Helsinki University of Technology Automation Technology
Series A: Research Reports No. 34

Espoo, December 2009

CONFIGURATION OF SKILLED TASKS FOR EXECUTION IN MULTIPURPOSE AND COLLABORATIVE SERVICE ROBOTS

Mikko Heikkilä

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Electronics, Communications and Automation for public examination and debate in Auditorium AS1 at Helsinki University of Technology (Espoo, Finland) on the 12th of December, 2009, at 12 noon.

HELSINKI UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, COMMUNICATIONS, AND AUTOMATION
DEPARTMENT OF AUTOMATION AND SYSTEMS TECHNOLOGY

Distribution:

Helsinki University of Technology
Faculty of Electronics, Communications, and Automation
Department of Automation and Systems Technology
P.O. Box 5500
FIN-02015 TKK
FINLAND

E-mail: Mikko.Heikkila@tkk.fi
Tel. +358 9 470 25151
Fax. +358 9 470 23308

ISBN 978-952-248-245-7 (print)
ISBN 978-952-248-246-4 (pdf)
ISSN 0783-5477

Yliopistopaino
Helsinki 2009

Available on the internet at http://lib.tkk.fi/Diss

ABSTRACT OF DOCTORAL DISSERTATION	HELSINKI UNIVERSITY OF TECHNOLOGY P.O. BOX 1000, FI-02015 TKK http://www.tkk.fi

Author Mikko Heikkilä

Name of the dissertation
Configuration of skilled tasks for execution in multipurpose and collaborative service robots

Manuscript submitted 14.9.2009	Manuscript revised 11.11.2009
Date of the defense 12.12.2009	
☒ Monograph	☐ Article dissertation (summary + original articles)

Faculty	Faculty of Electronics, Communications, and Automation
Department	Department of Automation and Systems Technology
Field of research	Service Robotics
Opponent(s)	Professor Erwin Prassler
Supervisor	Professor Aarne Halme
Instructor	Professor Aarne Halme

Abstract

Several highly versatile mobile robots have been introduced during the last ten years. Some of these robots are working among people in exhibitions and other public places, such as museums and shopping centers. Unlike industrial robots, which are typically found only in manufacturing environments, service robots can be found in a variety of places, ranging from homes to offices, and from hospitals to restaurants.

Developing mobile robots working co-operatively with humans raises not only interaction problems but problems in getting tasks accomplished. In an unstructured and dynamic environment this is not readily achievable because of the high degree of complexity of perception and motion of the robots. Such tasks require high-level perception and locomotion systems, not to mention control systems for all levels of task control. The lowest levels are controlling the motors and sensors of the robots and the highest are sophisticated task planners for complex and useful tasks. Human-friendly communication can be seen as an important factor in getting robots into our homes.

In this work a new task configuration concept is proposed for multipurpose service robots. The concept gives guidelines for a software architecture and task managing system. Task configuration process presents a new method which makes it easier to configure a new task for a robot. The idea is the same as when a person tells another how a task should be performed. Novel method for executing tasks with service robots is also presented. Interpretive execution, keeping the focus on only one micro task at a time, makes it possible to modify plans during their execution. Multimodal interaction is important feature to provide collaboration between humans and robots. Multimodal interaction reduces the workload of the user by administering task configuration and execution. A novel solution for using multimodal human-robot interaction (HRI) as a part of the task description is presented.

This thesis is a case study reporting the results when developing a task managing (from configuring to execution) platform for multipurpose service robots and studying its performance and use with several test cases. The platform that was developed has been implemented with the WorkPartner multipurpose service robot. The structure and operation of the platform have proved to be useful and several tasks have been carried out successfully.

Keywords service robots, task control, human-robot interaction

ISBN (printed) 978-952-248-245-7	ISSN (printed) 0783-5477
ISBN (pdf) 978-952-248-246-4	ISSN (pdf)
Language English	Number of pages 153
Publisher Helsinki University of Technology, Department of Automation and Systems Technology	
Print distribution Helsinki University of Technology, Department of Automation and Systems Technology	
☒ The dissertation can be read at http://lib.tkk.fi/Diss/	

VÄITÖSKIRJAN TIIVISTELMÄ	TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi

Tekijä Mikko Heikkilä

Väitöskirjan nimi
Taitoa vaativien työtehtävien määrittäminen sekä suorittaminen monikäyttöisissä ja yhteistyökykyisissä palveluroboteissa

Käsikirjoituksen päivämäärä 14.9.2009	Korjatun käsikirjoituksen päivämäärä 11.11.2009
Väitöstilaisuuden ajankohta 12.12.2009	
☒ Monografia	☐ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)

Tiedekunta	Elektroniikan, tietoliikenteen ja automaation tiedekunta
Laitos	Automaatio- ja systeemitekniikan laitos
Tutkimusala	Palvelurobotiikka
Vastaväittäjä(t)	Professori Erwin Prassler
Työn valvoja	Professori Aarne Halme
Työn ohjaaja	Professori Aarne Halme

Tiivistelmä

Useita monikäyttöisiä liikkuvia palvelurobotteja on esitelty viimeisen kymmenen vuoden aikana. Osa näistä roboteista työskentelee ihmisten keskuudessa messuilla tai muilla julkisilla paikoilla, kuten museoissa tai kauppakeskuksissa. Toisin kuin teollisuusrobotit, jotka ovat vain teollisen tuotannon tehtävissä, palvelurobotit voivat olla erilaisissa paikoissa kodeista toimistoihin ja sairaaloista ravintoloihin.

Liikkuvien robottien kehitystyössä keskeiset ongelmat ovat yhteistyö ihmisten kanssa sekä tapa, jolla työt saadaan tehdyiksi. Jäsentymättömässä ja muuttuvassa ympäristössä nämä ongelmat eivät ole vielä ratkaistavissa, koska siihen vaadittaisiin roboteilta nykyistä korkeamman tason ympäristönhavainnointia ja liikuntakykyä. Myös robotin ohjausjärjestelmien tulee olla erittäin kehittyneitä. Tämä tarkoittaa kehittyneisyyttä aina alimman tason moottoreiden ja sensoreiden ohjauksessa, mutta myös hienostuneita tehtävän suunnittelujärjestelmiä vaadituille käyttökelpoisille työtehtäville. Ihmisystävällinen kommunikointi on myös erittäin tärkeä tekijä, jotta hyväksymme robotit koteihimme.

Tässä työssä ehdotetaan uudenlaista tehtävien määrittelykonseptia monikäyttöisille palveluroboteille. Konsepti antaa ohjesäännöt ohjelmiston arkkitehtuurille ja tehtävien hallintajärjestelmälle. Tehtävien määrittelyyn esitetään menetelmä, joka helpottaa uusien tehtävien antamista roboteille. Menetelmä perustuu tapaan, jossa ihminen kertoo toiselle, kuinka jokin tehtävä tulee tehdä. Työssä esitetään myös uusi menetelmä sille, kuinka työtehtävät ajetaan robotissa. Tulkattava ajotapa, jossa tehtävä ajetaan vain yksi osatehtävä kerrallaan, mahdollistaa koko tehtävän muuttamisen myös ajon aikana. Multimodaalinen kommunikointi on tärkeä ominaisuus, kun halutaan ihmisten ja robottien yhteistyötä. Multimodaalisuus pienentää käyttäjän työkuormaa sekä tehtävien määrittelyssä että niiden ajamisessa. Työssä esitetään uusi tapa käyttää multimodaalista käyttöliittymää osana tehtävien määrittelyä.

Tämä työ on ns. case-tutkimus, jossa selostetaan tulokset tehtävien hallintajärjestelmän kehittämisestä palveluroboteille. Lisäksi esitetään tulokset kehitetyn järjestelmän käytöstä ja tehokkuudesta useissa testitehtävissä. Järjestelmä on testattu käyttäen tutkimusalustana WorkPartner-palvelurobottia, ja toiminta on osoitettu käyttökelpoiseksi..

Asiasanat palvelurobotti, tehtävän ohjaus, ihminen-robotti kommunikointi

ISBN (painettu) 978-952-248-245-7	ISSN (painettu) 0783-5477
ISBN (pdf) 978-952-248-246-4	ISSN (pdf)
Kieli englanti	Sivumäärä 153

Julkaisija Teknillinen korkeakoulu, Automaatio- ja Systeemitekniikan laitos

Painetun väitöskirjan jakelu Teknillinen korkeakoulu, Automaatio- ja Systeemitekniikan laitos

☒ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/

Preface

This study was conducted at the Automation Technology Laboratory of Helsinki University of Technology during the years 2003-2009. The main part of the work was carried out within a research project called WorkPartner, financed by the Finnish Funding Agency for Technology and Innovation (Tekes). Furthermore, the author received scholarship from the Emil Aaltonen Foundation. The work was eventually finalized in 2009 at the Centre of Excellence in Generic Intelligent Machines, which is partly funded by the Finnish Academy of Sciences. I would like to thank all these sources for their valuable financial support.

I wish to express my sincere gratitude to Professor Aarne Halme, head of the Automation Technology Laboratory, for his encouragement, inspiration, and expert guidance during this work, and, in particular, the freedom to concentrate on this work without having to worry too much about other duties in the laboratory.

Special thanks go to Adjunct Professor Mika Vainio for offering his expertise and assistance during the project. Thanks also to other colleagues, especially Dr. Jari Saarinen, MSc. Tapio Taipalus, and MSc.Sami Terho, who were testing the developed system in three-shift rotation to get the work done. Thanks too, to Dr. Teppo Pirttioja, who encouraged me when the work felt heavy. My thanks are further due to the whole staff of the Automation Technology Laboratory for creating a friendly and inspiring atmosphere for the work.

I thank my preliminary examiners Professor Miguel A. Salichs and Dr. Ilkka Kauppi for their reviews and valuable suggestions on the thesis.

Finally my warmest thanks go to my wife Paula and our children Juuso, Pihla and baby for their love, patience, and support, which gave me the ultimate motivation to complete the work. I would also like to thank my parents and my siblings, who provided support in many ways for all these years. Thanks to my mother and father-in-law, who helped, especially with childcare, when work took its time.

Espoo, 2009

Mikko Heikkilä

Contents

List of Figures

List of Tables

List of Abbreviations

3D	Three dimensional
A/D	Analog/digital measurement
ADE	Architecture Development Environment for Virtual and Robotic Agents
AGV	Automatic Guided Vehicle
AMR	Autonomous mobile robot
CAN	Controller Area Network
Carmen	Carnegie Mellon Navigation Toolkit
CDL	Configuration Description Language
CLARAty	Coupled-Layer Architecture for Robotic Autonomy
CURL	The Cambridge Robot Language
DARPA	Defense Advanced Research Projects Agency
DCI	Distributed Collaboration and Interaction system
DoF	Degree-of-Freedom
FSM	Finite State Machine
GenoM	Generator of Modules
GIM	Generic Intelligent Machines
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HRI	Human-Robot Interaction
HRI/OS	Human-Robot Interaction Operating System
ICE	Internet Communication Engine
IDC	Intelligent Data Carrier
ILMR	Intermediate Language for Mobile Robots
JAUS	Joint Architecture for Unmanned Systems
KAoS	Knowledgeable Agent-oriented System
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
LED	Light-Emitting Diode
MCA2	Modular Controller Architecture version 2

Mini-ITX	Standard for motherboard of PCs
MT	Micro Task
NASA	National Aeronautics and Space Administration
OROCOS	Open Robot Control Software
PC	Personal Computer
PC/104	Standard for PC-cards
PLEXIL	Plan Execution Interchange Language
PTU	Pan-Tilt Unit
QNX	Realtime operating system
ROCI	Remote Objects Control Interface
RoSta	Robot Standards and Reference Architectures
SIWE	Simultaneous Instruction Writing and Execution Method
TCA	Task Control Architecture
TDL	Task Description Language
UE	Universal Executive
URBI	Universal Real-Time Behavior Interface
VPL	Visual Programming Language
WLAN	Wireless Local Area Network

Glossary

Collaboration
Collaboration is defined as when humans and robot(s) work together toward common goals. However, a human partner has the final word on what to do and how to do it.

Task
A task is defined as a common noun for a real-world task, for example, “clean yard” or “guard house”. Executing the task means movements in an environment and sensing features. See *skilled task* which is more specific kind of task.

Skilled Task
Skilled tasks are real-world tasks which need orderliness and continuous perception during their execution. These tasks are usually executed in an unstructured environment, such as a yard, a parking lot, a park, etc. Tasks also need to be programmable by the user without extensive user training, and they have to be adaptable to varying situations. In other words, skilled tasks are non-trivial actions that may require learning or training before successful execution.

Task Description
The idea of what the task of the robot will be. A human feeds the task description to the robot.

Task Configuration
Process of defining the task description.

Task Configurator
A person who configures the task description.

Plan
A plan is the result of the task configuration process for the robot. A plan has a logical sequence of required actions needed for the desired task. This includes not only a correct sequence of operational micro tasks but also exception handling.

Micro Task (µTask)
A micro task is the lowest-level building block for skilled tasks. A micro task will always be executed as a whole, and the feedback will be used as additional information for the later micro tasks. Typical examples might be gotoxy, findobject etc.

eTask
Software module of micro task.

1 Introduction

1.1 Background of the Dissertation

The era of robots started in 1962, when the first industrial robot, called UNIMATE, was introduced by Joseph F. Engelberger [IEEE]. After a rather slow start, the growth was nearly exponential for many years and, although the field has matured somewhat during the past 10 years or so, currently there are more than one million industrial robots operating worldwide [IFR Statistical Department]. No wonder, because various industries provided numerous applications that were perfectly suitable for industrial robots. These machines were, and still are, very effective in various tasks demanding repetitive actions in fixed static locations. The famous 3Ds of robotics (i.e., Dirty, Dangerous, and Dull) were guiding the field. In particular, the automotive and piece goods industries were typical areas where these robots were actively installed.

Although the first primitive autonomous mobile robots (AMRs) were introduced as early as in the 1970s, the real breakthrough took place in the 1980s. By the end of the ’70s factory floor-based automatic trucks, Automatic Guided Vehicles (AGVs), which had been around for some time (the first primitive model was introduced in 1953) had reached a stage where their capabilities had expanded considerably. These vehicles, which earlier had been mere towing trailers, were now capable of performing much more complicated operations as part of the new and emerging paradigm called Flexible Manufacturing Systems. Inspired by the success of AGV systems, and aided by the fast development in electronics, sensors, and computers, several research groups started to invest their resources heavily in mobile robotics.

Mobility brought with it a series of new challenges related to topics such as safety, energy resources, and the perception of the environment. Unlike industrial robots, which work in structured static environments, mobile robots had to be able to survive in unstructured dynamic environments teeming with uncertainty and randomness. Despite all these challenges and others too, the development efforts of mobile robots have increased year after year and the trends also show very strong continuous growth for the coming years. Unlike industrial robots, which are typically found only in manufacturing environments, service robots can be found in a variety of places, ranging from homes to offices, from hospitals to restaurants. It can be stated that service robots are here to serve us in various tasks involved in everyday life. The term ‘service robot’ could be defined as a mobile robot which operates semi- or fully autonomously, performing services useful to the wellbeing of humans and/or equipment, excluding manufacturing operations.

Nowadays, service robots are still in their childhood stage. There are some service robots on the consumer market, including entertainment robots, vacuum cleaners, and lawnmowers, which are simple enough applications, but service robots are mainly used in professional applications such as medical and underwater work and surveillance. The largest application area is the military one, where robots perform various tasks, including bomb disposal, monitoring, and reconnaissance, thus improving the safety of humans. It is also worth mentioning that some high-profile events, such as DARPA's Grand Challenge and Urban Challenge, are funded by the military and aim to further develop unmanned military vehicles. Regardless of the strong military connection, it is clear that in the near future service robots will also make the lives of us civilians easier by taking care of boring everyday tasks such as cleaning, laundering, cooking, driving a car etc. However, there are still many challenging steps to be taken before that is reality on a larger scale. For example, it is essential that proper service robotic laws and standards are globally uniform and widely accepted. These regulations are needed to make sure that robots are safe for humans and for the environment. Though service robots are still in their infancy, they are growing rapidly and we are witnessing perhaps the most exciting and promising robot evolution of all time.

1.2 Research Problem and Aims of the Study

Several highly versatile mobile robots have been introduced during the last ten years. Some of these robots are working among people in exhibitions and other public places such as museums and shopping centers. The most famous service robot in the world is currently, without any doubt, *Asimo*, a very sophisticated humanoid robot created by Honda. Asimo can walk, run, and climb stairs. It understands preprogrammed gestures and spoken commands but also recognizes voices and faces. It has arms and hands, so it can, for example, turn on light switches, open doors, carry objects, and push carts. WorkPartner is one example of a multipurpose service robots. *WorkPartner* can be imagined as a janitor of the future. It is centaur-like robot for outdoor use. *Minerva* is another famous example of a versatile service robot. It is a talking tour-guide robot which guides people through public places such as a museum, explaining simultaneously what they see along the way. Both of these robots are state-of-the-art quality, and although both are designed for indoor usage, they demonstrate that navigation is no longer the main problem in mobile robotics. Outdoor localization (and thus navigation too) methods rely heavily on GPS and high-quality gyros and accelerometers. Indoor navigation is very much based on probabilistic estimation techniques (if no supporting infrastructure is available) thanks to the increased

computation power available on board even a small-scale robot and to improved and refined estimation algorithms.

One of the current fundamental challenges (there are still a few around) in this research field is how to make a robot understand what it perceives, i.e., how it processes the huge amount of raw data coming from various sensory systems, extracts necessary and valid information, and, finally, defines the proper context for it in order to synthesize a valid representation of the surrounding world. This procedure is inherently easy for us humans, mainly on the basis of our previous experiences, but really challenging for mobile robots. The main focus of the thesis is not perception but there are some other persons in our research group (the Finnish Academy's Centre of Excellence – Generic Intelligent Machines) who are working actively on that topic.

Instead of tackling the problems related to perception, this thesis deals with the big challenge of how to manage real tasks in real environments. Managing tasks means the overall task-handling process in which tasks are configured, executed, and monitored. Managing the complexity of not only the tasks but also the environment and the robot platform is far from being trivial and requires specific control architecture to master this. There are several robotic system architectures that have been developed for the task-oriented control of multi-purpose service robots. Some widely known examples include architectures such as Task Control Architecture (TCA), MissionLab, CLARAty, and HRI/OS. These are presented and analyzed in Chapter 2. Most of the control architectures include methods for configuring tasks and description languages for defining tasks for the robots. The description language specifies the behavior of the robot during the performance of the task. Using the description language, the user defines the plan of what the robot should do. Some of these available languages used for defining tasks are also presented briefly in Chapter 2.

