like *(type: action(name:(to open) (side :left))*. The resolving mechanism only extracts the value of the side variable and forwards this value to the Nao-ROS wrapper to execute the actions.
As presented in the basics of 2.2, the ROS framework provides several approaches for interaction among nodes. To solve this challenge the ActionLib interface will be used which is part of the ROS middleware. As already mentioned in section 2.2, this pattern presents a typical communication between client and server.
The manipulation process module is using the nao_controller ROS-Node, which is part of the Nao-Wrapper. This node provides a Joint-Trajectory-Action-Server to control the Nao joints. The resolved meta information will be wrapped into a ROS-Action-Goal. Action-Client and Action-Server communicate via the ROS Action Protocol with each other, which specifies the messages: goal, result, feedback, cancel and status.
The *NaoActionClient* sends an Action-Goal to the Joint-Trajectory-Action-Server of the nao_controller. The nao_controller receives the goal and executes the requested task like moving the arm or opening the gripper. Note that the ActionLib calls are working asynchronously. This is well-founded, because many computations are long-running. Hence, it is advantageous to control another component of the robot while the Action-Server is calculating the result.

The following pseudocode shows an Action-Client to close the gripper by sending a goal to an Action-Server of the Nao wrapper. The Nao-Action-Client acts as ROS node and is registered under the topic *JointTrajectory*. Thereby, the client can send a special ROS-message, which is wrapped into a ROS-goal, to the Action-Server of the nao_controller node. This goal contains information about opening or closing as well as which gripper shall be manipulated.

Algorithm 5.7 Low-level function to close the gripper of the Nao robot

```
Require: $Side = left || Side = right$
1: procedure CloseGripper($Side$)
2:     $NaoActionClient \leftarrow makeActionClient(“/jointTrajectory“, “JointTrajectoryAction“)$
3:     $waitingForServer(NaoActionClient)$          ▷ Waiting until server is available
4:     if (Side=right) then $Goal \leftarrow createCloseRightGripperMsg()$
5:     else$Goal \leftarrow createCloseLeftGripperMsg()$
6:     end if
7:     $callGoal(NaoActionClient, Goal)$
8: end procedure
```

Similar to the algorithm 5.7, the next pseudocode 5.8 shows how to create an Action-goal to open the gripper of the Nao robot.

Algorithm 5.8 Low-level function to open the gripper of Nao

```
Require: $Side = left || Side = right$
1: procedure OpenGripper($Side$)
2:     $NaoActionClient \leftarrow makeActionClient(“/jointTrajectory“, “JointTrajectoryAction“)$
3:     $waitingForServer(NaoActionClient)$          ▷ Waiting until server is available
4:     if (Side=right) then $Goal \leftarrow createOpenRightGripperMsg()$
5:     else$Goal \leftarrow createOpenLeftGripperMsg()$
6:     end if
7:     $callGoal(NaoActionClient, Goal)$
8: end procedure
```

Algorithm 5.9 receives a 2D position which describes the target position to move the head. Note in line 3, that the Nao-Action-Client connects itself with the Joint-Trajectory-Action-Server.

Algorithm 5.9 Low-level function to move the head of Nao

```
1: procedure MoveHead($X, Y$)
2:     $NaoActionClient \leftarrow makeActionClient(``/jointTrajectory``, ``JointTrajectoryAction``)$
3:     $waitingForServer(NaoActionClient)$          ▷ Waiting until server is available
4:     $Goal \leftarrow createMoveHeadMsg(X, Y)$
5:     $callGoal(NaoActionClient, Goal)$
6: end procedure
```

The following pseudocode 5.10 shows how the challenge of moving the arm is solved.

Algorithm 5.10 Low-level function to move the arms of Nao

```
Require: $Side = left || Side = right$
1: procedure MoveArm($Target, Side$)          ▷ Target: 3D Position
2:     $NaoActionClient \leftarrow makeActionClient(``/jointTrajectory``, ``JointTrajectoryAction``)$
3:     $waitingForServer(NaoActionClient)$          ▷ Waiting until server is available
4:     if (Side=right) then $Goal \leftarrow createMoveRightArmMsg(Target)$
5:     else$Goal \leftarrow createMoveLeftArmMsg(Target)$
6:     end if
7:     $callGoal(NaoActionClient, Goal)$
8: end procedure
```

The stable position means that the Nao robot will be moved to a position which can be used as starting situation to execute every kind of action like walking.

Algorithm 5.11 Low-level function to move Nao to a stable position

```
1: procedure GoStable
2:     $NaoActionClient \leftarrow makeActionClient(``/jointTrajectory``, ``JointTrajectoryAction``)$
3:     $waitingForServer(NaoActionClient)$          ▷ Waiting until server is available
4:     $Goal \leftarrow createStablePositionMsg()$
5:     $callGoal(NaoActionClient, Goal)$
6: end procedure
```

Figure 5.1 provides an overview of the communication between an Action-Client and the nao_controller via ROS-Action-Messages. During the execution a server is able to send feedback messages to the client. After the execution the client receives the result from the server, which contains the target positions of the Nao joints. Furthermore, the client is able to subscribe status information related to the execution.

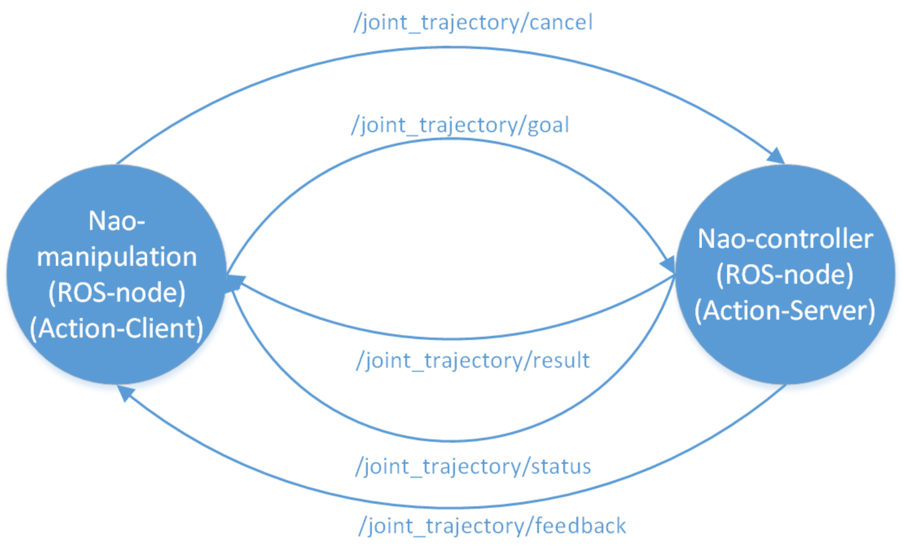

/joint_trajectory/cancel
/joint_trajectory/goal
Nao-manipulation (ROS-node) (Action-Client)
Nao-controller (ROS-node) (Action-Server)
/joint_trajectory/result
/joint_trajectory/status
/joint_trajectory/feedback

Figure 5.1: ROS-nodes for manipulation

Because of the encapsulation of client and server, it is possible to create Action-Clients and Action-Servers in different programming language like C++, LISP, Java or Python. This process module creates corresponding Action-Clients which delegates the resolved meta information to the Joint-Trajectory-Action-Server by sending an Action-Goal for every manipulation task.

Navigation process module

With this interface it is possible to move the Nao robot to an explicit position via high-level commands. It receives an action designator that contains itself a location designator including all needed meta information for the target position and resolves it to numeric values. Besides, this module delegates these values to the Nao control layer for their execution.
The ROS-node nao_path_follower is used, which implements an Action-Server to navigate the robot. This process module implements a mapping of the high-level command walkTo(Position) to Nao-specific low-level instructions of the nao_path_follower node. The following algorithm 5.12 creates one Action-Client which wraps the resolved target position to an Action-Goal and sends this message to the Action-Server, which is provided by the nao_path_follower.