Multipurpose service robots developed for consumer markets are the platforms focused on in this thesis. These kinds of robots are not available today but will be on the market in the future. Usability will be one of the biggest challenges to service robots in our homes. Fluent interaction in many everyday situations is a large step to be taken to get robots into our houses and yards. During a robot's life span its knowledge of the environment and skills should also improve, just as human children improve their skills. Everyday working areas will became more and more familiar and task sequences will also be fitted to this well-known environment. Finally, the robot will know every corner of its home yard and all the important objects in it. The situation is totally different when the environment is new not only to the robot but also to its user. For

example, working in a familiar house yard and an unknown environment are very different, and this has an effect on interaction and task management. In both of these cases the usability of the robot should be on a high level. Ideally, the user should not face a situation where he has to take the manual to solve problems.

Tasks performed with a service robot are defined as skilled tasks. Skilled tasks are real-world tasks which need orderliness and continuous perception during their execution. In other words, skilled tasks are non-trivial actions that may require learning or training before successful execution. These tasks are usually executed in an unstructured environment, such as a yard, a parking lot, a park, etc. Tasks also need to be programmable by the user without extensive user training, and they have to be adaptable to varying situations. The plan of the task remains constant during the task execution rounds, but the real-world actions vary. For example, the task "clean yard" could be a static plan, but the cleaning work differs because of changes in the environment. Perception of the environment and adaptation are key elements in skilled tasks. There is a huge difference from tasks which are always geometrically the same, such as the tasks of industrial robots. Figure 1 presents differences between the skilled task and traditional task executed in industry.

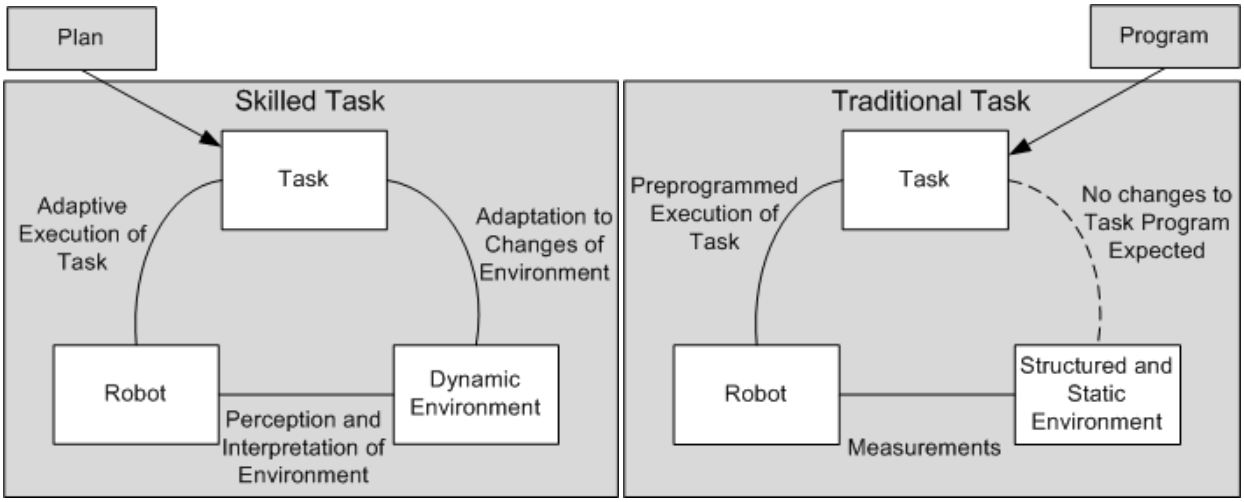

Figure 1. Skilled Task vs. Traditional Task.

There is a huge gap of methods how to configure tasks for the service robots. Configuration of tasks can be called also teaching of task. Teaching a task means providing or configuring a plan of how to use available resources. The operator makes the plan interactively with the aid of the user interface by utilizing suitable input functionalities, in most cases in the first phase graphics

and “imitation” through teleoperation. The thesis presents a novel task configuration concept to fill this gap.

The problem from the point of view of a future user, with all of these examples and with many others too, is the way in which tasks are configured to them. Most of the cases require programming skills and knowledge of the internal software architecture. The roots of these state-of-the-art examples can be found from computer science and are thus not directly very suitable for a wider user population. Instead, looking back at the history of human society. The master and apprentice model was a very popular way to teach skills to younger students/workers in the past. The idea was that the master taught the basics of the skill by taking the lead and then instructed the apprentice during his own attempts. Nowadays there are not so many masters and their apprentices in modern societies, mainly because machines have taken such a great role in most of the common activities that were earlier related to master- apprentice-based industries. As robots are getting more and more versatile and are equipped with some level of cognitive capabilities, it is worth considering the case where the apprentice would be the robot and the master would be the human user. Bearing that in mind, the aim of this study is to propose a new task configuration concept for multipurpose service robots. The concept gives guidelines for a software architecture and task managing system that is reported later on in this thesis.

Figure 2. WorkPartner centaur-like service robot.

The research platform for this study is WorkPartner (in figure 2), which is a modern service robot test bed. It is targeted for light outdoor applications like property maintenance, gardening and light forestry tasks. The robot can be considered as being like the janitor of the future. The robot is being developed in the Automation Technology Laboratory of Helsinki University of Technology (TKK). The hybrid locomotion system provides four motorized wheels with articulated steering for easy terrain. Four 3DoF legs make it possible to perform walking and control the body attitude more freely. The humanlike torso includes 2 DoF waist, two 5 DoF hands with grippers and a 2DoF camera/laser pointer head. The total weight of *WorkPartner* is about 270kg, the payload is about 50 kg and it can handle objects up to 10 kg. The hybrid energy system is based on small batteries (48V) and a supporting unit, which can be either 1kW motor – generator or 500W alkalic fuel cell. All actuators are electric.

WorkPartner was designed to be a multipurpose service robot. As a partner it should be capable of performing several different tasks either alone or in continuous cooperation with its master. The goal is that most of the interaction is human like conversation by speech and gestures to minimize the operator hardware. However, different traditional and novel interfaces are also developed to help the common understanding between robot and operator, especially in teaching and teleoperation situations. More details of the robot are presented in Chapter 6.

1.3 Contribution of the Dissertation

At the beginning of this research work there were drivers and interfaces available for sensors and actuators and control software for several subsystems of the test robot, called WorkPartner. These subsystems included navigation, perception, and motion control. It was assumed that there would be enough processing power available. The missing factor was the structure that was needed to configure and control new tasks via a multimodal user interface. That process was guided with the following research questions.

- How can the service robot be made to perform the desired work tasks so easily that the teaching/task configuration processes look natural?
- How can humans and robots converse about abilities, goals, and achievements in order to work effectively together?
- How can task execution be monitored and controlled in an explicit way?

Figure 3 depicts the main topics presented in this dissertation. The key element is a continuous interaction between humans and the robot at all stages during the task configuration. This can be imagined as a process where the master guides an apprentice from receiving his first advice to becoming a fully-trained professional. The large circles in Figure 3 present processes for covering the task configuring and execution stages. The intersections present proposed methods for reaching the targets of the thesis.

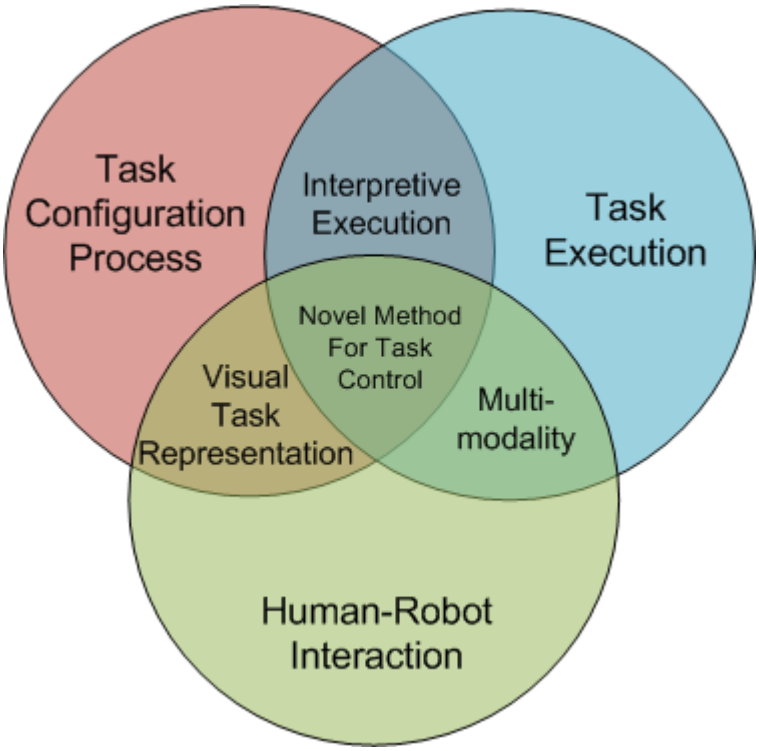

Figure 3. Three-way diagram depicting the contribution of the dissertation.

The first scientific contribution is a task configuration process presenting a new method which makes it easier to configure a new task for a robot. The idea is the same as when a person tells another how a task should be performed. This method could be compared to the process where a human writes an instruction guide. In the method a user interprets the stages of the desired task and configures a preliminary plan for the work task using a visual tool. Visual programming languages (VPL) are a common name for programming languages defined as a visual diagram instead of text. The structure of a work task is based on an activity diagram which offers a powerful method to configure step-by-step work flows. The building blocks of the preliminary plan are called micro tasks in this thesis.

For a fluent task configuration process the user should have knowledge of the abilities of the robot in use but also knowledge of the task under construction. This work gives one possible way to combine the movements of robots with their perception.

The second scientific contribution of this thesis is a novel method for executing tasks with service robots. Interpretive execution, keeping the focus on only one micro task at a time, makes it possible to modify plans during their execution. Modifications are possible not only for the structure of the task but also for the parameters defining the internal operation of the micro tasks. If changes are possible online it is an important feature, especially for the task configurator, during tests of tasks that have just been created. This feature makes it possible to create an automated optimization system for task configuration in the future.

The third scientific contribution of this thesis is a novel solution for using multimodal human-robot interaction (HRI) as a part of the task description. Multimodal interaction is important feature to provide collaboration between humans and robots. Multimodal interaction reduces the workload of the user by administering task configuration and execution. Several modalities are presented in this dissertation, including speech, gestures, pointing, and direct teleoperation, among others. Because of the integration of communication channels, the task description work is simpler. The use of external markers (called signs) as a part of task description is a good way to lighten the commissioning stage of the task. A good example of using signs is traffic signs. A road user receives sufficient information from the traffic signs, for example, following arrow-shaped signs. Signs are also easy to use during road construction by using only the user's cognitive capacity. The same idea could be transferred to service robotics. For example, a human user could walk to the designated work area and put up signs to define exact routes and the area. During the task initialization the user needs only to tell the robot that the signs are used for definition. No mouse clicks or drawings or textual input are needed.

1.4 Outline of the Thesis

This thesis is a case study reporting the results when developing a task managing (from configuring to execution) system for multipurpose service robots and studying its application and use with several test cases.

The organization of the dissertation is as follows.

Chapter 1: Introduction. The introduction gives a short overview of the history and future of robotics. It also describes how the topic for this dissertation was chosen.

Chapter 2: Related work. This chapter reviews the current state of the art in the field of mobile robots and task control to robots.

Chapter 3: Configuring Tasks: Master and Apprentice. This chapter describes a novel method for configuring tasks to service robots. The master-apprentice model was a very popular way to teach skills to younger students/workers in the past and the same method is proposed as a method to teach tasks to robots. A visual programming language is proposed as a configuring method for a preliminary work task plan.

Chapter 4: Task Control Platform. This chapter presents a software architecture developed to fulfill the software requirements to solve the task configuring problem. A task composition language for skilled tasks is presented. An interpretive task execution method is proposed to increase the adaptability of service robots. A modular task structure is presented to provide decentralization to get enough processing power, for example. The structure also enables multi-robot solutions to be implemented in a better way.

Chapter 5: Configuration of Tasks for Multimodal Service Robot. This chapter presents a multimodal user interface to reduce the workload of the user during the task configuring process. Several modalities and their suitability for applications are presented. Using external signs as a part of the task description is presented.

Chapter 6: Experimental Verification. One test vehicle is used to verify the system that was developed in real test cases. The test vehicle used, the WorkPartner wheel-legged robot, is explained in detail.

Chapter 7: Conclusions. A summary of the research and recommendations for future work are presented in the last chapter.

1.5 Author's Contribution within the Research Group

The major part of this work was performed in the WorkPartner project. The project took place in three phases in 1998-2005. The project has been published in several publications [Halme et al. 2003], [Suomela, Halme 2004], among others. The WorkPartner robot was used as a test platform in the work documented here. The thesis work was conducted during the period 2003-2009 in a dynamic research group (from five to ten members).

The author's principal contribution to the group was designing and developing the task control platform and task description language as a part of the robot's software architecture. The integration of a multimodal user interface is also one of the key topics of the author's work. The other contributions were from Jari Saarinen, who implemented the final version of the database and mission-handling interface and also parts of the conceptual design of the system, and Sami Terho, who contributed to the implementing of the perception system. Additionally, Tapio Taipalus was responsible for the voice recognition system and was a co-author in the development of the task description language. Details of the system are presented in Chapter 4.

1.6 Relation to Earlier Publications

Parts of the work reported in this thesis have been published in [Heikkilä et al. 2006] and [Terho et al. 2006]. The results of the WorkPartner project have been published in several publications, for example in [Halme et al. 2003] and [Suomela, Halme 2004]. Moreover, in PhD. theses, studies of the intermediate languages for mobile robots [Kauppi 2003] and studies of human-robot interaction [Suomela 2004] are related to the thesis reported here.

2 Related Work

2.1 Introduction

Some of the dreams of robotic scientists, also seen in Hollywood movies, are easily described: robots doing hard and/or tedious housekeeping work, robots playing with the children, helping elderly people at home in their daily tasks, robots assisting in the therapy of patients in hospitals. Robots are also in factories, manufacturing and lightening the work of humans. Naturally, these robots – whether they assist us at home or at work – should be instructed as effectively and intuitively as a human [Prassler et al. 2005].

Service robots which work interactively with humans form the next generation in robotics. Since the development of industrial robots, which work mainly in factories, the priority has been focused on the development of robots or intelligent machines that can perform non-industrial tasks in non-industrial environments, such as work sites, urban environments, offices, and homes. Space and undersea technology, as well as nuclear power technology, will also use service robots extensively. Most of the world's research activity in robotics is presently in this area. The problems to be solved are considerably more challenging than in industrial robotics.

It is often imagined that service robots could work autonomously and that the operator just gives them tasks to do and supervises the work. The interface that is needed mainly depends on the task of the robot.

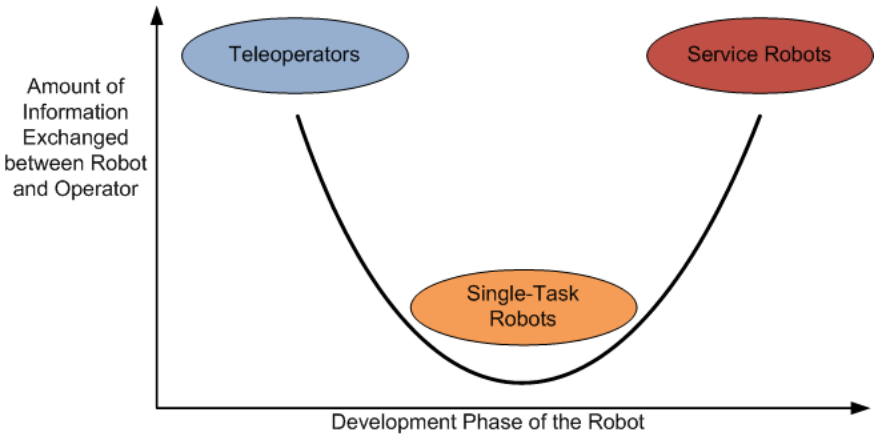

Figure 4. Amount of information exchanged between operator and robot. Adopted from [Suomela, Halme 2004].

A versatile service robot is much more complicated from the interface point of view, as can be seen in Figure 4. If the robot can do all the tasks it is planned for and it knows the environment and possible objects the interface can be very simple. The operator just gives the task by speech or in some other way. If there are limitations to the robot's knowledge of the aforementioned parameters, the amount of information that needs to be transmitted between the robot and its operator increases rapidly. With an increase in information, the complexity of the interface also usually increases. Most of the information could be transferred via speech, as humans do, but because of the limited speech-processing and machine cognition capabilities, more technical interfaces have to be used.

Developing mobile robots that work co-operatively with humans raises not only interaction problems but also problems with getting tasks accomplished. In an unstructured and dynamic environment this is not readily achievable because of the high degree of complexity of perception and motion of the robots. Such tasks require high-level perception and locomotion systems, as well as control systems for all the levels of task control. The lowest levels control the motors and sensors of the robots and the highest are sophisticated task planners for complex and useful tasks.

Human-friendly communication is seen as a method of interaction which is similar to interaction between humans. There are several different kinds of communication methods in human-robot interaction. Nowadays, the most common interaction method has its roots in computer science. Human-computer interfaces facilitate communication, assist in the exchange of information, and process commands and controls, among many additional interactions. Visual or text-based displays on a computer screen are the easiest way to make a user interface for robots. Speech, gestures, and haptics are possible interaction choices to study human-robot interaction [Goodrich, Schultz 2007].

2.2 Relation to Other Fields

Computers and computer software are the hearts (or brains) of service robots nowadays. Even though traditionally computers are static machines on a table, the outcomes of computer science have given a lot to robotics. HCI researchers have developed methodologies, design principles, and computing metaphors which support the science of robotics as well.

Service robots are composed of many different parts, both hardware and software. The software of service robots is actually a very large software system. The agent technology area is quite near in that point of view and Jadex ([Pokahr, Braubach & Lamersdorf 2003] and the [Jadex

]) software framework is one example. The agent paradigm allows a system to be viewed as being composed of autonomous interacting entities which pursue their own goals and act in a rational manner. Interestingly, the agent metaphor fits the demands of distributed and concurrent systems perfectly. In a multi-agent system each agent is autonomous from other agents and hence can potentially perform concurrently with others. In addition, agents communicate via messages and do not need a shared memory environment. This kind of software system has several correspondences to robotics, but because of its roots in a web-based system it does not answer all the challenges raised by human-robot interaction.

2.3 Software Architectures for Service Robots

Robot functions such as locomotion, navigation, and perception have reached a technical level which has raised the need for the integration of those services. Nowadays, robots are platforms carrying several actuators, sensors, and a lot of computing power. Robots have become large software systems. To manage this complexity, a lot of development work is being done to simplify the controlling of the robots. This presentation of software architectures is organized as follows. Middleware presents some well accepted and used software middleware, such as *Player* etc. This is followed by robot control software architectures such as *MissionLab*. Finally, information is provided about development toolkits and task programming languages to implement executive functions in service robots.

2.3.1 Middleware

There is plenty of work in defining and developing so-called robot middleware. Middleware is the glue software that every robot system developer wants to use to connect software (and hardware) components together. Middleware often includes functions such as inter-process communications, the transparent distribution of resources, low-level abstractions (such as data type), and primitive robot behaviors (such as locomotion, object avoidance etc.).

On the lower level, where the actuators and sensors are, there is a need for a common set of drivers, libraries, and algorithms. A common interface will help to make robot control software easier to reuse and simpler and increase the speed of the development process. *Orca2* [Makarenko, Brooks & Kaupp 2006], *OROCOS* ([Bruyninckx 2001] and [Orocos group]), and *Player/Player2* [Collett, MacDonald & Gerkey 2005] are some examples of reusable robotics software that have been made. These systems are presented in more detail below.

Orca2 provides a means for defining and developing building blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks. Orca2 is a component-based system and represents a robotic software application as a network of communicating independent software components. Each component is typically a binary deployment unit and is a standalone process at runtime. Orca2 is based on the ZeroC Internet Communication Engine (ICE) [Henning 2004] middleware system and also uses ICE's communication mechanisms between the software components. Orca2 works in many operating systems (Linux, Windows, and MacOS X) and programming is possible with several programming languages (C++, Python, Java, and C#, among others). The origin of Orca2 is in the EU-funded OROCOS (Open Robot Control Software) project [Orocos group]. OROCOS has produced programming libraries for robot and machine control. OROCOS is an international effort in Europe that aims at developing a common architecture. The project's aim is to develop a general-purpose free-software modular framework for robot and machine control. It is organized in the form of four C++ libraries: the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library, and the OROCOS Component Library. Supported operating systems are Linux and MacOS X and also Windows with MinGW.

Player/Player2 [Gerkey et al. 2001] is a client /server architecture which allows access to a variety sensor over a network. Applications can be written in any programming language, and require only network access to the sensors. Player supports the plugin drivers and simulation environments Stage [Gerkey et al. 2001] and Gazebo [Collett, MacDonald & Gerkey 2005]. Stage is a robot system simulator in a two-dimensional bitmapped environment. Gazebo is a multi-robot simulator for outdoor environments. Like Stage, it is capable of simulating a population of robots, sensors, and objects, but does so in a three-dimensional world. Player is supported on Linux and under Windows using Cygwin.

There are also other projects proposed as a solution for robot middleware or as a framework. *Microsoft Robotics Studio* [Gates 2006], *MIRO* [Enderle et al. 2001], *OpenRDK* [Calisi et al. 2008], *URBI* [URBI], *Marie* [Cote et al. 2004], and *SmartSoft* [Schlegel, Worz 1999] are some examples, but others also exist.

2.3.2 Control architectures

Control architectures can be defined as the whole control system of the robot. The main idea is that there some kind of “higher levels” of control, for example planners, task managing systems etc. Often control architecture has its own middleware offering lower-level control functions. A number of robot control architectures have been proposed for managing service robots, but also to provide human-robot interaction.

TCA (Task Control Architecture) is one of the earliest task-oriented architectures [Simmons 1994]. It was developed by Reid Simmons at CMU, starting in 1988, and has been extended over a number of years with many additional packages. The idea of TCA is the integration and coordination of perception, planning, and real-time control to achieve a given set of goals (tasks). TCA is not only architecture but it also provides a toolbox for building the communication and control routines of a robot. Under these functionalities TCA could also be presented in the section on development toolkits. The task-level control in TCA includes a library of functions for coordinating so-called task trees. A task tree is a hierarchical decomposition of a complex task into a number of smaller subtasks. The task description language (TDL) [Simmons, Apfelbaum 1998] presented later is a language for constructing and managing the underlying representation of tasks, i.e. the task trees. The interprocess communication (IPC) [Simmons 1994] package is part of TCA but is also widely used in other projects, for example, in Carmen [Montemerlo, Roy, & Thrun 2003] and MissionLab [MacKenzie, Arkin, & Cameron 1997]. IPC features efficient transmission of general C data types, anonymous publish/subscribe and client/server capabilities, and automatic marshalling (serialize) and unmarshalling (de-serialize). IPC invokes user-defined handlers when a message is received, and invokes user-defined callbacks at set intervals. In essence, IPC performs a function similar to a naming service for components; besides providing the means to define message abstractions used for communication over a network, it also encourages extensibility and fault tolerance [Simmons 1994].

One example of software systems for developing missions and work tasks is MissionLab, developed at the Georgia Institute of Technology [MacKenzie, Arkin, & Cameron 1997]. MissionLab is multiagent robotics mission specification and control software. MissionLab is based on Autonomous Robot Architecture [Arkin, Riseman, & Hanson 1987]. The main focus of the platform is on military kinds of plans. One of the key components is called CfgEdit, a graphical tool for building robot behaviors. The user can build complex control structures with the pointing and clicking of a mouse through its graphical user interface.

MissionLab supports the performance of multiple robots both in simulations and actual robotics platforms. Missions are portioned into temporal sequences of discrete operating states with perceptual triggers causing transitions between those states. The Societal Agent theory (developed in [MacKenzie, Arkin, & Cameron 1997]) is used as a basis for constructions of this form. MacKenzie calls active components in MissionLab agents. There are two types of agents: atomic and assemblages. The atomic agents are primitive behaviors, while assemblages are coordinated societies of agents. These agents are specified as:

$$Agent = Behavior(Stimulus)$$

$$Agent = C(Agent_1, Agent_2, \ldots, Agent_i)$$

where C is a suitable coordination operator.

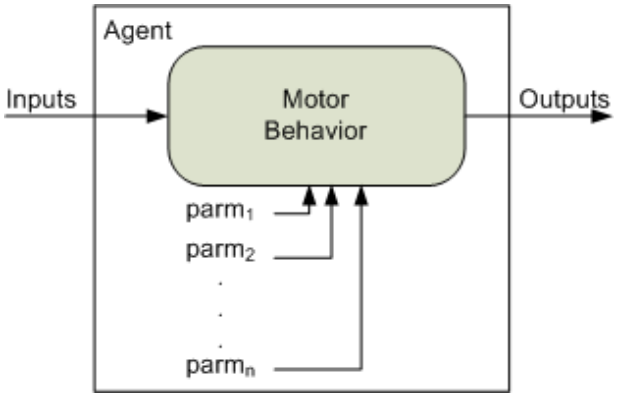

Figure 5. Schematic diagram of an atomic agent. Adopted from [MacKenzie, Arkin, & Cameron 1997].

Configuration is the name for the specification of the components, connections, and structure of the control system for a group of robots. A configuration consists of a collection of agents, communication links between agents, and a data-flow graph describing the structure of the configuration. An example of an atomic agent is presented in figure 5. Figure 6 presents an example of assemblages in a particular configuration. That configuration can be expressed, using:

$$C_5(C_2(C_1(A_1,A_2)),C_4(C_3(A_6,A_7),A_8))$$

Tasks are defined using *the configuration description language* (CDL) instead of the TDL used in TCA [Simmons 1994], HRI/OS [Fong et al. 2006], and CLARAty [Estlin et al. 2001]. CDL is used to specify the construction of primitives, including their instantiations and coordination. Linux is the operating system supported by MissionLab.

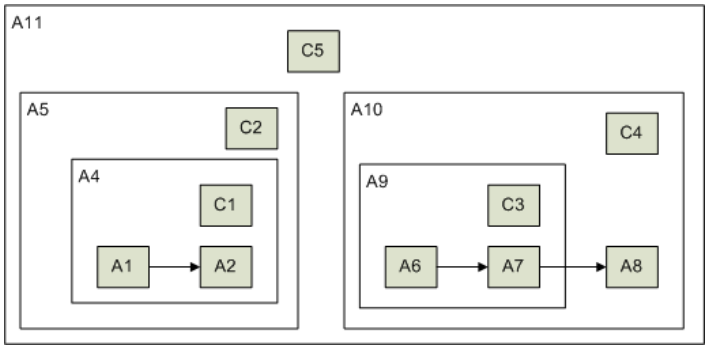

Figure 6. Schematic diagram of a configuration. A_n refers to atomic or assemblage agents and C_n is the corresponding coordinating operator. Adopted from [MacKenzie, Arkin, & Cameron 1997].

Some architectures are focused on interaction between humans and robots, such as HRI/OS from NASA [Fong et al. 2006], KAoS [Johnson et al. 2008], and DCI (Distributed Collaboration and Interaction system) [Martin et al. 2003]. HRI/OS is a framework developed in NASA's peer-to-peer human-robot interaction project [Fong et al. 2006]. For example, in MissionLab the focus is on the missions, but, according to the authors, in HRI/OS the objectives are the collaboration of humans and robots. In the project the goal is to develop an interaction infrastructure that enables humans and robots to communicate and work as partners. HRI/OS has been evaluated with a human-robot team consisted of five different members working in parallel on a simulated work task [Fong et al. 2006]. Direct point-to-point communication is performed using ICE middleware [Henning 2004]. Tasks are implemented using the Task Description Language (TDL) [Simmons, Apfelbaum 1998]. Figure 7 presents the architecture of HRI/OS. The system is agent-based; it incorporates embodied agents (humans and robots) and software agents (see Figure 7). A major difference compared to other architectures is that there is a human as a key member of the system. In HRI/OS there are quite a large number of functional elements matching the purposes of our research, but unfortunately the system has not been opened to developers around the world. In addition the HRI/OS is designed for co-operation between astronauts and the robots. Our research is focused on tasks related to the work of a janitor. Astronauts are highly educated people and they are trained to their tasks in details.

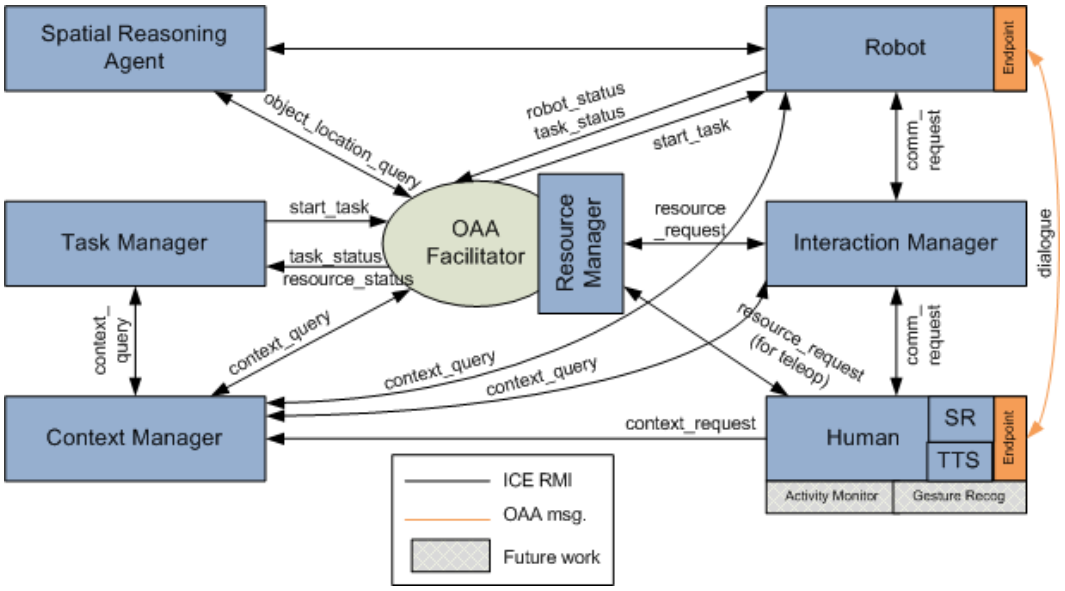

Figure 7. Architecture of the HRI/OS. Adopted from [Fong et al. 2006].

The LAAS architecture [Alami et al. 1998] for autonomous systems is a 3-layer architecture (in Figure 8) with a decision layer (planning and supervision of action), an execution control layer (coordination of actions), and a functional layer (performance of actions). The functional layer of LAAS integrates all the operational functions (hardware control, servo control, and data processing). The functional layer is structured as a set of independent modules which are entities responsible for a physical or logical resource. Modules are generated with GenoM (Generator of Modules), which is, according to the authors [Fleury, Herrb, & Chatila 1994], "a tool for the specification and the implementation of Operating Modules in a Distributed Robot Architecture". GenoM is a collection of development tools, programming libraries, and a development environment. The platform and languages supported are UNIX/Linux/Solaris and TCL, Ruby and C++.

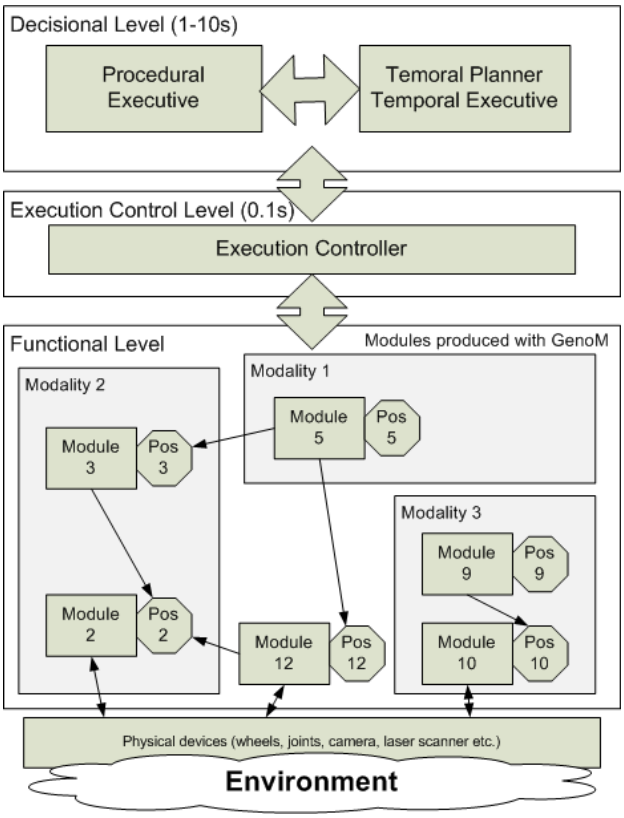

Figure 8. Overview of LAAS architecture. Adopted from [Alami et al. 1998].

CLARAty [Volpe et al. 2001] was developed in the Jet Propulsion Laboratory. CLARAty (Coupled-Layer Architecture for Robotic Autonomy) is a framework that promotes reusable robotic software and it is widely used in robotic programs in NASA. CLARAty is an architecture including specifications for its middleware. For example, it defines interfaces for common robotic functionalities such as pose estimation, navigation, locomotion, and planning. CLARAty also supports multiple robotic platforms. The main application of CLARAty is in NASA's Mars program and rovers to be sent to Mars in the future [Volpe et al. 2001].

CLARAty, on the system level, is organized as a 2-layered abstract machine and is illustrated in Figure 9. The top level is a decision layer, which encapsulates both the planning and execution functionalities of a standard 3-tiered approach. The second level is the functional layer, which is responsible for interfacing all the platform hardware and using its full capabilities/resources [Volpe et al. 2001]. On the functional layer Universal Executive (UE) executes plans, which are configured using PLEXIL (Plan Execution Interchange Language) [Verma et al. 2005]. Despite its two-layer architecture, the concept is quite similar to the LAAS architecture. The combined decision and execution control levels in LAAS are similar to the

decision layer in CLARAty. C++ is used as a programming language in CLARAty and operating systems supported are VxWorks, Linux, and Solaris.

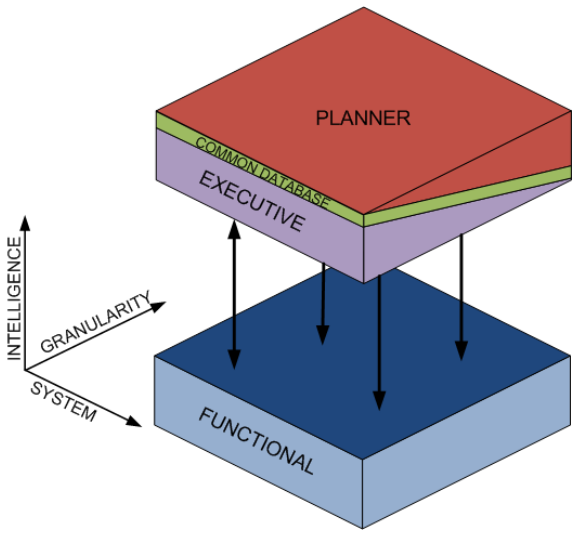

Figure 9. Two-layer architecture of CLARAty. Adopted from [Volpe et al. 2001].

One example of a framework focusing on multi-robot control is ROCI (Remote Objects Control Interface), presented by Chaimowicz et al. [Chaimowicz et al. 2003]. The ROCI architecture consists of several different components: kernel, database, network, module, task, browser, etc. The ROCI kernel is the core control component that manages the network, the database of nodes and services, and the module and task control process in the system. The ROCI browser is used to present information about the network to a human user. There are also other approaches, such as Modular Controller Architecture version 2 [MCA2] and Architecture Development Environment for Virtual and Robotic Agents (ADE) [Andronache, Scheutz 2006].

2.3.3 Implementing executive-level capabilities

There are several task development toolkits and task-level programming languages that have been developed for robots, including manipulators and mobile robots. A task programming language is for implementing executive-level capabilities for service robots.