Algorithm 5.12 Low-level function for walking

```
1: procedure WALKTO(TARGET)                                         ▷ Target: 3D Position
2:     $NaoActionClient \leftarrow makeActionClient(``/walkTarget``, ``MoveBaseAction``)$
3:     $waitingForServer(NaoActionClient)$                          ▷ Waiting until server is available
4:     $Goal \leftarrow createWalkingMsg(Position)$
5:     $callGoal(NaoActionClient, Goal)$
6: end procedure
```

Furthermore, this ROS-node commands the Nao to walk to the target position. The main difference to the manipulation process module is that this client communicates with another Action-Server. Additionally, the two nodes interact with each other by using a different topic. Note that the communication takes place via sending ROS-messages like goal, cancel, result, status and feedback on topic */walk_target*. The following figure shows the communication via ROS-messages between the ROS-nodes nao_path_follower and *NaoActionClient*.

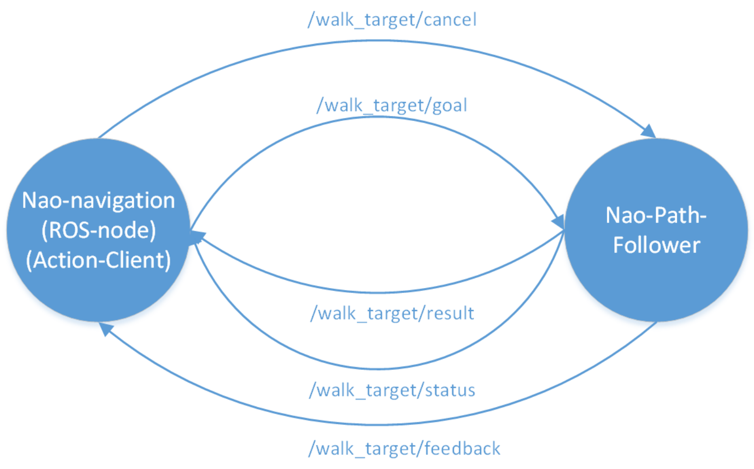


Figure 5.2: ROS-nodes for navigation

The client receives the temporary position via feedback message. After reaching the target position the nao_path_follower sends the result back to the client.

Speech process module

This module has the responsibility to translate text to speech. Pseudocode 5.13 shows how a node sends a sentence to the nao_speech server node. This process module uses the publish/subscribe pattern to achieve the communication between the ROS-Nodes.

Algorithm 5.13 Low-Level function translate text to speech

```
1: procedure TextToSpeech(Sentence)
2:     $NaoROSnode \leftarrow createROSnode()$
3:     $Topic \leftarrow "/speech"$
4:     $NaoROSnode.publish(Sentence, Topic)$
5: end procedure
```

The client publishes the sentence to the server which has already a subscription on the topic */speech*. Finally, the nao-speech server node delegates the message to the speech component of the NaoQi SDK.

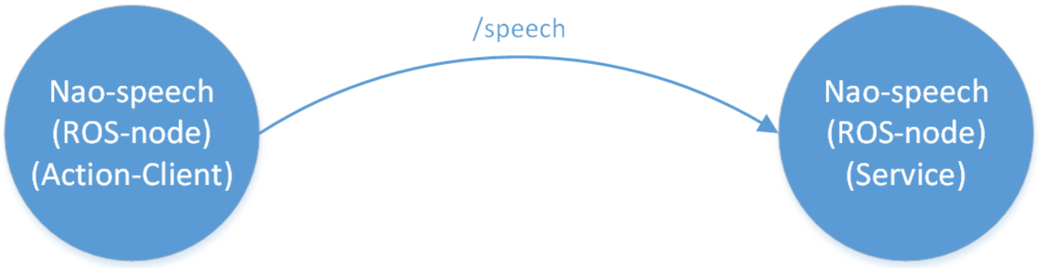

/speech
Nao-speech (ROS-node) (Action-Client)
Nao-speech (ROS-node) (Service)

Figure 5.3: ROS-nodes to tanslate text to speech

Creating Plans

In the following the goal term will be used as expression to describe the semantic of plans. As mentioned in section 3.4, goals can be used to create high-level plans by achieving goals as shown in peseudocode 5.14. Abstract plans defined in CPL[58] communicate with process modules by calling goals. This master thesis has defined particular goals for each action of the three process modules. Algorithm 5.14 demonstrates a plan by achieving several goals to manipulate the grippers as well as the head of the Nao robot.

[58]CRAM Plan Language

Algorithm 5.14 Creating High-Level Plans

```
1: procedure AbstractPlan()
2:     Parallel
3:         $achieve(openGripper, left)$
4:         $achieve(openGripper, right)$
5:         $achieve(moveHead, Position(2,3))$
6:     EndParallel
7: end procedure
```

5.2 Gateway

Note that the design approach in 4.2.1 considers placing the interface on the gateway as interaction point between the Nao robot and RoboEarth. The open source operating system Ubuntu 12.04 is chosen to be installed on the gateway, because it is superior compatible with the used components than other operating systems like Windows or Mac OS X. According to `http://wiki.ros.org/ROS/Installation` Ubuntu seems to be the best supported operating system for the ROS framework. The 12.04 version is selected because of its compatibility with the Nao-ROS-Stack. As shown in 5.4, the Nao interface is built on top of the Ubuntu operating system. It is unimportant whether the gateway is placed on a laptop or on a desktop computer, but it is recommended to place the gateway next to the environment of the robot to reduce the distance between the Nao robot and its interface. Thereby, this work has arranged to place the gateway on a remote laptop.

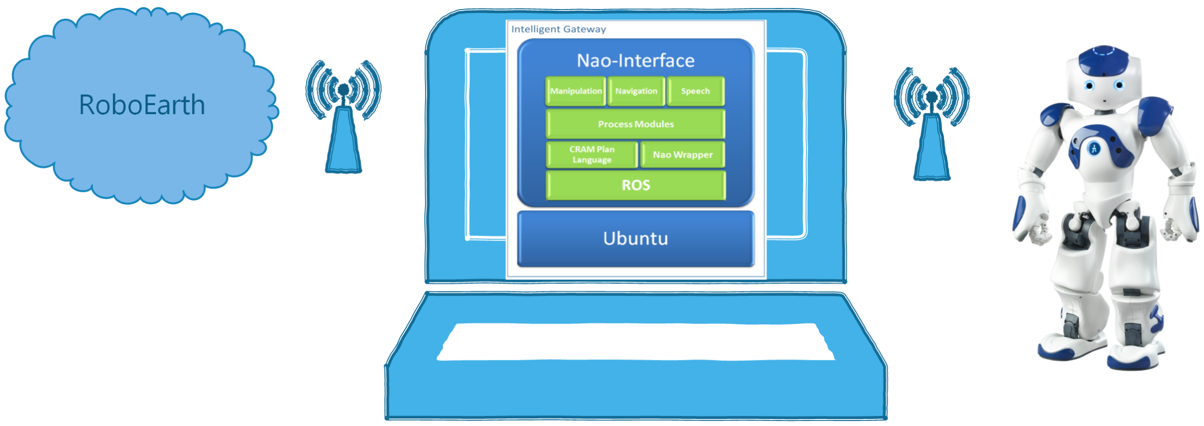


Figure 5.4: Overview of a plan sharing system by using the intelligent gateway approach Nao (2014)

Chapter 6

Experiments

This part of the master thesis shows some practical experiences with the implemented interface of the Nao robot. Chapter 6 presents some trials to test the implemented process modules on the real Nao robot. The experiments aim to investigate the possibility to access the Nao process modules (*:manipulation*, *:navigation*, :*speech*) by high-level plans. These tests are used to analyze the behaviour of the process modules. Furthermore, another experiment demonstrates the possibility to download and execute an abstract plan from the RoboEarth web service.

Setup

All experiments are performed on the same infrastructure as presented in figure 6.1.