ILMR (Intermediate Language for Mobile Robots) [Kauppi 2003] is the language which is supported by WorkPartner's architecture and it is used in our approach. ILMR is intended for service and field robots and it acts as an intermediate link between the robot platform and higher-level planner. ILMR supports both the serial and parallel execution of tasks. The strength of ILMR is that it offers an interface for the easier implementation of task elements. For example, coordinated locomotion is implemented on this intermediate level. ILMR does not answer the

research questions of this thesis, for example, regarding task configuration and human-robot interaction, but it offers a good supporting system for the platform that was developed. The execution and interpretive system of ILMR runs on the QNX real-time operating system. The next lines present an example of ILMR. In the example the robot is commanded to follow a path created with the createpath command at a speed of 0.4 m/s. While following the path, the robot avoids obstacles.

```
obsavoid(on)
speed(0.4)
createpath(myroute,1,1,5,1,10,8,13,15,20,25)
followpath(myroute)
```

Another similar kind of language is the System for AUtonomous Specification, Acquisition, Generation, and Execution of Schemata (SAUSAGES) which provides a specification language, as well as run-time execution and monitoring support [Gowdy 1994]. SAUSAGES is used, for example, in MissionLab [MacKenzie, Arkin, & Cameron 1997] and it is a Lisp-based script language tailored for specifying sequences of behaviors for large autonomous vehicles.

PLEXIL (Plan Execution Interchange Language) is a stand-alone language for the hierarchical decomposition of tasks into subtasks [Verma et al. 2005]. The fundamental block of PLEXIL is called a *node*. A hierarchical composition of nodes is called a *plan*. PLEXIL has a syntax that distinguishes the leaf nodes of the task tree/graph from the interior nodes. A node has two primary functional components, a set of conditions that drive the execution of that node, and the content of the node, which specifies what gets done. Conditions are defined for controlling, for example, when a node should start executing or for failure conditions. Following example presents a PLEXIL plan that drives the rover one meter while controlling the safety (Adopted .from [Verma et al. 2005])

```
Node:{
    NodeId: SafeDrive;
    Repeat-until-condition:
        Lookup{“Rover:wheelStuck”}==false;
    NodeList:{
        Node:{
            NodeId: DriveOneMeter
            Command: Rover:Drive(1);
        }
    }
}
```

A graphical interface to PLEXIL plans is being developed, named Luv (The Lightweight Universal Executive Viewer) [Verma et al. 2006]. Luv can be used for executing and debugging plans. Universal Executive (UE) is an implementation and execution environment for PLEXIL. UE runs on VxWorks, UNIX variants, and Linux operating systems.

One example of a high-level task programming language is TDL (Task Definition Language), which is part of TCA, mentioned in the control architectures chapter. TDL is a superset of C++ for specifying tasks. The idea with TDL is to simplify the task-level control of robot programming. Task-level control refers to robot capabilities such as deliberation and reactivity, recovery from exceptions, and resource management. TDL supports task decomposition, the synchronization of subtasks, execution monitoring, and exception handling, to meet the requirements that such robot capabilities demand. [Simmons, Apfelbaum 1998]

The hierarchy of the TDL is presented in Figure 10. The language, originally designed for single robot use, has been extended to facilitate task-level coordination between robots, as well as the possibility of robots spawning or terminating tasks for each other. TDL is also used in HRI/OS [Fong et al. 2006], which is reported above.

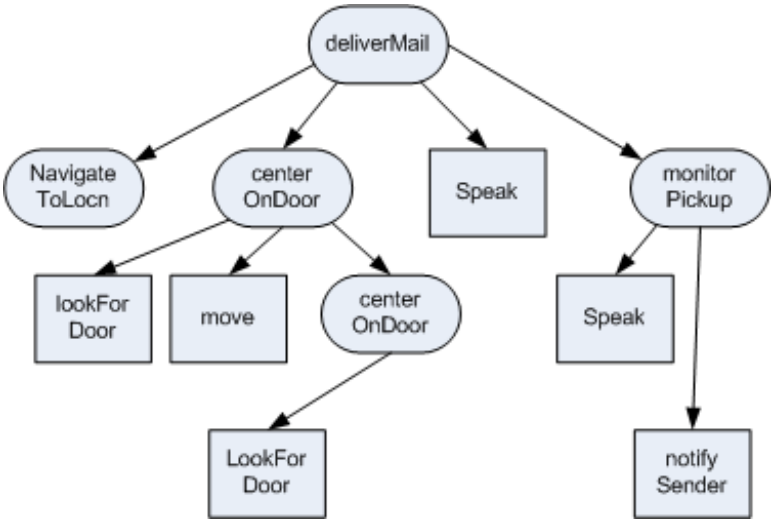

Figure 10. Example task tree of TDL. Adopted from [Simmons, Apfelbaum 1998].

Task tree presented in Figure 10 is a representation of following simplified example of TDL (Adopted from [Simmons, Apfelbaum 1998]). Example presents delivering mail to some room involving navigation, centering the doorways and waiting for the mail to be taken.

```
Goal deliverMail (int room)
{
    double x, y;
    getRoomCoordinates(room, &x, &y);
    spawn navigateToLocn(x, y);
    spawn centerOnDoor(x, y)
        with sequential execution previous,
            terminate in 0:0:30.0;
    spawn speak("Xavier here with your mail")
        with sequential execution centerOnDoor,
            terminate at monitorPickup completed;
    spawn monitorPickup()
        with sequential execution centerOnDoor;
}

Goal centerOnDoor (double x, double y)
    delay expansion
{
    int whichSide;
    spawn lookForDoor(&whichSide) with wait;
    if (whichSide != 0) {
        if (whichSide < 0)
            spawn move(-10); // move left
        else
            spawn move(10); // move right
        spawn centerOnDoor(x, y)
            with disable execution until
            previous execution completed;
    }
}
```

Carmen (Carnegie Mellon Navigation Toolkit) [Montemerlo, Roy, & Thrun 2003] was designed as modular software architecture, with each major capability built as a separate module. Carmen provides basic navigation primitives such as base and sensor control, obstacle avoidance, localization, path planning, mapping, and logging.

The Cambridge University Robot Language (CURL) is a Windows-based robot-independent programming environment [Harwin et al. 1997]. Complex tasks may be realized by creating CURL procedures using commands which employ natural language syntax. It provides an intuitive method of performing pick-and-place operations through the definition of objects and goals within a world model. Direct robot control functions are also provided to complement the task-level facilities.

There is also an XML-based language for data representation and interchange like RoboML [Makatchev, Tso 2000]. It is designed to serve as a common language for robot programming, agent communication, and knowledge representation. The ROCI framework mentioned in the control architectures chapter also uses an XML language as a configuration channel [Chaimowicz

et al. 2003]. In the framework for composing tasks self-contained, reusable modules are organized into tasks. A module contains a process which takes data as input, processes the data, and presents its resulting data as output. The modules are connected with pins. Data from modules or tasks can be subscribed to by other modules. PLEXIL plans can also be written in XML using the PLEXIL XML schema definition [Verma et al. 2006].

One way to simplify programming is through the utilization of graphical interfaces. Visual Programming Language (VPL) is a common name for programming languages that let users specify programs in a two-dimensional (or more) way. The usage of visual programming languages was evaluated in [Green, Petre 1992]. It was found to be very useful for robotics. Some visual programming environments are *Simulink* [MathWorks] from MathWorks and *Labview* [National Instruments] from National Instruments. One example of using VPL is *Microsoft Robotics Studio*, released in 2006 [Gates 2006] and [Microsoft]. In robotics, too, there are examples such as *Roboglyph* [Harwin et al. 1997], *the Robot Programming Simplification Project* [Arai, Itoko & Yago 1997], and *Onika* [Gertz, Stewart & Khosla 1993].

2.3.4 Software architecture comparisons

There are several comparative surveys to study the differences between the robot middleware or control architectures that have been developed. A common name could be Robotic Development Environments (RDE).

There are some overall approaches to architecting robotic systems, such as [Kortenkamp, Simmons 2008] and [Kramer, Scheutz 2007]. Several publications are focused on studying middleware and their evaluation, for example [Shakhimardanov, Prassler 2007], [Namoshe et al. 2008], [Long et al. 2007], and [Makarenko, Brooks & Kaupp 2007]. There are also surveys of execution systems, such as [Verma et al. 2005], and a survey of robot programming languages from Pembeci [Pembeci, Hager 2001]. Most of the surveys are literature reviews but there are also experimental tests, for example [Orebäck, Christensen 2003]. The deliverable document D3.1 “State of the art report Robot Middleware” from the RoSta project is also a good survey of state-of-the-art middleware [Rosta].

The fact is that almost every research group has developed its own solution for RDE. The comparative evaluation of systems is challenging. There is variation in several categories like robot hardware abstraction, platform support, extendibility, scalability etc. Also software tools, application development and methods have influence.

2.3.5 Standardization of robot architectures

There are plenty of robot software architectures which advertise themselves as the best platform for all the robots, as seen in the previous sections. In several projects researchers are developing similar kinds of systems, so there is a lot of overlapping. Because software development is a necessary process in autonomous mobile robotics, it is becoming more and more important to assist developers in their scientific and engineering work.

One significant project for standardization was *RoSta* (Robot Standards and Reference Architectures), a project funded under the European Union's Sixth Framework Program (FP6). The project was defined as the main international contact point for robot standards and reference architectures in service robotics [Rosta]. The technological objectives were to coordinate a set of actions initiating and preparing a set of standard defining activities on the following topics of advanced robotics [Rosta]:

1. the creation of a glossary/ontology for mobile manipulation and service robots
2. the specification of a reference architecture for mobile manipulation and service robots
3. the specification of a middleware for mobile manipulation and service robots
4. the formulation of benchmarks (of components, methods, middleware, and architectures) for mobile manipulation and service robots

For the standardization there is also the *JAUS* (Joint Architecture for Unmanned Systems) project, which is the standardization process to get an open architecture for the domain of unmanned systems [JAUS group]. Nowadays the JAUS standard is a messaging architecture facilitating communication with, and the control of, unmanned systems. JAUS has emerged in the military robotics community and is founded by DARPA (Defense Advanced Research Projects Agency).

2.4 Human-Friendly Communication

The field of study called Human-Robot Interaction (HRI) is dedicated to understanding, designing, and evaluating robotic systems for use by or with humans [Goodrich, Schultz 2007]. Interaction, by definition, requires communication between robots and humans. The aim of HRI is to develop principles and algorithms to allow more natural and effective communication and interaction between humans and robots. The field of study covers topics that vary from how humans will work with remote, teleoperated unmanned vehicles to peer-to-peer collaboration with anthropomorphic robots. In many studies the motivation of how robots should interact with humans comes from studies of how humans collaborate and interact.

Ferketic et al. [Ferketic et al. 2006] list several challenges that must be addressed in HRI. Humans and robots must be able to:

1. communicate clearly about their goals, abilities, plans, and achievements;
2. collaborate to solve problems, especially when situations exceed their autonomous capabilities, and
3. interact via multiple modalities (dialog, gestures, etc.) both locally and remotely.

There are several interaction methods to communicate with robots. Goodrich has listed the media manifested in HRI as follows [Goodrich, Schultz 2007]:

- visual displays, typically presented as graphical user interfaces or augmented reality interfaces
- gestures, including hand and facial movements and movement-based signaling of intent
- speech and natural language, which include both auditory speech and text-based responses, and which frequently emphasize dialog and mixed-initiative interaction
- non-speech audio, frequently used in alerting
- physical interaction and haptics, frequently used remotely in augmented reality or in teleoperation to invoke a sense of presence, especially in telemanipulation tasks, and also frequently used proximately to promote emotional, social, and assistive exchanges.

Natural language (speech), gestures, and touch are the most familiar and effective ways to communicate between humans. Why not provide equally natural interfaces to communicate with

robots? Using easy and friendly multi-modal interfaces between humans and robots might allow the users to concentrate more on the task at hand, rather than on the tools of the interface.

Human-robot interaction is not only the transfer of digital data between a robot and a device carried by a user. The form and structure of a robot are important because they help establish social expectations. The appearance of the robot has a great influence on the interconnection. Physical appearance biases interaction. A robot that resembles a dog will be treated differently than one which is human-like. Moreover, the relative familiarity (or strangeness) of a robot's morphology can have profound effects on its accessibility, desirability, and expressiveness [Fong, Nourbakhsh, & Dautenhahn 2003]. Additionally, the interaction between the operator and the service robot is often indirect. In indirect interaction, the operator controls the robot, which communicates information about its environment and its tasks back to the operator [Thrun 2004].

In the following sections selected ways to interact with robots are presented from the perspective of human-friendly communication. Visual displays, non -speech audio, and other non-human-like interfaces are excluded.

2.4.1 Speech and natural language

Human speech provides a natural and intuitive interface both for communicating with and teaching robots. In general, speech contains three kinds of information: who the speaker is, what the speaker said, and how the speaker said it. The content of speech is the type most used in robotics and also in computer science, but using speech as a source to recognize the speaker and his/her feelings is less used. Speaking robots are quite popular because of the good availability of speech synthesizers.

Speech recognition systems have become better in the past ten years. There are still no commercial recognition applications which understand natural language very well, regardless of who the speaker is. If the speaker is always the same and there is time for teaching the system then the results would be good enough. Again, if the application area is limited (with a corresponding limited vocabulary), such as medical dictation applications, the results are good. Using natural language is a different case because it is not enough to understand words but also the content and there are many alternatives for similar topics.

For applications or robotics where the robot and human establish and maintain a long-term relationship, such as robotic nursemaids for the elderly or robotic pets for children,

communication of affect is essential [Breazeal 2003b]. Bringing emotions to speech increases this affect. Emotional speech has been used in a few robot systems. Breazeal describes the design of the Kismets vocalization system. Expressive utterances (used to convey the affective state of the robot without grammatical structure) are generated by assembling strings of phonemes with pitch accents [Breazeal 2003a]. The primary parameters that govern the emotional content of speech are loudness, pitch (level, variation, and range), and prosody [Fong, Nourbakhsh, & Dautenhahn 2003].

2.4.2 Gestures and facial expressions

Non-verbal communication is often conveyed through gestures, body movement, and facial expressions. People use gestures to clarify their speech and to compactly convey geometric information (location, direction, etc.). A speaker will use hand movement very often (speed and range of motion) to indicate importance and will point to clarify spoken directions (e.g., “My home is over there.”). Waldherr et al. present a vision-based interface that has been designed to instruct a mobile robot through both pose and motion gestures [Waldherr, Romero, & Thrun 2000].

An approach commonly used for identifying people is face detection and recognition. There are several commercial digital cameras nowadays with a face detection application. Face detection in cameras is used to focus images on faces. An improvement on face detection is face recognition. In face recognition human faces are not only detected but also identified and matched to a person.

Facial expressions are considered to express emotions (such as a smile). Facial expressions have also been thought to function as social signals of intent. Lisetti et al. [Lisetti, Schiano 2000] have published a very good review of facial expression recognition. There are some basic approaches to facial expression recognition [Lisetti, Schiano 2000]. Image motion techniques identify facial muscle actions in image sequences. Anatomical models track facial features, such as the distance between the eyes and nose. Principal component analysis (PCA) reduces image-based representations of faces into principal components such as eigenfaces or holons [Lisetti, Schiano 2000].

Gaze tracking is the process of measuring either the point of somebody’s gaze ("where we are looking") or the motion of an eye relative to the head. Gaze is a good indicator of what a person is looking at and paying attention to. A person’s gaze direction is determined by two factors: head orientation and eye orientation.

2.4.3 Haptic interface

One possible interface between human and robots could be using touch. Haptic interfaces are usually understood as being similar to force-feedback in computer game controllers. Haptics is familiar to everyone, even if the word is not. It is the feel associated with operating a mechanical device, such as dialing a touch-tone phone, finding first gear in a manual transmission car, or playing a musical instrument like a guitar or a piano, which all rely heavily on the tactile and kinesthetic cues we receive [Immersion]. "Haptics" is a Greek word meaning "the science of touch." Haptic feedback is a natural side-effect of manual controls and helps an operator know more about the environment in which a machine is operating, as well as its performance [Georgia Institute of Technology].

In service robots haptics is useful not only in teleoperation applications but also in direct control of the robot itself. For example, you can grasp your child's hand and lead him/her home. A similar application on the robotic side is presented in the WorkPartner project from Helsinki University of Technology [Suomela, Halme 2004].

Figure 11. Leading the WorkPartner robot.

The haptic interface that is presented allows the operator to control the robot's movements by touching its hand (see Figure 11). It is based on multiaxis force measurements of the hand. The measurements are performed with multiple strain gauge bridges, which measure the pulling and pushing forces and torques to the left and right. These signals are amplified to the robot's movements. A similar power assisting system has been presented and tested by Saarinen et al. [Saarinen, Suomela, & Halme 2002]. In this case the application is a rowing assistant, which allows a rower to row with normal movements but with less power.

2.4.4 Multi-modal communication

People do not use only one communication method at a time. For example, speech and gestures are often used to support each other, such as saying “Go over there” and pointing with the hand. Spoken natural language and natural gestures are more user-friendly when communicating with robots. The human is not required to learn additional interaction methods, but can rely on natural ways of communication. Multimodal interaction provides the user with multiple modes of interfacing with robots. A typical interface in the past has been the use of a computer with a keyboard and mouse. A possible addition could be the use of speech or gestures. Additionally, any combination of the interfaces presented above could be a potential solution.

Perzanowski et al. [Perzanowski et al. 2001] present a multimodal human-robot interface. They implemented multimodal interface on a team of robots. The robots understand speech, hand gestures, and input from a handheld Palm Pilot or other Personal Digital Assistant (PDA).

2.5 Summary

Designing a development and integration software environment for robotic systems is not an easy task. There are still no all-purpose standard robotic platforms; any reusable framework must be sufficiently flexible to address the variations in robots. Robotic systems present challenges because of the differences in their physical capabilities, sensor configuration, and hardware control architectures. Many existing programming environments that have been presented all propose different approaches to the development and integration of robotics systems. Most of them are incompatible with each other for different reasons [Orebäck, Christensen 2003], such as the use of specific communication protocols and/or mechanisms, different operating systems, robotics platforms, architectural concepts, programming languages, intended purpose, proprietary source codes, etc. This leads to code replication of common functionalities across different programming environments, and to specific functionalities often being restricted to one programming environment [Côté et al. 2006].