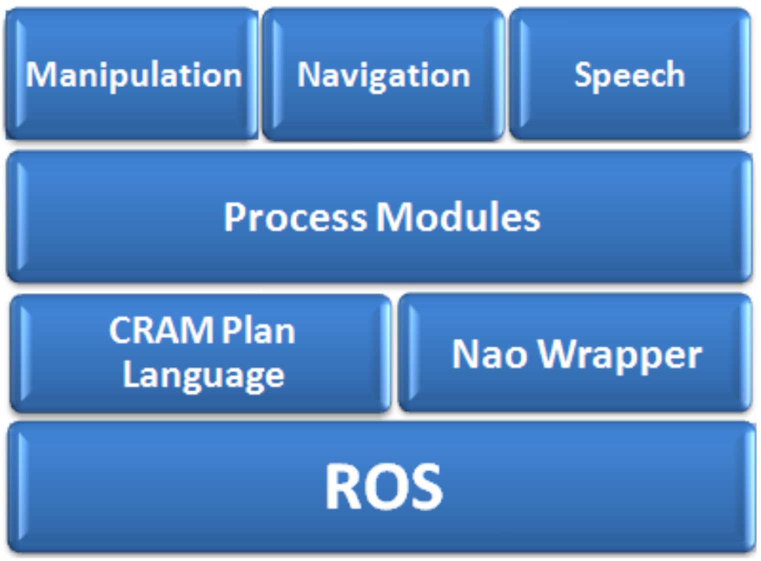


Figure 6.1: Cloud-enabled Nao architecture

Thereby, the Nao interface runs inside a gateway placed on a laptop. The following trials use the Emacs[59] environment to develop and execute abstract plans in the CRAM plan language. Before high-level plans can be executed, a special setup is needed to perform the experiments. Since the CRAM plans run inside the ROS middleware, the Emacs editor has to be integrated into ROS as well. Therefore, `http://wiki.ros.org/rosemacs` provides a ROS-based editor for Common Lisp. Firstly, the Nao interface has to be started inside the Emacs environment. This step includes the launch of the process modules in separate threads. Besides, it has to be ensured that the laptop is connected with the Nao network via WLAN[60] or LAN[61]. The simulation tool ChoregrapheAldebaran-Robotics (2013c) provided by Aldebaran-Robotics (2013c) is also used to display the behaviour of the Nao robot as shown in figure 6.2. The following figure shows the test environment including Choregraphe, Emacs editor, laptop and Nao robot.

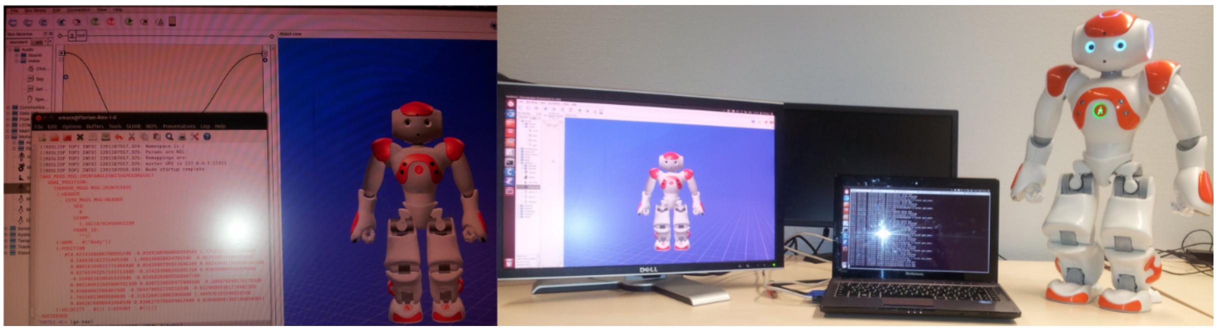


Figure 6.2: Test environment for the following experiments

6.1 Manipulation process module

The concept of the process module has already been explained in sections 4.1 and 5.1.4.3. This experiment shows some practical trials of the manipulation process module. High level plans are able to call actions like moving the arm or opening the gripper by accessing this interface. This experiment tests the usability by execution of some actions, which are implemented in subsection 5.1.4.3. The next algorithm presents a high-level plan to manipulate the Nao robot.

[59]http://www.gnu.org/software/emacs/
[60]Wireless Local Area Network
[61]Local Area Network

Algorithm 6.1 Manipulate the joints of Nao by calling high-level commands

```
 1: procedure AbstractPlan()
 2:     Parallel
 3:         $achieve(openGripper, left)$
 4:         $achieve(openGripper, right)$
 5:         $achieve(crouch)$
 6:         $achieve(moveHead, Position(2, 3))$
 7:         $achieve(moveArm, left, Position(1, 1, 1))$
 8:         $achieve(moveArm, right, Position(2, -1, 3))$
 9:         $achieve(closeGripper, left)$
10:         $achieve(closeGripper, right)$
11:     EndParallel
12: end procedure
```

Pseudocode 6.1 creates the following behaviour of the real Nao robot, presented in figure 6.3.

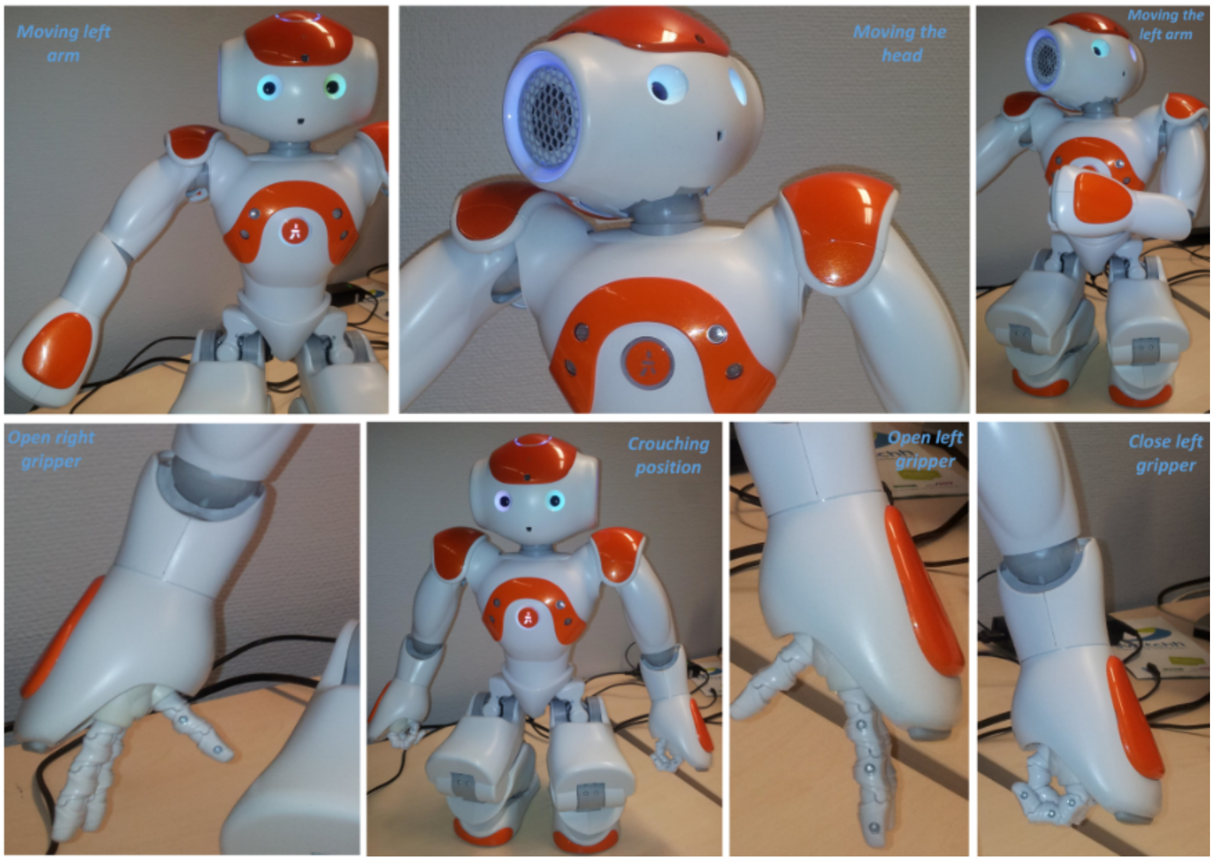


Figure 6.3: Experiments to manipulate the Nao robot

6.2 Navigation process module

The next test aims to command the Nao robot to walk to the specific position by executing the next pseudocode listing. The walking behaviour is presented in the following graphic slide 6.4.