In thesis the own platform was also developed and there are many reasons, as mentioned previously. Research project started in 1998 and platforms developed for our purposes did not exist. The platform (the WorkPartner robot) is unique in the world and is also used for teaching, project work etc., which have their own effects. At the beginning of the author’s work a huge software platform had already been developed and the work was a further development of that.

The one main difference compared to the system reviewed is the lack of a link to the human user. There has been a lot of research work into making robots move but there is not so much studying how to configure tasks in as human-friendly a way as possible in co-operation between humans and robots. Users and robots should have a common understanding in their working environment and it should be possible to use external markers as a part of the task control strategy.

3 Configuring Tasks: Master and Apprentice

3.1 Introduction

To be of any use, a service robot must be able to perform the tasks given by the user. Sometimes in a very simple scenario, all tasks can be given a priori, but in more realistic cases - in a dynamic world - that is not the case. That is the main reason why configuring or, some would say, the teaching of tasks is an essential part of the human-robot interaction in current service robots. Approach in this thesis to this challenging problem is tight co-operation between a human and a robot. The methods presented have a strong influence from craftsmen's trades, where the master-apprentice type of training has already been the main method of training for centuries. It is a continuous process containing several steps/levels/stages/phases. A carpenter is a good example of a traditional craftsman's profession and a typical task for him is the building of a chair. The work includes several phases which are presented in Figure 12

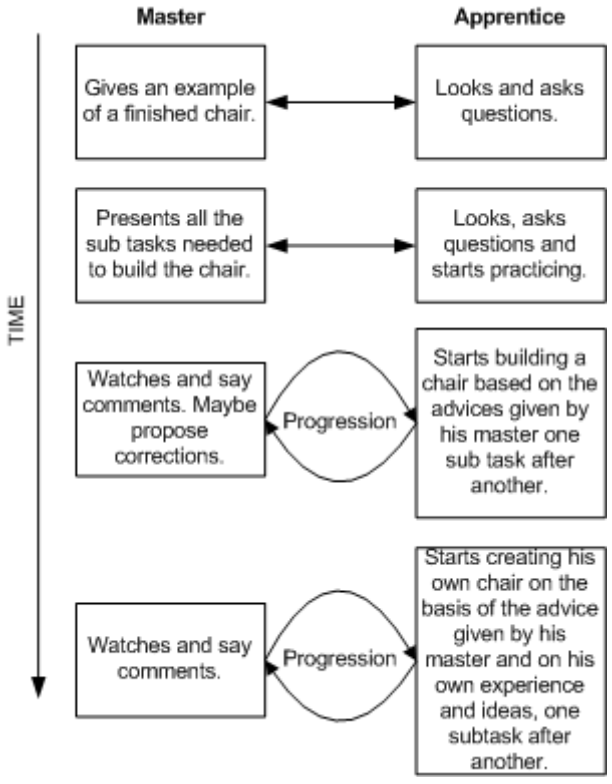

Figure 12. Working phases when a master trains an apprentice.

In reality, of course, the case is not as simple as shown in the table. The master cannot give all the instructions needed in the first two phases, because of the lack of information about how the apprentice has absorbed the given instructions. Proper feedback can be obtained only after

the first trials. After that the master gives more detailed information and adds the missing phases to the building process. The apprentice continues his work and finally he can accomplish the task and after that he can start to design his own prototypes.

By simplifying a previously described process and by taking into account the limitations of modern robots, the proposed method could be considered as the writing of an instruction guide. In the first phase the master writes down the steps involved in building a chair. Next, the apprentice reads the instructions and starts building the chair step by step. The master follows the process and interrupts the work or proposes corrections. At the same time the master makes corrections to the original instructions. This proposed process is named the *Simultaneous Instruction Writing and Execution method (SIWE)* in this work and author argues that it is directly transferrable to service robotics, as shown in the following sections.

3.2 Human-Robot Task Configuration

When applying the SIWE approach to robotics, the human user can be imagined as acting as a master and the robot as the apprentice. First, the user decides on the work phases needed on the basis of his experience and "writes" them down. This is called *the task description*. The human user who makes this preliminary plan for the final work task is called *the task configurator.* The process is called *task configuration* and the making of this first plan can be thought as the sketching of a plan. Obviously, this sketch of a plan is an opinion of the task configurator on how the task should be performed. The end user might have a different view and the working environment might also change. Therefore, the end user might want to change the original plan. The situation is quite similar in process automation industry. There engineers develop the control system and user interfaces of process. However, the configuration might be different than end user is experienced. End user might want to make changes to the system but it is often challenging. Sometimes the changes might be permanent settings to the system or sometimes only temporal. The task configuration process should be so explicit that the end user too is capable of succeeding. A visual activity diagram [Douglass 2000] is proposed as a method to represent the work tasks. So, instead of writing the plan down, the task configurator draws it.

The preliminary plan is sent to the robot and then executed. During the execution the task configurator follows the work and makes changes to the plan or interrupts the execution if needed. The plan will thus get better and better through this iterative process. This plan

enhancement loop is presented in Figure 13. In the figure blocks T1c to T4c are examples of subtasks of plan.

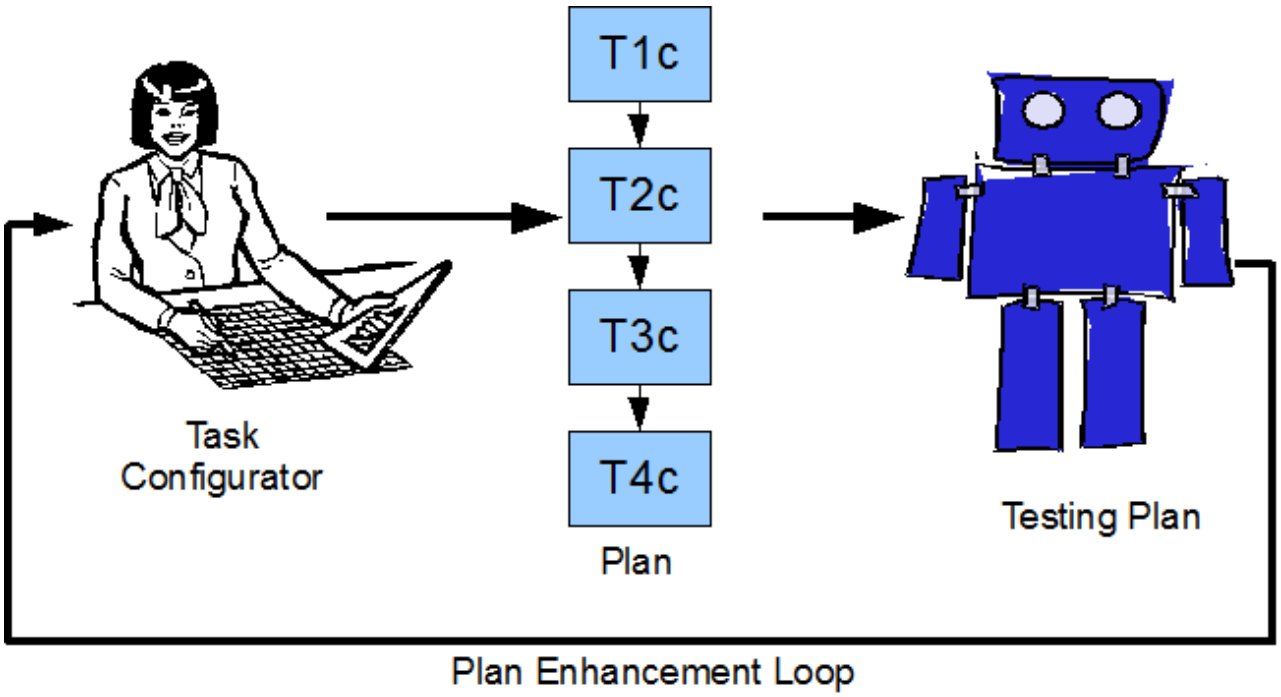

Figure 13. An overall description of the task configuration.

The proposed method is based on the idea of splitting the task into modules, just as an experienced craftsman splits his task into smaller subtasks. These modules, called micro tasks (MT), perform simple motion, sensing, or perception subtasks. The method enables movement and perception functions to be combined in the same work task. For example, independent micro tasks called *MT_locate_object* and *MT_gotoxy* could be paired into a perception and movement function. As an example, possible tasks might be the moving of the robot near to a red box. The task configurator defines a plan and enhances it by changing the parameters of the micro tasks. Enhancing *MT_locate_object* will improve the recognition of the red box and locates the box more accurately. The position obtained for the red box is now the target point for *MT_gotoxy*.

Teaching a task means providing or configuring a plan of how to use available resources, i.e., the micro tasks. The operator makes the plan interactively with the aid of the user interface by utilizing suitable input functionalities, in most cases in the first phase graphics and “imitation” through teleoperation. A micro task includes parameters, such as motion speeds, distances, and forces, which are optimized in the second “training” phase. This training uses forced or reinforcement learning under the operator’s supervision, i.e., the operator grades the

performance. The third phase would be based on unsupervised learning, where the parameters (or even the structure of the original plan) can be evolved on the basis of some suitable fitness criterion, for example the execution time. Development of the presented learning capabilities (phases two and three) are a real challenge for the future. The first steps are already taken, for example, in [Kaski 2004]. Kaski applies genetic algorithms to simple subtasks (finding a box from the surroundings, moving to the box, lifting the box and taking the box to a specific place). This can be imagined as optimization of single micro task. The next phases would be optimization of parameters of whole plan and finally optimization of the original plan.

3.3 Classification of Users

The users of the proposed task configuration method can be categorized into three groups on the basis of their skillfulness and their job descriptions. These groups are named *end user*, *task configurator*, and *engineer*. Classification is important because it is not possible to develop interaction methods suitable for every user. One example of the classification of users is from process automation factories. The control room has a different kind of user interface than for people working in the offices or for engineers who are configuring the system. Sometimes a worker in the control room might become enough skillful to be able to work as an engineer. The groups are presented in more detail in Table 1. In addition, the boundary between the end user and the task configurator is often vague and flexible (such as in the factory scenario presented above), because the end user can become rather skillful when they have a long work history with the robot. PC support is one example where a worker could become more experienced during work history.

Table 1. Classification of the users on the basis of their skillfulness and their job descriptions.

User group	Description
End users	End users fulfill task plans to accomplish real work tasks. End users use a multimodal user interface, including a graphical user interface presenting a map of the work environment and other work-related information. Tasks are performed using speech, gestures, and teleoperation devices, for example. End users do not need plenty of knowledge about robots or about the internal functions of the tasks.
Task Configurators	Task configurator creates new work tasks using the Configure tool (see Chapter 4.5.3) and in co-operation with the robot. They sketch a plan of work tasks by picking suitable sub-tasks (i.e., micro tasks) from the list of available ones. After dragging and dropping micro task blocks to the design frame, the user defines the order of execution of micro tasks by drawing connections between the blocks. The micro tasks have some parameters that might need modifications before the plan is finished.
Engineers	Engineers design and program new abilities (functionalities) to the robot. This means the programming of new micro tasks. For example, the robot could get the ability to recognize a new plant in the yard. This might be important during a 'clean yard' task in order to prevent damage to some particular plants.

3.4 Task Configuration

The overall description of the task configuration process is presented in Figure 14. The same figure includes a description of the elements and the job statuses, as well as the three functional elements of the task teaching stages. These elements are: the task configuration process, modular and interpretive task execution and plan control. The three phases of task configuration from the user's point of view are presented in Figure 14 as bold dashed arrows numbered from one to three.

The task configurator sketches a plan, as mentioned in section 3.2. The Task Configure tool is used not only for configuring tasks off-line but also on-line when making modifications to the task description during the execution of the task. This is made possible by using an interpretive execution of tasks (see Chapter 4 for details). The robot stores the finalized plan and offers it for real use. The end users will see these plans configured for the robot.

The end users work together with the robot by interacting through the multimodal user interface. More details of the user interface will be presented in Chapter 5.

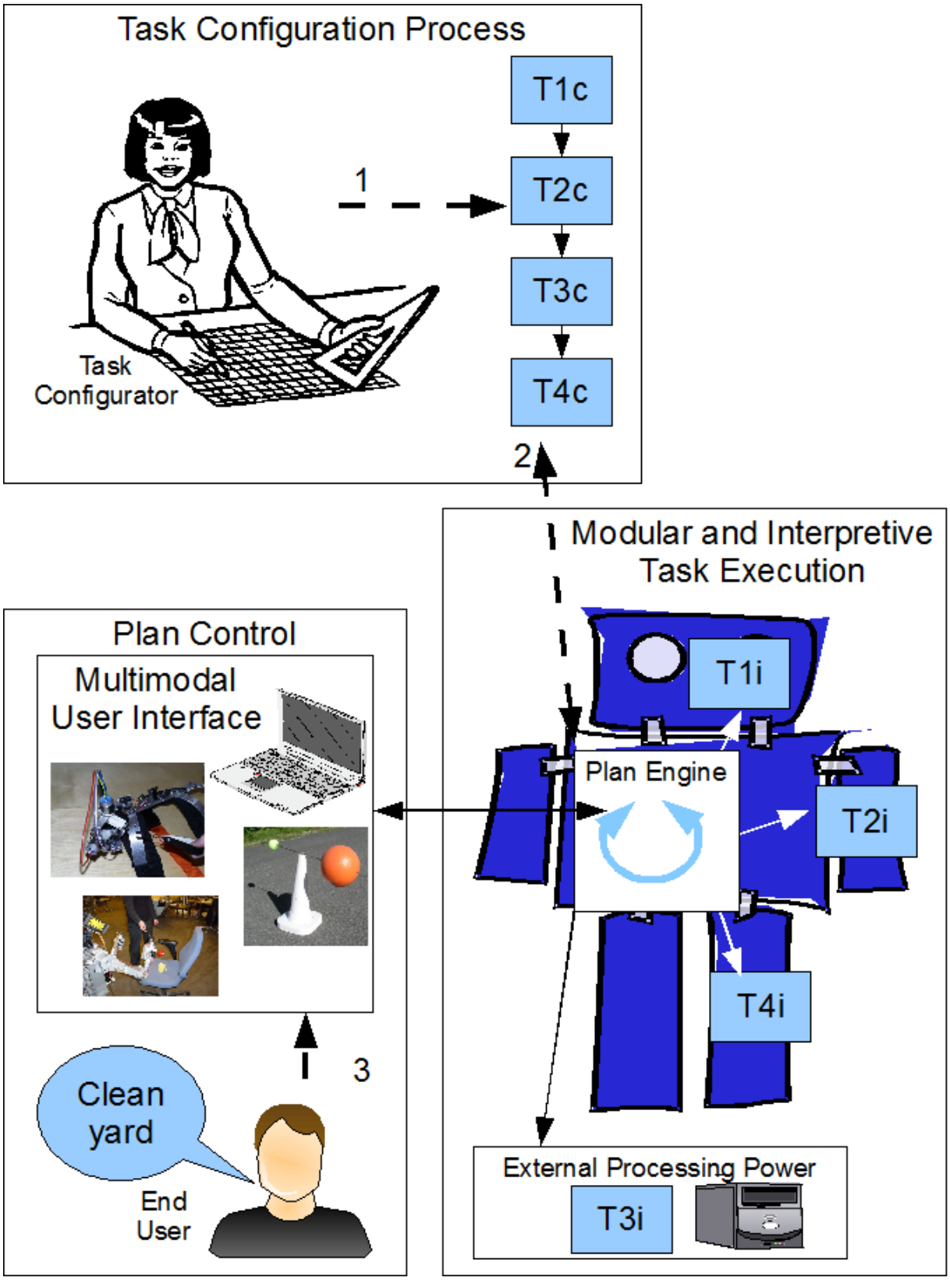

Figure 14. An overall illustration of the task configuration and execution process.

The proposed Simultaneous Instruction Writing and Execution (SIWE) method for configuring new tasks for service robots has its roots in the ancient master-apprentice approach. Method is also a good base for human-robot interaction and task configuration research. Instead of programming complex work tasks with all the details, it is better to just sketch a plan of the desired task and then improve it through an enhancement process. It is really quite similar to the process in which a master teaches his apprentice how to master new skills.

The platform proposed in this thesis also provides for the sharing of configured plans over a network. In the future the owners or users of robots could have access to an internet-based service offering new tasks (for example pick up mail from the letter box). Instead of configuring the plan, the user looks whether the service already offers a suitable task. Maybe not entirely the same task is configured (for example, pick up newspaper from the ground), but with a few modifications it will be. Some of these loadable tasks might be free and some subject to a charge. This is quite similar to the services offering applications and source codes for personal computers. Community-based software projects (for example the Firefox web browser) have shown their strength today. Some skillful end users can even become task configurators who can create new tasks and upload these to the service. In this way the large society of robot users of the future will develop more and more sophisticated work tasks. Therefore, these work task configuration communities may be one conductive factor for service robots. This in turn will speed up the development of the field in general and so the vision of using robots extensively in our everyday life as an important part of society might turn into reality faster than anybody can currently believe.

4 Task Control Platform

4.1 Introduction

The method for configuring new work tasks in the previous chapter presents the theoretical background for the real software implementation proposed in this chapter. The software platform presents a novel way to control service robots for accomplishing real work tasks. The platform is developed for the easy implementation of complex work tasks to be executed by various service robots. It defines the final plan of the work task and the building blocks used to create the work task. The platform offers a specification of the services and communication channels between the modules implemented and the user. Reliable communication with the user is one of the key elements in the process of the development of the platform.

The configuration of the plan is stored by using a so-called task composition language. A task composition language is a representation of a plan providing a hierarchical structure of the actions of the desired work task. The language provides the elements for representing the actions needed, their internal parameters, and the order of execution of the actions, including handling exceptions.

To create these final plans, a task execution system is needed. This engine should be robust and flexible in order to cope with varying plans.

4.2 Objectives and Definitions for the Platform

There are several needs for a robot task management system. It should be decentralized, scalable, and portable. The system should support high-level interaction between the user and the robot. The configuring or teaching of the tasks should be as easy as possible. The hardware of the system should not be a barrier to using the system. The platform should be applicable to various hardware systems and situations. If the existing tasks are not sufficient, more tasks should be easily programmable. The matter of portability includes several aspects. The tasks should be functional if the robot is transferred to a different environment or if the code of the tasks is transferred to a different robot. The subparts of the task should also be functional when transferred to another work task.