Algorithm 6.2 Navigates the Nao robot to the particular position

```
1: procedure ABSTRACTPLAN()
2:     $achieve(walkTo, Position(1,1,1))$
3: end procedure
```

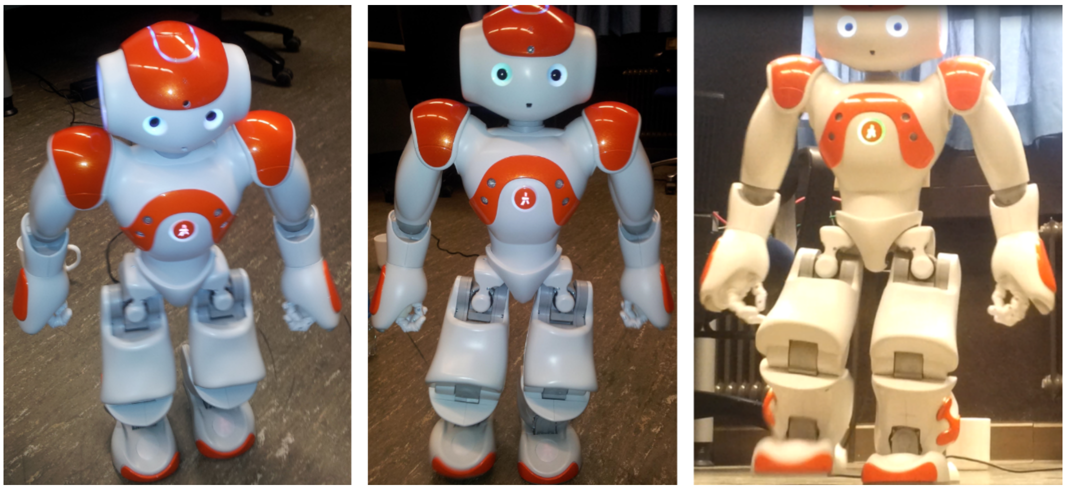


Figure 6.4: Experiments to navigate the Nao robot

6.3 Speech process module

This test sends English sentences to the speech process module as shown in algorithm 6.3. The process module sends the sentence to the text-to-speech module of the NaoQi framework. The output is displayed by the Choregraphe simulator presented in graphic 6.5.

Algorithm 6.3 Translates text-to-speech

```
1: procedure ABSTRACTPLAN()
2:     $achieve(speech, “HiIamNao.LetsRock‘nRol“)$
3: end procedure
```

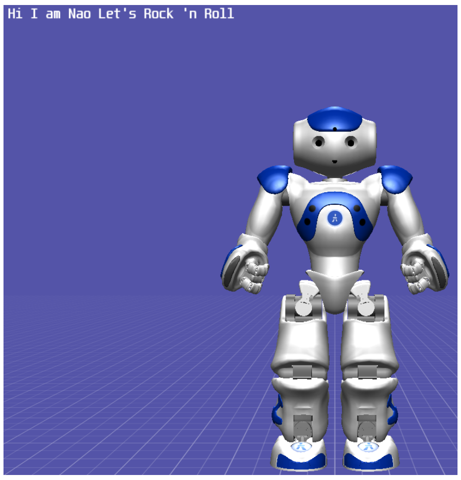
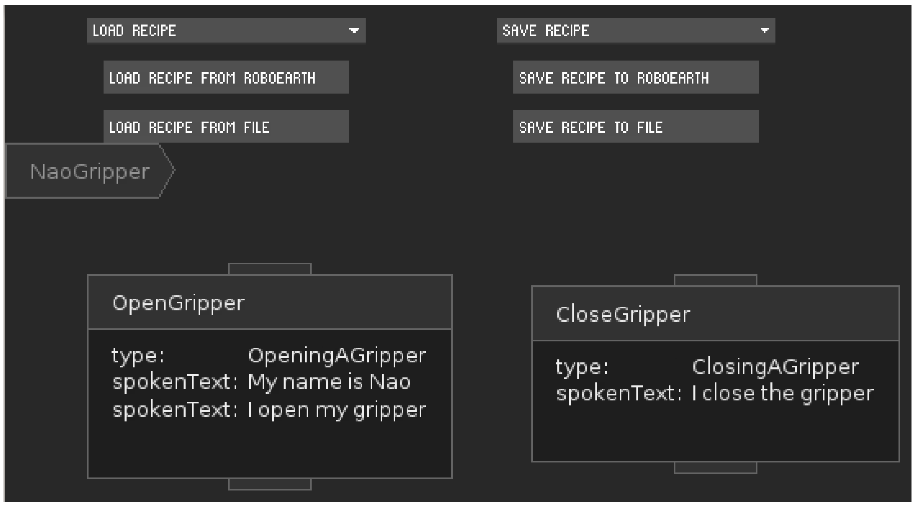


Figure 6.5: Experiments to translate text-to-speech

6.4 Downloading an abstract plan

This experiment demonstrates how to download a high-level plan from RoboEarth to manipulate the gripper of the Nao robot. As presented in Figure 6.6, an abstract plan is created by using the Action Recipe Editor[62] provided by the KnowRob framework. Through activating the button "*Save Recipe to RoboEarth*", the plan *NaoGripper* will be uploaded to the RoboEarth database in OWL[63]-format.



OpenGripper

type: OpeningAGripper
spokenText: My name is Nao
spokenText: I open my gripper

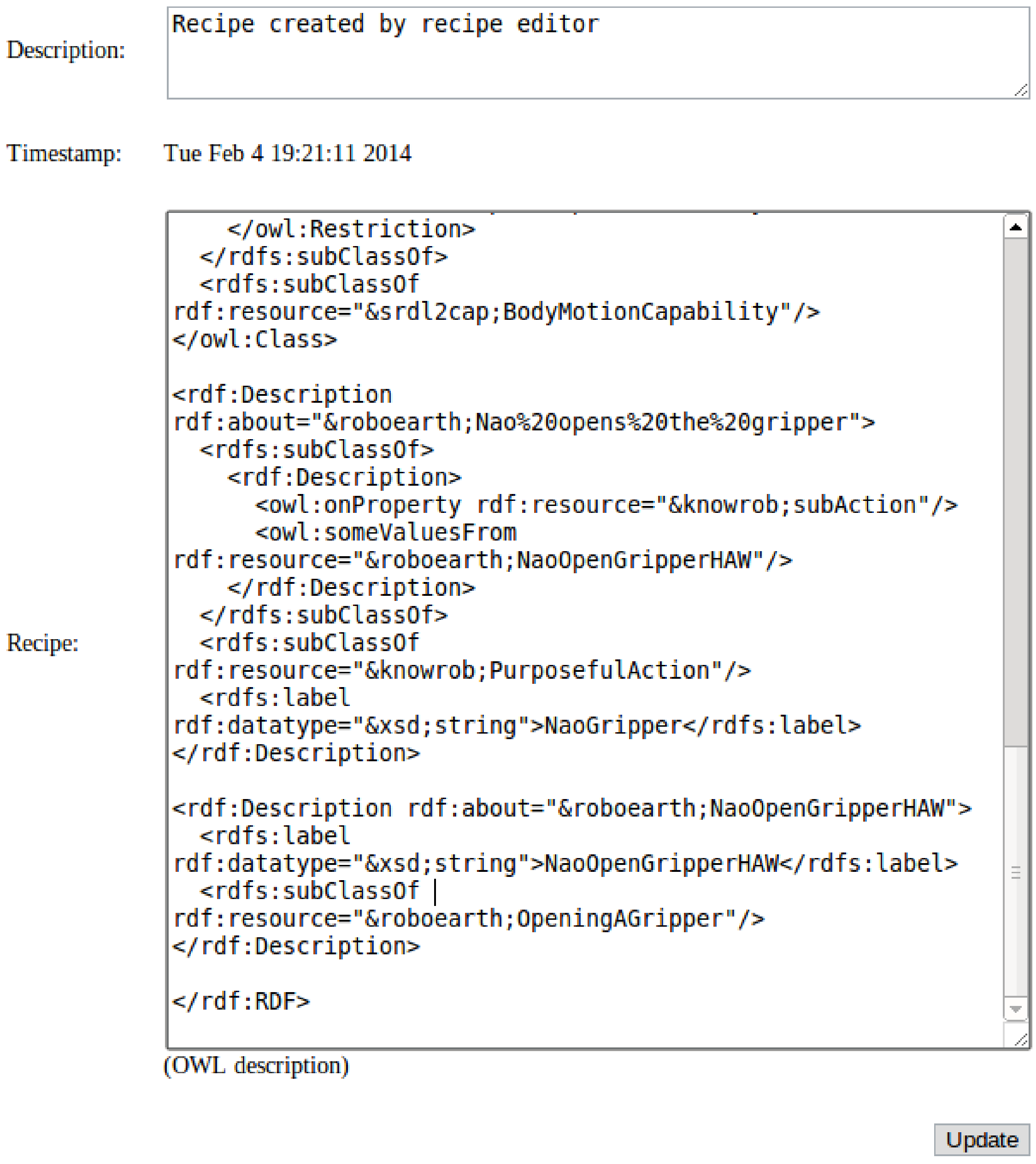
CloseGripper

type: ClosingAGripper
spokenText: I close the gripper

Figure 6.6: Creating an abstract plan to manipulate the gripper

[62]`http://www.knowrob.org/doc/action_recipe_editor`
[63]Web Ontology Language

As presented in figure 6.7, this abstract plan can be found via requesting the RoboEarth web interface on `http://api.roboearth.org`.


Description: Recipe created by recipe editor

Timestamp: Tue Feb 4 19:21:11 2014

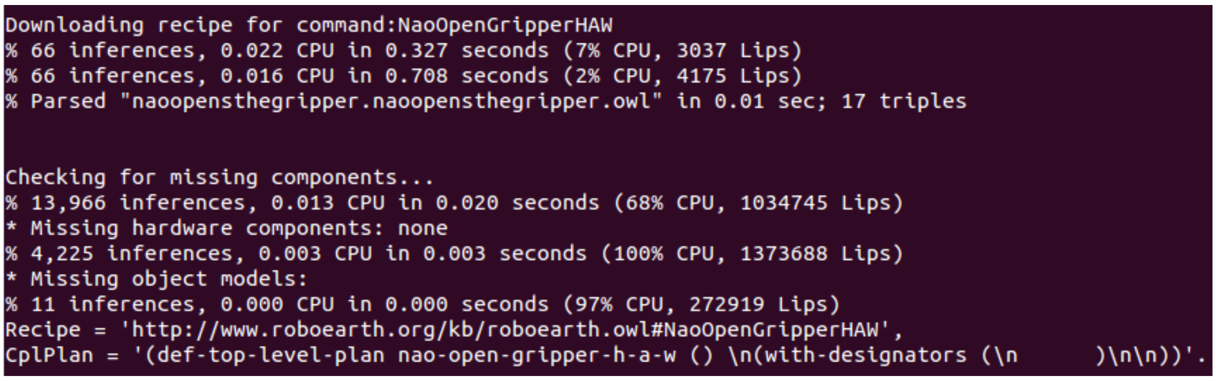
Recipe:

```
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf
rdf:resource="&srdl2cap;BodyMotionCapability"/>
</owl:Class>

<rdf:Description
rdf:about="&roboearth;Nao%20opens%20the%20gripper">
    <rdfs:subClassOf>
      <rdf:Description>
        <owl:onProperty rdf:resource="&knowrob;subAction"/>
        <owl:someValuesFrom
rdf:resource="&roboearth;NaoOpenGripperHAW"/>
      </rdf:Description>
    </rdfs:subClassOf>
    <rdfs:subClassOf
rdf:resource="&knowrob;PurposefulAction"/>
    <rdfs:label
rdf:datatype="&xsd;string">NaoGripper</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="&roboearth;NaoOpenGripperHAW">
    <rdfs:label
rdf:datatype="&xsd;string">NaoOpenGripperHAW</rdfs:label>
    <rdfs:subClassOf
rdf:resource="&roboearth;OpeningAGripper"/>
</rdf:Description>

</rdf:RDF>
```

(OWL description)

Figure 6.7: Creating an abstract plan to manipulate the gripper

By using the transforming engine of subsection 5.1.4, it is possible to download such kind of high-level plan. The following graphic presents a download-client, which also generates a CPL plan. Since the Nao interface provides CPL-functions to manipulate the gripper, this plan can be executed on the Nao robot.



```
Downloading recipe for command:NaoOpenGripperHAW
% 66 inferences, 0.022 CPU in 0.327 seconds (7% CPU, 3037 Lips)
% 66 inferences, 0.016 CPU in 0.708 seconds (2% CPU, 4175 Lips)
% Parsed "naoopensthegripper.naoopensthegripper.owl" in 0.01 sec; 17 triples


Checking for missing components...
% 13,966 inferences, 0.013 CPU in 0.020 seconds (68% CPU, 1034745 Lips)
* Missing hardware components: none
% 4,225 inferences, 0.003 CPU in 0.003 seconds (100% CPU, 1373688 Lips)
* Missing object models:
% 11 inferences, 0.000 CPU in 0.000 seconds (97% CPU, 272919 Lips)
Recipe = 'http://www.roboearth.org/kb/roboearth.owl#NaoOpenGripperHAW',
CplPlan = '(def-top-level-plan nao-open-gripper-h-a-w () \n(with-designators (\n      )\n\n))'.
```

Figure 6.8: Download-client

6.5 Résumé

This chaper presented the possibility to control the Nao robot via abstract commands in the CRAM Plan Language. The implemented process modules were tested to manipulate, navigate as well as to translate text to speech. Finally, experiment 6.4 showed that the interface of section 5.1 permits Nao robots to download and execute a high-level plan to manipulate the gripper.

Chapter 7

Evaluation

This chapter evaluates the Nao interface based on the executed experiments in chapter 6. It is divided into three parts. Firstly, this chapter starts with a discussion about if and how this work answers the research hypothesis of subsection 3.6.3 as well as which requirements of subsection 3.6.2 have been satisfied. According to 3.6.3, this master thesis has investigated how robots are able to transform and execute high-level plans from a cloud service by proving this approach on one specific robot platform. Chapter 4 has designed an interface to meet the conditions 1-5 of subsection 3.6.2.

1. ***Portability*** is provided by the interface using the concept of process modules as well as the aid of the ROS framework. Section 8.2 shows an example how to configure this interface for the TurtleBot[64] robot. Thereby, it is potential to execute the same abstract plan on a different robot platform by exchanging the process modules and the ROS wrapper. The power of this approach lies definitely on the flexibility of the robot programs.

2. ***Sending plan requests*** is solved by using the KnowRob framework to explore the database of a web service like RoboEarth for a high-level plan. The experiment of section 6.4 has demonstrated how high-level plans can be downloaded through the exploring engine (see subsection 5.1.2) of the Nao interface.

3. ***Mapping of abstract commands to robot-specific instructions*** is created by this master thesis through implementing process modules inside a transformation engine to manipulate, to navigate as well as to access the speech module of the Nao robot. The trials of sections 6.1, 6.2 and 6.3 have presented some tests to access the process modules by high-level plans. It was demonstrated how to start the three process modules manipulation, navigation and speech separately in their own thread. Thus, it is possible to access these modules in parallel.

[64] `http://www.turtlebot.com`

The processed tests have shown first experiences how to control the Nao robot with high-level plans via process modules like manipulating the actuators of the Nao as well as to navigate the robot to a specific position. Furthermore, another trial tested the speech process module to translate text to speech.

4. ***Downloaded plans are executable*** by using the CRAM Plan Language. Experiment 6.4 has demonstrated how to send a plan request to the RoboEarth web service. The implemented interface of section 5.1 enables Nao robot to execute abstract cloud-stored commands.

5. ***Robots can process knowledge, which is generated by different robots*** by using the ROS framework as well as RoboEarth that stores robot-independent information. The trials of chapter 6 have shown the possibility to execute abstract commands which are not Nao specific.

The second part evaluates which requirements were not implemented in the realization chapter 5.

1. ***Knowledge sharing between different robot platforms*** was not implemented while writing this master thesis. This work has carried out a knowledge sharing among Nao robots.