The platform could support future task trading between users. If there is a functional task for some particular type of robot, it should be possible to distribute the code over the Internet. In

this way all the users of similar robots could utilize the tasks implemented and tested by another user. In the following sections a more detailed description of the terminology is presented.

4.2.1 Plan

A plan implies the required human and robot functions and the goals they are attempting to achieve. From the software point of view a plan is a set of micro tasks which contain all the components and their interconnections to get a particular task or tasks accomplished (see Figure 15). The lowest-level micro tasks contain the basic operations of the robot. Micro tasks can be grouped together to form higher-level tasks. The actual skilled tasks can be built up from micro tasks and higher-level tasks. The skilled tasks can again be used as a part of a yet higher-level skilled task. The number of levels is not limited. In this way the robot can gain a very high level of expertise.

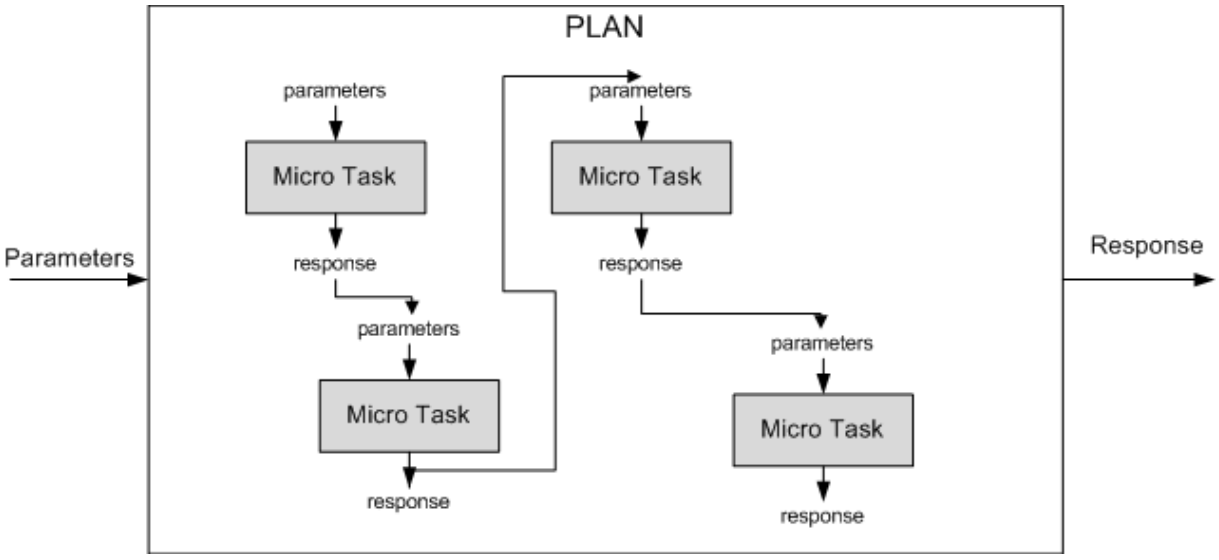

Figure 15. Plan is a set of micro tasks. Information is passed between micro tasks with parameters and responses.

4.2.2 Micro task (µTask)

A micro task is the lowest level building block for a work task. A micro task will always be executed as a whole, and the response will be used as additional information for the future tasks. The platform allows the user to group micro tasks into more complex tasks, and these can be grouped to yet higher-level tasks. In the concept, micro tasks can be divided into three groups: movement, perception, and calculation related micro tasks. Physically micro tasks are similar implementations. Movement related micro tasks can be used for the locomotion of the robot, and for the movement of its other degrees of freedom, for example a manipulator torso. Perception-

related micro tasks are used to perceive the environment by using the sensors of the robot. This might also need some movements. For example, if a camera is attached to a pan-tilt unit, it can be turned to focus on the subject of interest. Calculation-related micro tasks neither gather data from the sensors, nor move the robot, but simply process the input data. These micro tasks can be used, for example, for coordinate transforms, or for database operations. In general, the data are gathered with perception micro tasks, then they are processed with calculation micro tasks, and finally the robot is moved according to the results. If feedback is needed, more information can be gathered with some perception-type micro tasks. Micro tasks can be gathered into larger entities that can again be used as building blocks. These higher-level tasks can be considered as the "skills" of the robot. These skills can be reused in other similar tasks and environments, as long as they contain an implementation of the micro tasks used to build the higher-level tasks.

The internal structure of a micro task is very straightforward. A micro task contains a state machine with predefined "start", "stop", and "pause" states. This is somewhat similar to the software components by [Ando et al. 2005] but lighter and more flexible in means of programming. Usually a micro task also defines various other states for the execution of that particular micro task. A typical case is taking a measurement. In one state the micro task sends a request to the perception server, and then it moves to the next state, where it waits for the response. A "start" state is used for the initialization of the variables and to ensure that all the conditions needed to run the micro task are met. In this phase the micro task determines which state is the first functional state. If the micro task is just a simple task that does not require interaction with other subparts of the system, all the logic can be included in the "start" state. A "stop" state is executed after the micro task notifies the higher level that the required states have been executed. This state can also be executed if the user uses the emergency stop option. The "stop" state finalizes the execution of the task. For example, in a movement-related micro task the end state ensures that the robot has actually stopped after the movement. A "pause" state is used either when the user suspends the execution, or when the robot encounters a situation that it cannot resolve. In these situations the user guides the actions of the robot. For example, if the robot is working in challenging terrain it might be very difficult for the robot to plan its moving trajectories, for example because of the slipperiness of the surface. The user may suspend the execution, lead the robot around the difficult area, and let the robot handle the rest of the movement.

4.2.3 Parameters and responses

Work tasks usually need parameter information to be passed between the plan's building blocks. The goal is to keep the number of parameters low, so that the order of the building blocks and the connections between them would best describe the structure of the work task. The parameters and responses use named variables that describe some global value, such as the orientation of the robot or the position of an object being manipulated. These kinds of variables are usually responses from the perception or calculation tasks, and are used as input parameters for the manipulation or other calculation tasks. Sometimes, a perception task needs parameters, for example to define the appearance of a target being sought. Parameter types are described in Section 4.4.4.3.

4.2.4 Visual realization of the plans

The structure of a plan is shown as an activity diagram (see Figure 17) [Douglass 2000]. It shows the sequence of the tasks to be executed, and is chosen for the representation because of its ease of interpretation. There are four types of blocks used in the diagram, micro task blocks, if-blocks, fork blocks, and join blocks. A large number of different sequences can be built with these types of blocks. All the normal micro tasks are shown as normal blocks containing the name of the block and maybe the most important parameter information of the block. The if blocks are used as branching points in situations needing decision making. For example, making loops with the graphical if-blocks is very intuitive. The fork blocks are used to separate the task flow into two or more branches that all continue to run actively. For example, with this kind of structure, one branch of the program can take care of the collision detection while another branch is moving the robot. The join blocks are used to merge two or more branches back into one. The execution after this block continues when the last branch has reached the merging block.

4.2.5 Perception of the environment

Perception is an important part of any type of task execution. When executing skilled tasks, humans usually get feedback from their senses, including, for example, vision, hearing, and their tactile senses. This information is used for the coordination of muscles and for evaluating the success of the execution. In the learning phase the senses are integrated as a part of the execution of the whole task. When learning a completely new task, a human sees many details which are

used as aids for the execution of the task. When the learning proceeds the human learns which details need to be observed. This reduces the amount of information to be processed.

The perception of robots differs from that of humans. The sensors of the robots are usually different than those used by humans. Some of the sensory inputs are more accurate, there are some sensors that do not have human equivalents, and some senses are completely missing. When a robot executes a task(s), it usually uses fundamentally the same senses as humans, such as visual, aural, or tactile sensation, if they are available. The input from these sensors might still be different. For example, our three-dimensional vision is produced by stereo vision. A robot can have binocular vision, but it might equally well use a laser scanner, a laser distance meter with a single camera, or some other kinds of sensors.

To be effective, the perception should be somehow independent of the actual actuation. For example, placing a camera on a turning head would allow the robot to change the target of its vision, but keep the manipulator still. This is useful, for example when the robot is holding an object and trying to find a suitable position to carry it to.

The perception is accomplished with tasks that perceive the environment. A locating task returns the location of the target on the map if the target was found. The task also returns a variable indicating whether the target was found or not. This information can be used later for executing an alternative branch that executes additional actions. For example, the robot can turn its head to look around, if the target object was not found from the image.

4.2.6 Interaction with the user

The user interface helps the user both in the configuring phase of the plan and in the execution phase. In the configuring phase possible mistakes are prevented by giving additional information to the user. In the execution phase the current state of the task is depicted. It is not always possible to complete an entire task with predefined sequences. Sometimes the robot needs help from the operator. If the robot encounters a situation that it cannot solve on its own, it can interrupt the execution and wait for the user to guide it to the next phase. When the user has helped the robot, the work task can continue from the next phase or from another phase determined by the user.

The parts of the task should be independent of each other in the work tasks configuration. In this way it is possible to bypass some parts without causing problems for the later phases. A problem might arise if the user wanted to bypass a one task, and later the results of that one were

needed. In this case the robot would ask the user for guidance again until the necessary preparatory phases had been executed, or the tasks could be bypassed.

The interaction between the robot and the user can be achieved in various ways. Usually, to operate a robot the user needs some kind of portable computer or something similar. He/she will use this device to see, for example, the image from the robot's camera, the map, or the state of execution of the task at hand. The user can, for example, point to objects to be manipulated from the image or from the map.

The interaction can also take place in the real world. The user can use gestures or show the objects to be manipulated with a pointing device, such as a distinctively colored stick. In addition, interesting objects can be marked with signs. Signs can also be used to determine the area where the robot is supposed to operate. [Heikkilä et al. 2006]

4.3 Functional Modules in the Platform

A very important feature of the implementation of the platform is decentralization. All the software modules in the systems are independent and communicate with each other via a common interface. The physical location of the modules can be anywhere in the network. The common communication interface is TCP/IP-based, providing communication over the internet, as well as in the local system. Decentralization offers, for example, external calculation power for small controllers and fluent communication in multi-robot systems. Figure 16 presents the functional modules of the platform that was developed that concern task control and there is a more detailed presentation in table 2. The development of the platform continued after this thesis work. The software architecture called GIMNet presented in [Saarinen et al. 2007] is a continuation of the software platform presented in this thesis.

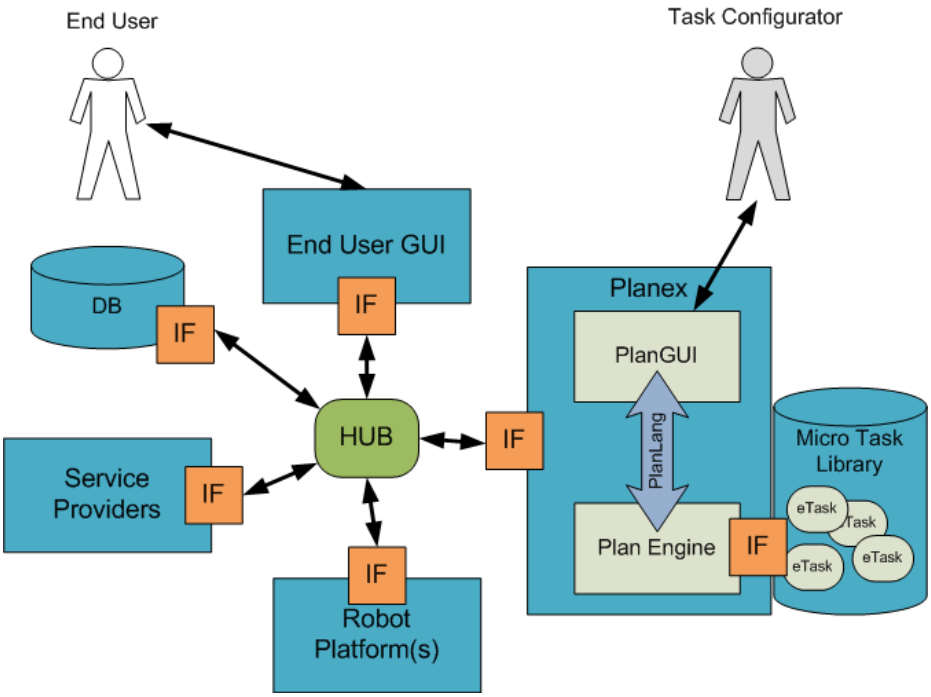

Figure 16. Functional Modules of Task Control Level in the Platform.

Table 2. The modules of the Task Control Platform.

Planex	Planex is a combination of a plan configuration tool and a task execution engine. The plan engine is a state machine-based task execution system controlling the tasks managed by the task library. This module takes care of interpreting the plan description and controlling the execution flow. Plans are described with the XML version of the PlanLang task composition language. Planex has its own graphical user interface (PlanGUI) for the task configurator. The task configurator creates plans of real-world work tasks using a visual programming language which is a representation of the PlanLang.
End User GUI	Graphical map-based user interface. End user GUI is the main module for end users commanding the robot and the tasks. The main interface is a map-based window, via which the user can see the map of the environment and the location of the robot. GUI depicts not only the working environment but also the common ground between the robot and the user. For example, the objects recognized by both the robot and the human are displayed on the screen.
Service providers	The perception server is a typical service provider and it interprets the sensor data. It can also actively take measurements of its environment by utilizing the degrees of freedom of the robot, which can move the sensors. For example, the location of an object on the ground can be measured by pointing a pan-tilt unit-controlled laser distance meter at the object. Other service providers could be: • modules that handle speech recognition and synthesize it. • Image Server communicates with a camera system and makes video and image data available for the other modules.
Database	All the important data of the objects are stored in a database. The database is designed to represent the surrounding world for the robot. The same information should be in human-readable form for fluent information exchange (see [Saarinen et al. 2004]). Large amounts of

	data have to be retrieved at once, for example in map creation, and the information that is required needs to be searched for fast by the classification of database objects. The database is like a model of the world.
Hub	Center of communication. All modules communicate with the hub, which transmits messages to the right targets.
Robot platforms	Communication channel between the low-level robot system and the higher-level task control system.
IF	A common TCP/IP-based interface module for inter process communication.

4.4 Task Composition Language

The task composition language can be considered as one of the basic requirements to make a robot execute plans. Instead of programming tasks in the common language (C++, for example) tasks should be configured easily. The task composition language is a format for storing plans in a form understandable by both robots and humans. Verma [Verma et al. 2005] defines a task composition language as "a representation of commands and plans that provides a representation for reasoning about [the] required robot and environment state, as well as the effects of executed commands." A task composition language provides a representation for monitoring the restrictions of tasks and encoding appropriate responses if these restrictions are violated. It also takes into account the interdependence between actions, in terms of temporal precedence and other constraints, such as resource contention.

4.4.1 Requirements for task composition language

There are several common features that drive the design of a task composition language. For one, the language should support the requirements listed in the previous chapters. The execution system that executes the language must be efficient. The language should be *interpretable* by an execution system and must run fast enough to meet the task-specific timing requirements. The language should be *modular*, so that commands for different operations can be encoded independently. The language should be semantically clear and deterministic, given the same sequence of inputs. The language needs to be expressive enough to represent simple branches, loops, concurrent activities, and sequences. At the same time the syntax should be simple and uniform, making plan interpretation simple and efficient for both a human and the planner on the robot, while enabling validation and testing techniques to be applied. In addition, it

should support multiple automated planners or task executors. The task composition language should be *amenable to mixed initiative planning*. In other words, it should be possible to transform plans between automated planners and humans fluently. Users or some high-level planners should be capable of editing these plans if necessary, even during runtime. The language definition should not define only the plans and tasks but also the control commands for the tasks. In other words, the phases of the execution of the task should be included in the language.The language should interface easily and naturally with other tools, such as path planners, a diagnosis tool, third-party libraries, etc.

4.4.2 Activity diagram as a source form as a task composition language

Visual representation is one of the key requirements for the task composition language. The heart of the language is the XML syntax, but the human user mainly sees a visual diagram while configuring a plan. An activity diagram is a diagram type that has been found to provide a good representation for plans and it depicts the XML format well.

Activity diagrams depict systems that may be broken down into activities – roughly corresponding to states that mostly terminate upon the completion of the activity rather than as a result of an externally generated event. Activity diagrams may be thought of as a kind of flowchart in which the diagrammatic elements are member function calls. [Douglass 2000]

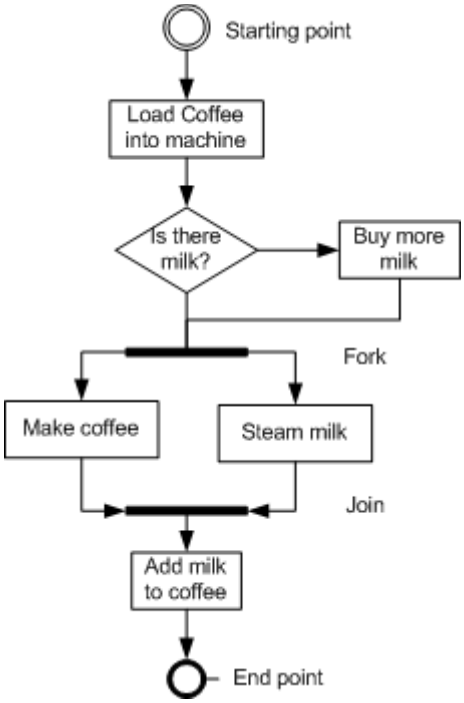

Figure 17. An example of an activity diagram.

Activity diagrams have several elements, including starting and ending activities, forks, joins, guards, and states. As shown in Figure 17, the activity diagram is suitable for the depiction of tasks that can be broken down into activities. Activities are terminated at completion rather than as a result of an externally generated event. Most real-world tasks could be imagined in the form of an activity diagram.

4.4.3 Selected description language

The graphical form of an activity diagram needs some description language as a storage format. XML (Extensible Markup Language) [W3C] has been chosen for several reasons. It is understandable both for a robot and for a human (up to a certain degree). There are good programming libraries for parsing and saving XML documents and it allows developers to set standards defining the information that should appear in a document. XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion. Even developers performing tasks on different types of applications with different interfaces and different data structures can share XML formats and tools for parsing those formats into data structures that their applications can use. The tree-like hierarchy of XML shares the hierarchy of the plans.

XML Format

XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree. The basic format of XML is presented as follows:

<Element1 Attribute1=”attributevalue1” Attribute2=”attributevalue2” Attribute3=”attributevalue3”>

<Element2 ..../>

</Element1>

Each XML document contains one or more elements, the boundaries of which are either delimited by start tags and end tags, or, for empty elements, by an empty-element tag. The previous example has two elements, named “Element1” and “Element2”. There is exactly one element, called the *root, or document element*, which will not appear in the content of any other element. “Element1” is the name of the root element in the above example. Because of the tree-

like hierarchy of XML, the content of the element might consist of other elements. These sub-elements are called *child elements*. In the example, the content of Element1 is Element2 and it is called a child element of Element1.

The elements might have *attributes*. Every attribute has a name and a value. For example, “Attribute1” is the name of the first attribute of “Element1” and “attributevalue1” is the value for that.

4.4.4 PlanLang – Task composition language

XML is used as the storage format in defining the interfaces of the micro tasks and the whole structure of the plans. The storage format of the plans is called *PlanLang*. PlanLang has several requirements to store plans in sufficient way:

- Should be able to store definitions of the micro tasks
- Should be able to store the structure of the plan
- Should be able to store all the information to and from the micro tasks
- Should be able to store the execution status of the plan

Based on above requirements the building blocks for PlanLang are defined. The fundamental building block in PlanLang is called a *MicroTask*. A micro task is a data structure with two primary components: a set of attributes that drive the execution of the task, and a content which specifies the data information to and from the task. PlanLang specifies the definition part of the micro task. The implementation part (the real executable part) is an external resource for PlanLang.

Another major building block is an element called a *Connection*. The connections define the order of execution of the task, but not only for the straight path but also for the possible branches. The basic idea is to define “What is the next task after this?”

A plan in PlanLang is a tree of children elements, which represents a hierarchical decomposition of tasks. Figure 18 exemplifies a simple plan as represented by a PlanLang task tree.

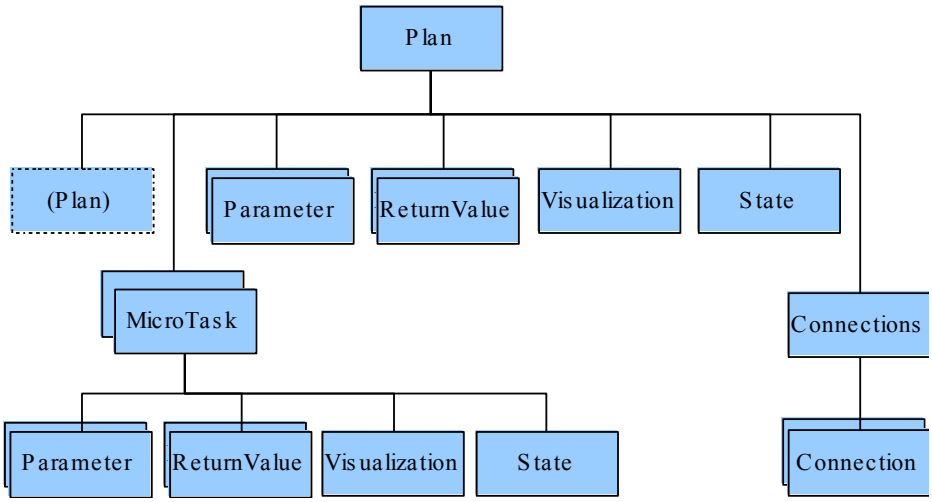

Figure 18. Hierarchy of PlanLang

The previous example is a representation of the following lines of PlanLang. Simplifying and highlighting the hierarchy, the next example presents only element names and all the attributes are shown with three dots (example of full example of one plan is presented in Appendix 2). This example is depicted on the PlanGUI screen, as shown in the next diagram, Figure 19.

```
<Plan …>
        <Parameter …/>
        <Parameter …/>
        <ReturnValue …/>
        <ReturnValue …/>
        <Visualization …/>
        <State …/>
        <MicroTask name=”Task1”…>
                <Parameter …/>
                <Parameter …/>
                <ReturnValue …/>
                <ReturnValue …/>
                <Visualization …/>
                <State …/>
        </MicroTask>
        <MicroTask name=”Task2”…>
                <Parameter …/>
                <Parameter …/>
                <ReturnValue …/>
                <ReturnValue …/>
                <Visualization …/>
                <State …/>
        </MicroTask>
        <Connections …>
                <Connection …/>
        </Connections >
</Plan>
```

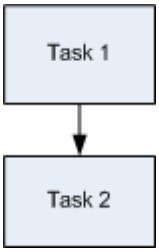

Figure 19. Example of depiction of plan in PlanGUI.

4.4.4.1 Element types in PlanLang

There are eight types of elements in PlanLang. Elements are selected to fill all the requirements presented in previous section. The root element is *Plan*. For example, a command such as “clean (yard)” might be called a plan. There are six children elements under the Plan. They are called *Parameter*, *ReturnValue*, *Visualization*, *State*, *MicroTask*, and *Connections* (as shown in the previous illustration). The Parameter, ReturnValue, Visualization, and State elements are children elements not only for a Plan but also for a MicroTask (as shown in Figure 18). These elements are used to store the input, output, and status information of the tasks. The plan has some input values (parameters) for defining the control command for the plan. For example, “yard” is a typical parameter. Not only the input but also the output (return values) are needed. For example, in the “clean yard” task it might have a “work done” kind of return value. As shown in Figure 18, there are also Visualization and State elements under Plan.

4.4.4.2 Attributes of micro tasks

Implementation format

PlanLang itself is only a descriptive language of plans, so there must be a real implementation of the tasks. This implementation format defines the interconnection between the plan engine and the task itself. The implementation varies greatly. The most primitive implementation format of the micro tasks is the use of hard-coded functions inside the plan engine. For example, the start, stop, and flow control tasks are some instances of these hard-coded tasks. The implementations of hard-coded tasks are both programmed with C++ and Ansi-C. Both versions have their own predefined format. The micro tasks might in the future be external services for the plan engine. SO, Python, and maci are examples of these external versions. In Table 3 the current implementation formats are presented.

Table 3. Implementation formats of the micro tasks defined in PlanLang.

Implementation Format	Definition
Unknown	Default value if the implementation format is not defined
CPlusPlus	Micro tasks are programmed inside the execution engine using the C++ language. This is the default implementation format used in the experiments in this thesis.
Python	Implementation is done using Python. The execution engine has an interface for interaction between these external micro tasks.
SO	Micro tasks are programmed as an external SO library file in Linux systems.
Maci	Micro Tasks are decentralized to a GIMNet platform. An implemented micro task might be executed anywhere in the network. For example, it could on another robot or on a more powerful computer. This format will be implemented in the future.
Cee	Micro tasks are programmed inside the execution engine using the Ansi-C language.

Category

Categories are used to classify the selected micro tasks. They give guidance only to the plan engine and are more like a type of micro task. Physically, all the micro tasks are similar. According to the category, the controller adjusts the way actions are executed. Category defines what kind of micro task is or what it does. It is also information for the user when configuring the plan. There are eight categories of micro tasks defined in PlanLang at the moment. In Table 4 the categories and the purposes they are used for are presented.

Table 4. Categories of the micro tasks defined in PlanLang.

Category	Definition
Unknown	Category is unknown.
Start	Starting point of the Plan. All plans need one Start task.
Stop	End of the Plan. The execution of the plan will stop this task.
FlowControl	All the plans defined as a flow control affect the control order of the Plan. Typical examples are if, for, and case sentences in common programming languages such as the C-language.
MicroTask	This is default value. Usually micro tasks have more detailed type (for example ILMR).
ILMR	The defined micro task is an expression of the Intermediate Language for Mobile Robots [Kauppi 2003].
Calc	A micro task calculating some mathematical function(s).
Plan	The root element plan could also act as a child element. Then it is an expression of another plan. It is like a short-cut.

Target platform

The implemented micro tasks are executed under some hardware platform. Most often the executive is the same as where the plan engine is running. External micro task services such as Maci, presented in the previous chapter, raise a need for platform definition, for example micro tasks on another robot than the one where the plan engine is running. In most cases the platform is defined as “Local”, which describes a platform that is the same as the plan engine. More target platforms will be presented in the future.

4.4.4.3 Attributes of parameter element

The parameter element has attributes named type, value, name, and description. These attributes are the same as the ReturnValue element, as seen in the PlanLang XML example. The types of parameters are presented in Table 5.

Table 5. Parameter types defined in PlanLang.

Parameter type	Definition
Unknown	Undefined parameter value. Not used in the plan engine.
Bool	Boolean variable. Values are 1 or 0 (yes or no).
Integer	Integer number
Double	Floating point number
String	Textual parameter
Object	Database object value
Pose	Pose of the object
Position	Position of the object
Area	Area object defining, for example, an area on a map
Node	Node database object
Pixel	Image pixel
MultiPixel	Undefined amount of pixels from the image
Struct	Structural object
Anything	If the type is unknown, this can be used.

4.4.4.4 Configuring tasks with PlanLang

There are different means of programming in PlanLang. The executable form of PlanLang is its XML representation. While one can write PlanLang plans directly in XML, this is generally not practical. The official programming for PlanLang is done using visual programming with a tool called PlanGUI. More details about using PlanGUI are provided in the next chapter.

4.5 Planex - Plan Executor

Planex is a combination of a plan configuration tool and a task execution engine. Planex has an interface for the data flow in user interaction and also for lower-level applications and robots. The plan engine is responsible for executing plans and handles information flow between all the modules. PlanGUI is a graphical user interface for configuring plans and monitoring plan execution. User interface and task execution engine are joined in the same module because of offering testing environment for configured plans. Task configurer can execute plan flow without direct link to the robot in the working area.

The next diagram, Figure 20, presents the structure of Planex. In the figure a realized plan written with PlanLang is shown on a model of the PlanGUI screen. PlanGUI communicates with the plan engine, allowing the task configurator to send and monitor new plans and their execution. The plan engine controls configured micro tasks under its state machines and takes control commands from the end user. eTasks in the micro task library are software implementations of the micro tasks. A more detailed presentation of Planex is presented in the following sections.

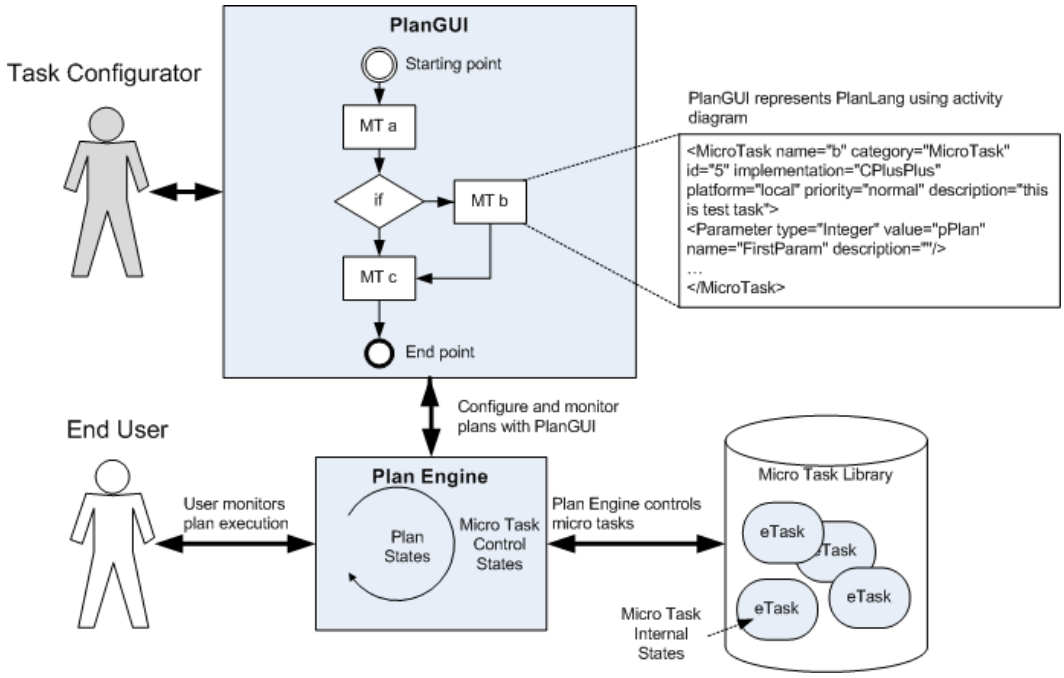

Figure 20. Structure of Planex.

4.5.1 Data resources

Parameter fields in PlanLang carry information in and out from the micro tasks. In addition to the numbers being written directly into the value fields in PlanLang a priori, there are also other

sources for the data. The data can come from the user or from some other micro tasks executed before the present one. Furthermore, the data can be obtained from the database, it can be measured by using perception functionalities, or they can be calculated from some other values.

One important data source is obviously the user, who uses the available parameters under the plan element in PlanLang. The values for these parameters are given through the user interface. The following example presents a simplified example of how data are carried from the plan level to the micro tasks. (Three dots represent parts of PlanLang that are left out because of their irrelevance to the case). Under the plan element there is one parameter named pPlan. The value field is left empty, meaning that the value will be asked for from the user. The next micro task, named MyTask, holds a parameter FirstParam. The value of FirstParam is called pPlan. Thus, during the execution of this plan the original pPlan will define the value of FirstParam in this micro task.

```
<Plan name="MyPlan" …>
<Parameter type="Integer" value="" name="pPlan" description=""/>
<MicroTask name="MyTask" category="MicroTask" id="3" implementation="CPlusPlus" platform=”local”
priority="normal" description="this is test task">
<Parameter type="Integer" value="pPlan" name="FirstParam" description=""/>
…
</MicroTask>
</Plan>
```

Data are transferred between micro tasks in the same way as presented above. For example, the parameter value field of prevailing micro tasks can have the same name as the name of the return value of the previous micro task. This notation can be thought of as being like global variables in common computer programming languages. Many micro tasks can be imagined to be data collectors instead of being locomotion- or manipulation-related tasks. Some micro tasks make calls to the database and retrieve information from the working environment. All directly perception-related micro tasks can be considered as data collectors, because most often perception tasks sense the environment and return measured values.

4.5.2 Plan engine

PlanEngine is a thread-based finite state machine (FSM) executing a plan and controlling the execution of all the micro tasks. PlanEngine has two FSMs running. The higher-level FSM is called plan_FSM. This machine controls the execution order of a plan. The lower-level state machine, called uTask_FSM, is for controlling the execution of the micro tasks. The plan-level state

machine is controlled with events sent from the end user or from the micro task level. The micro task control level is controlled by events from the higher plan level and from the micro tasks.

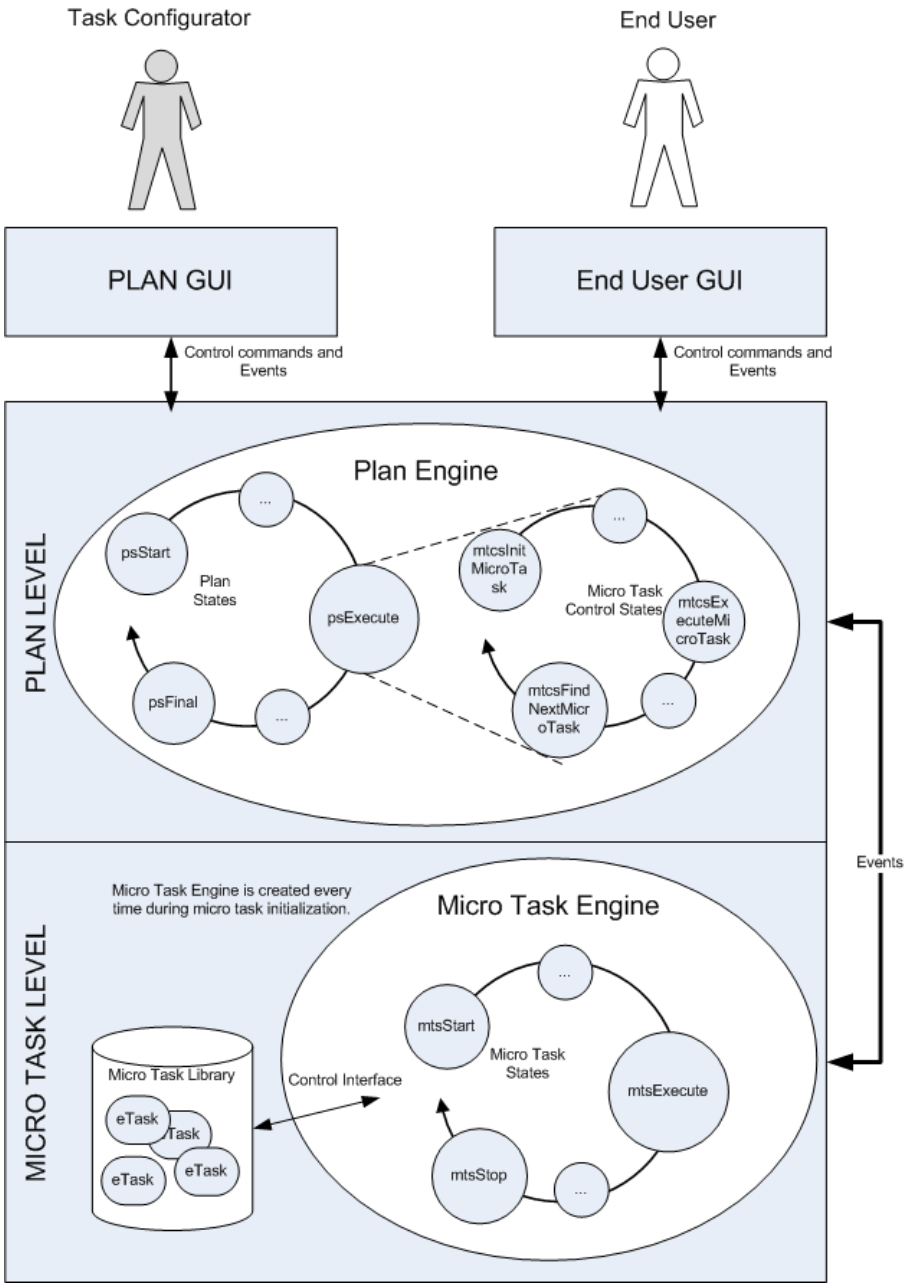

Figure 21. Processes and information flow of plan control engines.