2. ***An uploading mechanism*** was also not implemented.

Finally, this chapter highlights situations in which a cloud-enabled Nao robot has advantages compared to a cloud-disabled Nao robot. For example if the Nao robot is recently delivered to the end user, the Nao is equipped without software. Indeed, the user can create programs using the NaoQi framework, but it is not possible to execute complex applications immediately. Instead, a cloud-connected Nao robot can download executable plans instantly from a cloud service to perform a task without having any previous knowledge. Another point is that the user of a cloud-disabled robot has to provide programming skills in Python, Java or C++. Using the implemented interface of this work, a user without programming skills is able to control the robot in an abstract way.

Chapter 8

Conclusion

8.1 Summary

This master thesis showed how to apply the approach of knowledge sharing via cloud services on robots to improve their learning mechanisms. Moreover, it demonstrated this concept by developing an interface to connect the Nao platform to the web service RoboEarth. This interface permits Nao robots to download and execute abstract plans. It also allows Nao robots to process plans that can be generated by heterogeneous robots.
Chapter 3 introduced the topic of Cloud Robotics by demonstrating common challenges and related projects. It revealed which software components are required to provide a plan sharing among different robot platforms.
Chapter 4 designed a Nao-specific interface, which mainly provides a mechanism to explore a cloud service for a specific plan request as well as a transformation engine. Inside the transformation engine this work uses the concept of process modules to map abstract commands to Nao-depending instructions, which are provided by the robot middleware ROS[65]. These modules encapsulate the Nao hardware components. Furthermore, several communication approaches showed different design concepts providing a plan sharing system including the Nao robot, the Nao interface and a cloud service. These concepts differ from each other by locating the Nao interface in another mode.
The realization chapter 5 described the implementation of a plan sharing system via an intelligent gateway permitting Nao robots to download and execute high-level plans from the RoboEarth web service. The Nao interface is placed on a gateway acting as loose-coupled interaction point between the robot and RoboEarth. It presented the KnowRob framework as exploring engine to find a satisfying plan inside the RoboEarth database. This master thesis implemented the transformation engine with the aid of the CRAM Plan Language as well as the Nao-ROS wrapper.

[65]Robot Operating System

Chapter 5 carried out process modules to manipulate, navigate as well as accessing the speech module of the Nao in an abstract way without thinking about how the robot solves these tasks. Furthermore, it showed the communication between the process modules and the Nao-Wrapper via ROS-Nodes.
Besides, some experiments presented how to use the implemented process modules for the Nao platform and demonstrated how high-level plans can be downloaded. The evaluation showed that this approach provides definitely the possibility to control the Nao robot in an abstract way as well as to execute programs that are not Nao-specific. Finally, the outlook describes further suggestions to continue the work of this master thesis.

8.2 Outlook

This section aims to present possible advancements of the master thesis. It also shows further opportunities to continue this work.

Improvements of the interface

Firstly, the Nao interface offers some possibilities for advancements.

A perception process module extends the functionality of the Nao interface. With the aid of this interface a high-level plan would be able to access the object detection and speech recognition of the Nao platform.

The uploading mechanism would permit Nao robots to create abstract plans that can be uploaded to a web service like RoboEarth. As RoboEarth stores robotic-specific knowledge in a OWL[66]-like format, called RoboEarth languageTenorth et al. (2012), Nao has to transform plans defined in its own internal representation to the RoboEarth language. One opportunity is to generate knowledge by the tool Choreographe. This information will be translated into the RoboEarth language, which can be uploaded to the RoboEarth web service.

[66]Web Ontology Language

Plan Sharing among the Nao platform and the TurtleBot

Since the developed Nao interface provides a knowledge sharing with heterogeneous robots, further projects could continue this work to exchange executable plans between a Nao robot and another ROS[67]-enabled robots like the TurtleBot[68] or the PR2 robot. Both robots are developed by the popular robot manufacturer Willow Garage. As this company is also responsible for the maintenance and advancement of the ROS framework, both robots are fully integrated into ROS.
To adapt the plan sharing system of section 5.2 to the TurtleBot platform, there only have to be TurtleBot-depending process modules, which are located on top of the TurtleBot-ROS wrapper[69]. Figure 8.1 shows an architecture that allows the TurtleBot robot to download and execute high-level plans, which can be generated by the Nao robot.

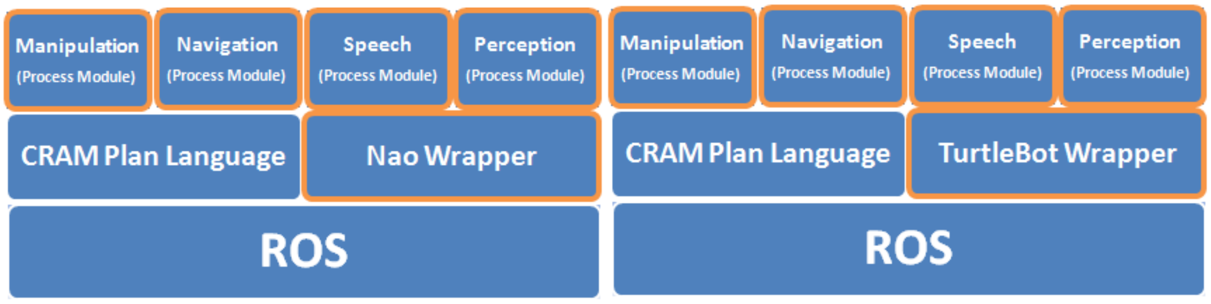

Manipulation (Process Module)
Navigation (Process Module)
Speech (Process Module)
Perception (Process Module)
CRAM Plan Language
Nao Wrapper
ROS
Manipulation (Process Module)
Navigation (Process Module)
Speech (Process Module)
Perception (Process Module)
CRAM Plan Language
TurtleBot Wrapper
ROS

Figure 8.1: Nao-TurtleBot Cooperation

[67]Robot Operating System
[68]`http://www.turtlebot.com`
[69]`http://wiki.ros.org/Robots/TurtleBot`

Stepwise search space extension

This concept aims to extend the approaches of section 4.2 by controlling which cloud services will be used when a Nao robot sends a plan request to the internet. Firstly, this approach places a private cloud into the environment of the robot to reduce the download latency. This private cloud is only accessible for robots that operate in the same network. If the private cloud does not provide a high-level plan, which satisfies the request, the cloud service connects itself to a bigger cloud service. Through this concept, the search space for an abstract plan is dynamic. This extension can be used by all of the three approaches presented in section 4.2. The following graphic shows the extension of a plan sharing system by using the concept of a stepwise search space.

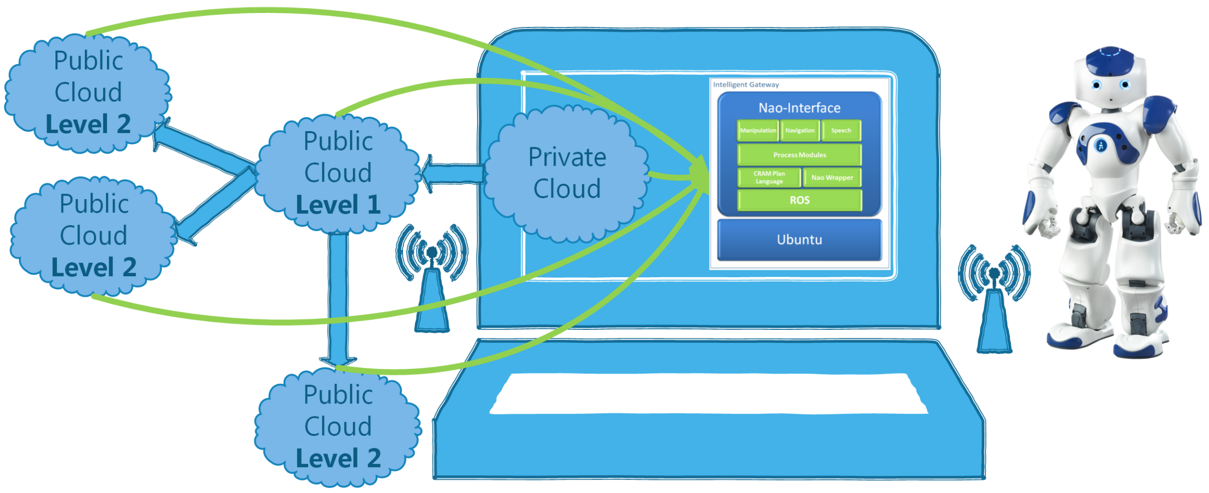


Figure 8.2: Sketch of a stepwise search space extension Nao (2014)

Downloading safety plans

Subsection 3.4 represents one of the challenges not considered in this work. Nevertheless, it is very important to be concerned about this topic. Plans stored in a private cloud can be considered as secure, but public cloud stored plans require specified verifying mechanisms. Similar to smartphone application stores like *Apple Store* or *Google Play* it is necessary that a knowledge database like RoboEarth is able to ensure that no malware will be downloaded by the robot.

Plan caching

Certainly, it is not very efficient to download a plan from a cloud service each time a robot wants to solve the same task. So, it is recommendable to think about caching policies to optimize a plan sharing system. This challenge depends on the memory capacity of the robot. For smart robots with low memory, it could be helpful to use algorithms swapping elder plans by recently executed plans. Robots that represent fat clients with high memory capacity are able to store all downloaded plans in its internal database.

Plan downloading via smartphone

This idea combines the topic Cloud Robotics with teleoperation[70]. As considered in chapters 3 and 4, robots receive the plan request by human through its own speech recognition system. An alternative could be to use a smartphone as interaction interface permitting humans to send plan requests to its robot from any place of the world.

Commercial app store for robots

Besides, it is interesting to think about how to earn money with this topic. Thus, the owners of the robots have to pay for each plan the robot downloads. Different from the smartphone application stores, it has to be checked if the plan is generally executable before a robot can download it.

Connecting Nao with the Google Cloud Platform

Subsection 3.5.5 has already demonstrated the possibility to use Google-Technologies to counter the challenges of Cloud Robotics. A very attractive opportunity to advance this work would be to use the Google Cloud Platform[71] as platform as a service permitting Nao robots to cooperate with other robots.

[70]Controlling a robot via remote commands

[71]`https://cloud.google.com`

References

IFR, “Executive summary,” Tech. Rep., 2013. [Online]. Available: http://www.ifr.org/uploads/media/Executive_Summary_WR_2013_01.pdf

E. Guizzo, “Robots with their heads in the clouds,” Tech. Rep., 2011. [Online]. Available: http://isdlab.aet.ntnu.edu.tw/roboticscloud/references/Robots%20With%20Their%20Heads%20in%20the%20Clouds.pdf

P. Mell and T. Grance, “The nist definition of cloud computing recommendations of the national institute of standards and technology,” Tech. Rep., 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

M. Quigley, B. Gerkeyy, K. Conleyy, J. Fausty, T. Footey, J. Leibsz, E. Bergery, R. Wheelery, and A. Ng, “Ros: an open-source robot operating system,” Tech. Rep., 2009. [Online]. Available: http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf

A. Martinez and E. Fernández, *Learning ROS for Robotics Programming*. Birmingham: Packt Publishing, 2013.

W. Woodall. (2013, Oct.) Client libraries @ONLINE. [Online]. Available: http://wiki.ros.org/Client%20Libraries

Nao. (2014) Nao robot, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://www.ais.uni-bonn.de/images/robots/Nao.png

——. (2012) Modeling of mesencephalic locomotor region for nao humanoid robot, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://www.emeraldinsight.com/content_images/fig/0490390203015.png

Aldebaran-Robotics. (2013) Nao documentation @ONLINE. [Online]. Available: https://community.aldebaran-robotics.com/doc/1-14/index.html#

——. (2013) Naoqi framework @ONLINE. [Online]. Available: https://community.aldebaran-robotics.com/doc/1-14/naoqi/index.html

M. Weiser, “The computer for the 21st century,” Tech. Rep., 1991. [Online]. Available: http://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf

M. Inaba, “Remote-brained robots,” Tech. Rep., 1993. [Online]. Available: http://ijcai.org/Past%20Proceedings/IJCAI-97-VOL2/PDF/118.pdf

M. Inaba, S. Kagami, T. Ishikawa, F. Kanehiro, K. Takeda, and H. Inoue, “Vision-based adaptive and interactive behaviors in mechanical animals using the remote-brained approach,” Tech. Rep., 1994. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=407540

M. Inaba, S. Kagami, F. Kanehiro, Y. Hoshino, and H. Inoue, “A platform for robotics research based on the remote-brained robot approach,” Tech. Rep., 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.3253&rep=rep1&type=pdf

M. Inaba, T. Ninomiya, Y. Hoshino, K. Nagasaka, S. Kagami, and H. Inoue, “A remote-brained full-body humanoid with multisensor imaging system of binocular viewer, ears, wrist force and tactile sensor suit,” Tech. Rep., 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.3253&rep=rep1&type=pdf

J. Kuffner, “Cloud robotics: (and the future of distributed intelligence).” IROS2011 Workshop: Knowledge Representation for Autonomous Robots, 2011.

J. M. Quintas, P. J. Menezes, and J. M. Dias, “Cloud robotics: Towards context aware robotic networks,” Tech. Rep., 2011. [Online]. Available: http://mrl.isr.uc.pt/archive/752-062.pdf

M. Quigley, B. Gerkeyy, K. Conleyy, J. Fausty, T. Footey, J. Leibsz, E. Bergery, R. Wheelery, and A. Ng, “Ros an open-source robot operating system,” Tech. Rep., 2009. [Online]. Available: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

P. Fitzpatrick, G. Metta, and L. Natale, “Towards long-lived robot genes,” Tech. Rep., 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889007001364#

R. Smits, T. D. Laet, K. Claes, P. Soetens, J. D. Schutter, and H. Bruyninckx, “Orocos: A software framework for complex sensor-driven robot tasks,” Tech. Rep., 2008. [Online]. Available: http://robotics.usc.edu/~gerkey/research/final_papers/icar03-player.pdf

B. P. Gerkey, R. T. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” Tech. Rep., 2003. [Online]. Available: http://robotics.usc.edu/~gerkey/research/final_papers/icar03-player.pdf

E. Kutluhan, J. Hendler, and D. S. Nau, “Htn planning: Complexity and expressivity,” Tech. Rep., 1994. [Online]. Available: http://www.cs.umd.edu/~nau/papers/erol1994htn.pdf

K. L. Myers and D. E. Wilkins, “The act formalism,” Tech. Rep., 1997. [Online]. Available: http://www.ai.sri.com/~act/act-spec.pdf

M. Ghallab, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “Pddl planning domain definition language,” Tech. Rep., 1998. [Online]. Available: http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0203/pddl.pdf

M. R. Genesereth and R. E. Fikes, “Knowledge interchange format version 3.0,” Tech. Rep., 1992. [Online]. Available: https://www.cs.auckland.ac.nz/courses/compsci367s2c/resources/kif.pdf

D. Lu, “Urdf and you.” ROSCon 2012, 2012.

M. Barnes, E. L. Finch, and S. C. E. Inc., “Collada digital asset schema release 1.5.0,” Tech. Rep., 2008. [Online]. Available: http://www.khronos.org/files/collada_spec_1_5.pdf

L. Kunze, T. Roehm, and M. Beetz, “Towards semantic robot description languages,” Tech. Rep., 2011. [Online]. Available: https://ias.in.tum.de/_media/spezial/bib/kunze11srdl.pdf

D. McDermott, “A reactive plan language,” Tech. Rep., 1991. [Online]. Available: http://www.cs.yale.edu/publications/techreports/tr864.pdf

M. Beetz, L. Moesenlechner, and M. Tenorth, “Cram a cognitive robot abstract machine for everyday manipulation in human environments,” Tech. Rep., 2010. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5650146

M. Beetz and D. McDermott, “Declarative goals in reactive plans,” Tech. Rep., 1992. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=AEBF0C40167CEFC345BF87BCCB81142B?doi=10.1.1.17.5031&rep=rep1&type=pdf

J. Bohren. (2010) Smach package summary, last accessed on 8.02.2014 @ONLINE. [Online]. Available: http://wiki.ros.org/smach

S. Harnad, “The symbol grounding problem,” Tech. Rep., 1990. [Online]. Available: http://courses.media.mit.edu/2004spring/mas966/Harnad%20symbol%20grounding.pdf

M. Waibel, M. Beetz, J. Civera, R. Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, A. P. J.M.M. Montiel, B. Schießle, M. Tenorth, O. Zweigle, and R. van de Molengraft, “Roboearth a world wide web for robots,” Tech. Rep., 2011. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5876227

M. Tenorth, A. Perzylo, R. Lafrenz, and M. Beetz, “The roboearth language: Representing and exchanging knowledge about actions, objects, and environments,” Tech. Rep., 2012. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6224812

M. Beetz and M. Tenorth, “Knowrob knowledge processing for autonomous personal robots,” Tech. Rep., 2009. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5354602

Nao. (2011) Robotique: Nao, robot humanoide, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://uncafemonblocnote.fr/wp-content/uploads/2011/02/NAO-4_stand.png

TurtleBot. (2014) Turtlebot robot, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://ros.informatik.uni-freiburg.de/roswiki/attachments/Robots(2f)TurtleBot/turtlebot320.png

D. Hunziker, M. Gajamohan, M. Waibel, and R. D. Andrea, “Rapyuta the roboearth cloud engine,” Tech. Rep., 2013. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6630612

M. Gajamohan, “Understanding the roboearth cloud.” ROSCon 2013, 2013.

Y. Ogata, E. Spaho, K. Matsuo, L. Barolli, and F. Xhafa, “A knowledge sharing p2p system between robots using jxta-overlay,” Tech. Rep., 2011. [Online]. Available: http://robotics.usc.edu/~gerkey/research/final_papers/icar03-player.pdf

X. Fan and T. C. Henderson, “Robotshare: A google for robots,” Tech. Rep., 2008. [Online]. Available: http://www.doc.ic.ac.uk/~xf309/Misc/IJHR.pdf

R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. Kumar, K. D. Meng, and G. W. Kit, “Davinci a cloud computing framework for service robots,” Tech. Rep., 2010. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5509469

B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, “Cloud-based robot grasping with the google object recognition engine,” Tech. Rep., 2013. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6631180

MyRobots.com. (2013, Dec.) Connect your robots. reap the benefits@ONLINE.

Oddwerx. (2012) Smartphone-roboter läuft mit open-source-betriebssystem, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://www.golem.de/1204/sp_91274-34227-i.jpg

Roméo. (2014) Roméo robot, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://spectrum.ieee.org/image/1749400

SmartBot. (2013) Smartbot puts your smartphone to a new use, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://images.gizmag.com/hero/smartbot.jpg

H. Y. Nii, “Blackboad systems,” Tech. Rep., 1986. [Online]. Available: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/86/1123/CS-TR-86-1123.pdf

PR2. (2013) Pr2 robot, last accessed on 7.02.2014 @ONLINE. [Online]. Available: http://robotics.usc.edu/resl/media/uploads/photos/robots/pr2/pr2_size-large.jpg

D. Prasad, “Multi-robot map data merging in cloud robotics systems,” Tech. Rep., 2013.

M. Tenorth, “Knowledge processing for autonomous robots,” Tech. Rep., 2011. [Online]. Available: http://ias.cs.tum.edu/~tenorth/thesis.pdf

M. Beetz and D. McDermott, “Improving robot plans during their execution,” Tech. Rep., 1994. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.2901&rep=rep1&type=pdf

A. Müller, “Transformational planning for autonomous household robots using libraries of robust and flexible plans,” Tech. Rep., 2008. [Online]. Available: http://mediatum.ub.tum.de/doc/645588/645588.pdf

A. Hornung. (2009, Dec.) Ros-stack for the nao humanoid robot @ONLINE. [Online]. Available: http://wiki.ros.org/Robots/Nao

F. Johannßen, “Nao in the cloud,” Tech. Rep., 2013. [Online]. Available: http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2012-proj1/johannssen.pdf

——, “Nao robots in the cloud: An interface to execute abstract plans,” Tech. Rep., 2013. [Online]. Available: http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master12-13-proj2/johannssen.pdf

L. Moesenlechner. (2009, Dec.) Cram-core stack @ONLINE. [Online]. Available: http://wiki.ros.org/cram_core

Aldebaran-Robotics. (2013) Choregraphe @ONLINE. [Online]. Available: http://www.aldebaran-robotics.com/en/Discover-NAO/Software/choregraphe.html

List of Figures

2.1 Core concepts of the ROS framework Martinez and Fernández (2013) 6
2.2 Nao Next Gen (Nao (2014); Nao (2012)) 8
2.3 NaoQi SDK Aldebaran-Robotics (2013b) 9

3.1 Description of the remote-brained robot approach Inaba (1993) 12
3.2 Knowledge sharing for robots 14
3.3 RoboEarth (Waibel et al. (2011); Nao (2011); TurtleBot (2014)) 20
3.4 Rapyuata Gajamohan (2013) 21
3.5 RobotShare 23
3.6 DaVinci Arumugam et al. (2010) 24
3.7 Cloud-Based Robot Grasping - Offline Phase Kehoe et al. (2013) 25
3.8 Cloud-Based Robot Grasping - Online Phase Kehoe et al. (2013) 26
3.9 MyRobots.com MyRobots.com (2013) 27
3.10 Smartphone Robots (Oddwerx (2012); Roméo (2014); SmartBot (2013)) . . . 29
3.11 Knowledge sharing among heterogeneous robots via cloud services (Roméo (2014); PR2 (2013)) 31
3.12 Use cases - Nao in the cloud 32

4.1 Component diagram of the Nao interface 35
4.2 Communication process between all related components Nao (2011) 37
4.3 Transformation of high-level plans to Nao-depending instructions 38
4.4 Interaction between process modules and Nao wrapper 39
4.5 Sketch of an intelligent gateway 41
4.6 Communication among the Nao robot, the intelligent gateway and the web service 43
4.7 Sketch of a transparent gateway 44
4.8 Sequence diagram for the transparent gateway approach 45
4.9 Sketch of a plan sharing system without a gateway 46
4.10 Communication between the Nao robot and the web service 47

5.1 ROS-nodes for manipulation 59

5.2 ROS-nodes for navigation 60
5.3 ROS-nodes to tanslate text to speech 61
5.4 Overview of a plan sharing system by using the intelligent gateway approach Nao (2014) 62

6.1 Cloud-enabled Nao architecture 63
6.2 Test environment for the following experiments 64
6.3 Experiments to manipulate the Nao robot 65
6.4 Experiments to navigate the Nao robot 66
6.5 Experiments to translate text-to-speech 67
6.6 Creating an abstract plan to manipulate the gripper 67
6.7 Creating an abstract plan to manipulate the gripper 68
6.8 Download-client 69

8.1 Nao-TurtleBot Cooperation 74
8.2 Sketch of a stepwise search space extension Nao (2014) 75

List of Tables

5.1 Packages of the Nao-stack 53
5.2 Nodes of the Nao-ROS-stacks 54
5.3 CRAM-Core packages 54

List of Algorithms

5.1 Downloading OWL-description of a high-level plan . . . 51
5.2 Downloading missing object models . . . 52
5.3 Translating OWL-Plan to CPL . . . 52
5.4 Exploring engine . . . 52
5.5 Main function of the transformation engine . . . 55
5.6 Functionality of a process module . . . 55
5.7 Low-level function to close the gripper of the Nao robot . . . 57
5.8 Low-level function to open the gripper of Nao . . . 57
5.9 Low-level function to move the head of Nao . . . 58
5.10 Low-level function to move the arms of Nao . . . 58
5.11 Low-level function to move Nao to a stable position . . . 58
5.12 Low-level function for walking . . . 60
5.13 Low-Level function translate text to speech . . . 61
5.14 Creating High-Level Plans . . . 62
6.1 Manipulate the joints of Nao by calling high-level commands . . . 65
6.2 Navigates the Nao robot to the particular position . . . 66
6.3 Translates text-to-speech . . . 66

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 10.02.2014 Florian Johannßen