PlanEngine interprets the configured (with PlanGUI) plans and controls the execution flow. PlanEngine has continuous communication to PlanGUI in order to react to plan changes made by the configurator (an experienced user). The thread-based approach means that every micro task is executed inside one of the independent threads. This provides more robust execution and parallelism for the micro tasks. The information needed for the micro tasks is collected from

various data sources. For example, if the parameter value attributes are empty, the data are asked for from the user. If the name of the value field in PlanLang is the same as the return value of some other previous micro task, the plan engine adds the correct value to that field.

The execution of a plan proceeds in discrete steps in plan_FSM. One of its states, called psExecute, includes a specific Micro Task Control FSM, which handles the task-level control of the selected micro task. In addition, the execution of the micro tasks proceeds in steps in uTask_FSM. All external events are processed in the order in which they are received. An external event and all its internal effects are processed before the next event is processed; this behavior is known as run-to-completion semantics [Selic, Gullekson, & Ward 1994]. In the following sections these state machines are presented in detail.

4.5.2.1 Plan states

Table 6 presents the discrete steps of the execution process of the plan. After the successful initialization of the plan, the state is changed to psExecute. The execution is continued in another state machine called micro task control. The states under micro task control are presented in Figure 22 and more detailed presentation in Table 7. The letters from A to O presents the transfer functions of the plan.

Table 6. States of the plan engine.

State	Process	Transfer function	Next state(s)
psNull	Default state. Do nothing.	NOT (A)	psNull
		"Start the plan" command from the user. (A)	psStart
psStart	Pre-initialize the plan.	Pre-initialization of the plan is done. (B)	psInit
		Engine startup failure. (C)	psError
		NOT (B OR C)	psStart
psInit	Initialize the plan.	Initialization of the plan is done. (D)	psInitDone
		Initialization failure. Give feedback to the user. (E) - PlanLang syntax error - Parameters missing or incorrect - No start task	psError
		NOT (D OR E)	psInit
psInitDone	Finalize initialization. Send feedback to the user.	Finalizing initialization is done. (F)	psExecute
		NOT (F)	psInitDone
psExecute	Start execution of the plan so start execution of the micro tasks. See micro task control states in next chapter.	User sends stop command. (G)	psStop
		User sends pause command. (H)	psPause
		Error in execution. (I)	psError
		NOT (G OR H OR I)	psExecute

psStop	Shut down the plan execution.	Shut down the plan execution is done. (J)	psFinal
		Error during shut down. (K)	psError
		NOT (J OR K)	psStop
psFinal	Stop engine thread.	*	psFinal
psPause	Pausing execution.	User pauses plan execution. For example, teleoperation of the robot is possible in this state. (L)	psWaitPause
		NOT (L)	psPause
psWaitPause	Waiting for continue command from the user.	Users send continue command. (M)	psContinue
		NOT (M)	psWaitPause
psContinue	Initialize execution.	Initialization is done. (N)	psWaitContinue
		NOT (N)	psContinue
psWaitContinue	Starting up the execution.	Starting up the execution is done. (O)	psExecute
		NOT (O)	psWaitCOntinue
psError	Error happened. Feedback to the user. Stop thread	*	psFinal

Figure 22. States of the plan engine.

4.5.2.2 Micro task control states

Micro task control states (in Figure 23 and Table 7) are handled under the plan engine. These states are extensions under the plan state psExecute. The division into plan states and micro task control states is for the reason of simpler notation. This could be thought of as plan states being controlled by the user and micro task control states being controlled by the execution routines of micro tasks.

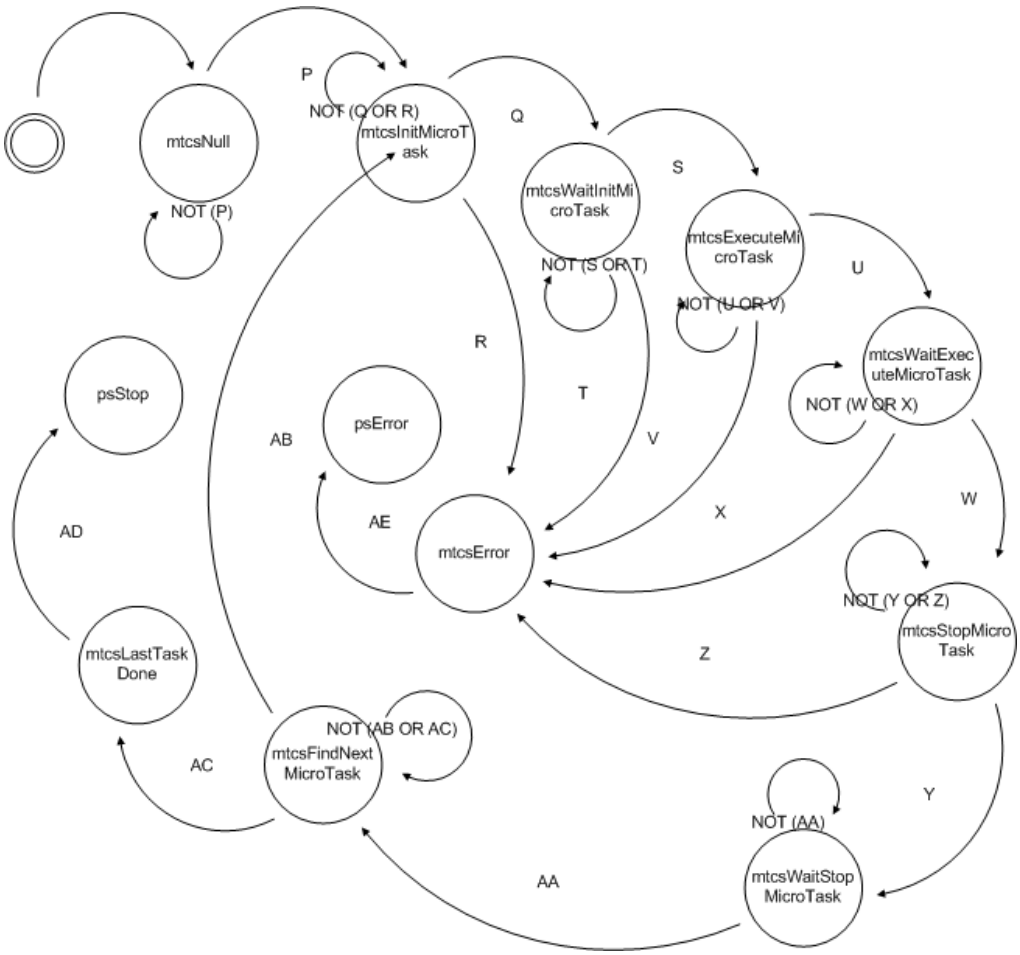

Figure 23. States of the plan engine when controlling micro tasks.

Table 7. States of the plan engine when controlling micro tasks.

State	Process	Transfer function	Next state(s)
mtcsNull	Default state. Do nothing.	NOT (P)	mtcsNull
		Initialization command from the plan engine. (P)	mtcsInitMicroTask
mtcsInitMicroTask	Create and start micro task engine. Initialize micro task and send parameters.	Initialization is started. (Q)	mtcsWaitInitMicroTask
		Error in the initialization. (R)	mtcsError
		NOT (Q OR R)	mtcsInitMicroTask
mtcsWaitInitMicroTask	Waits for initialization done event from the micro task.	Initialization done event received from the micro task. (S)	mtcsExecuteMicroTask.
		Error event received from the micro task. (T)	mtcsError
		NOT (S OR T)	mtcsWaitInitMicroTask
mtcsExecuteMicroTask	The execution of the micro task. The feedback information is received from the micro task.	Execution is started. (U)	mtcsWaitExecuteMicroTask
		Error during startup of the micro task. (V)	mtcsError
		NOT (U OR V)	mtcsExecuteMicroTask
mtcsWaitExecuteMicroTask	Wait for execution done event from the micro task. The feedback information is received from the micro task.	Execute done event received from the micro task. (W)	mtcsStopMicroTask.
		Error event received from the micro task. (X)	mtcsError
		NOT (W OR X)	mtcsWaitExecuteMicroTask
mtcsStopMicroTask	Stopping micro task.	The execution of the micro task starts to stop. (Y)	mtcsWaitStopMicroTask
		Error event received from the micro task. (Z)	mtcsError
		NOT (Y OR Z)	mtcsStopMicroTask
mtcsWaitStopMicroTask	Wait for stop done event from the micro task. The return values are received.	Stop done event received from the micro task. (AA)	mtcsFindNextMicroTask
		NOT (AA)	mtcsWaitStopMicroTask
mtcsFindNextMicroTask	The next micro task for execution is looked for.	The next micro task in the plan is found. (AB)	mtcsInitMicroTask
		No more micro tasks in the plan. (AC)	mtcsLastTaskDone
		NOT (AB OR AC)	mtcsFindNextMicroTask
mtcsLastTaskDone	All the needed micro tasks in the plan have been executed.	If all the needed micro tasks have been executed the plan execution will be stopped. (AD)	psStop.
mtcsError		Error state during micro task control. (AE)	psError

4.5.2.3 Micro Task Internal States

During the micro task initialization phase a independent thread for micro task execution is created. The state machine running under this thread has several states, which are presented in Figure 24 and Table 8. Micro task control states in plan engine drives the execution of micro task engine. Micro task engine with micro task internal states drives and communicates with eTasks (on the robot for example).

Table 8. States of the engine executing micro tasks.

State	Process	Transfer function	Next state(s)
mtsNull	Default state. Do nothing.	NOT (A)	mtsNull
		Plan engine creates thread for micro task. (A)	mtsStart
mtsStart	Micro task thread is started.	Plan engine sends initialization command. (B)	mtsInit
		NOT(B)	mtsStart
mtsInit	Initialization of the micro tasks. Parameters are received from the plan engine.	Initialization is started. ©	mtsWaitInit
		Initialization error (D)	mtsError
		NOT (C OR D)	mtsInit
mtsWaitInit	Wait until initialization is done.	Initialization is done. (E)	mtsInitDone
		Not (E)	mtsWaitInit
mtsInitDone	Initialization is done. Sending status to plan engine.	Plan engine sends execute command. (F)	mtsExecute
		NOT (F)	mtsInitDone
mtsExecute	Starts the execution of the micro task. Poll implemented micro tasks to get feedbacks. Sending status to plan engine.	Execution is done. (G)	mtsExecuteDone
		Startup error. (H)	mtsError
		NOT (G OR H)	mtsExecute
mtsExecuteDone	Wait for command from the plan engine.	Execution is done. (I)	mtsStop
		NOT (I)	mtsExecuteDone
mtsStop	Stop the execution. Send return values to the plan engine.	Micro task is stopped. (J)	mtsStopDone
		Stop error. (K)	mtsError
		NOT (J OR K)	mtsStop
mtsStopDone	Sending status to plan engine.	*	mtsStopDone
mtsPause	Pause execution.	Pause command received from the plan engine. (L)	mtsWaitPause
mtsWaitPause	Wait until plan engine commands to continue.	Continue command received from the plan engine. (M)	mtsExecute
mtsError	Error inside of the micro task. Stop micro task.	Error state. (N)	mtsStop.

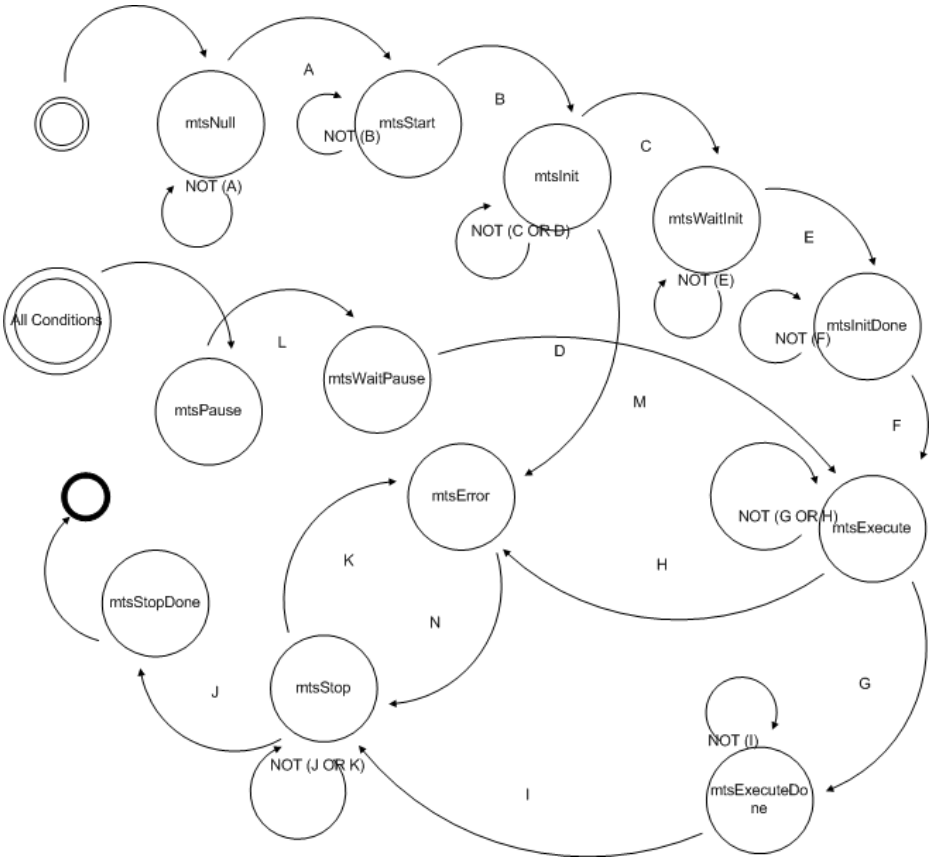

Figure 24. States of the engine executing micro tasks.

4.5.3 PlanGUI

PlanGUI is a tool for configuring plans. Configuring is done using visual programming. The activity diagram is the base format for the plans. In Figure 25 a screen shot of Plan GUI is presented. PlanGUI is programmed with the C++ language and runs in a Windows environment.

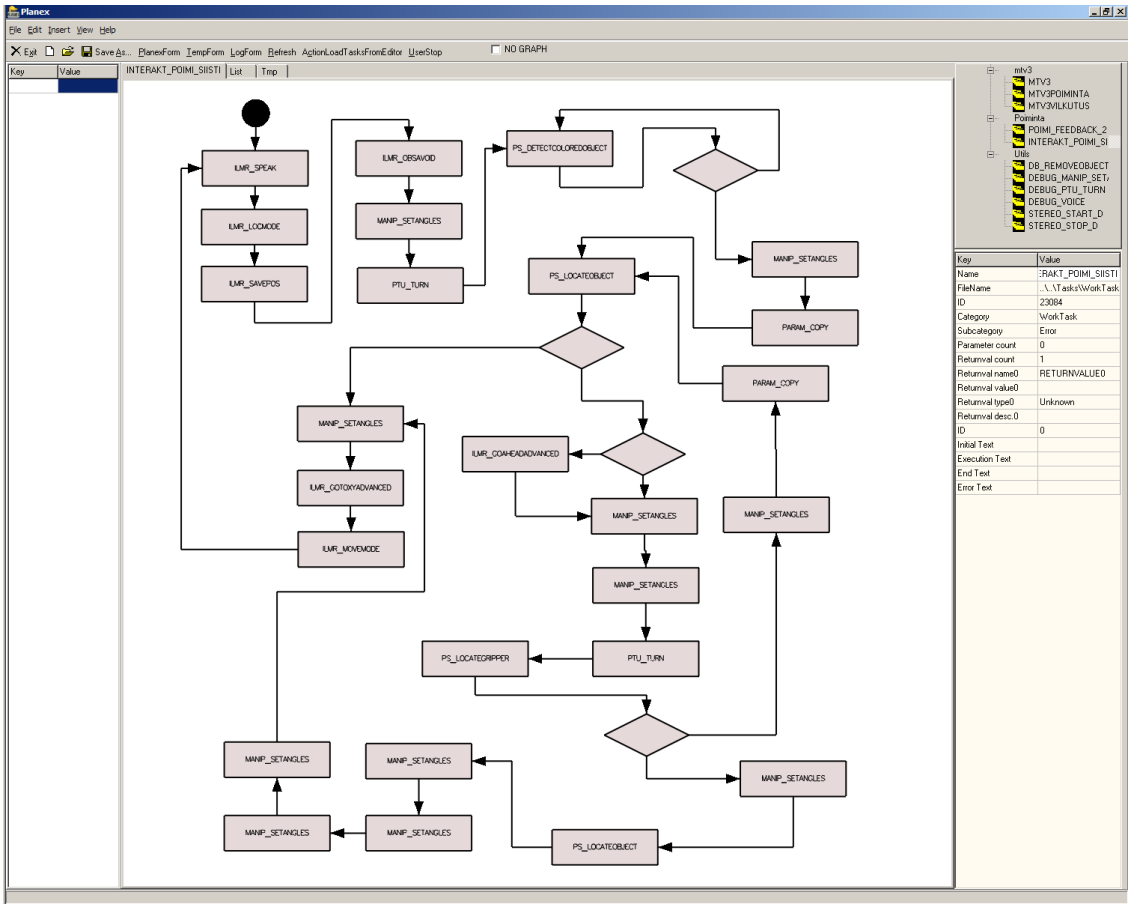

Figure 25. Screen shot of PlanGUI.

The configuring of the new plan is done by initiating micro tasks from the list on the right of the screen. They appear as a box in the workspace in the middle of the screen and can be moved around and connected to other boxes with a computer pointing device. When the task box is activated by clicking, its parameter list appears in the lower right-hand corner of the screen for editing. When no task box is active, the parameters of the plan are presented.

Planex interprets the work task structure by following the arrows from one block to another. These arrows, which are included in the PlanLang file, define the whole execution order and branching of the blocks. The arrows are defined in PlanLang using connection elements. PlanGUI is not only for configuring new tasks but also for monitoring their execution. During plan

execution the user can see all the micro tasks and the prevailing ones are highlighted. The status of the execution and the values of the variables are also presented on the screen. The debugging of new plans is one case where the monitoring is needed.

4.5.4 Implementation of micro tasks

As mentioned earlier, PlanLang specifies the definition part of the micro task. The implementation part (the real executable part) is an external resource for PlanLang. In a way, PlanLang defines how to interface with functionalities located elsewhere. The implementation of the micro tasks is programmed by using the C++ language and called eTasks. In the present software implementation, all micro tasks are embedded inside the Planex program. To ensure the expandability of the platform, the micro tasks need to be divided into separate units. In the Windows environment they could be implemented as DLL files, and in Linux or similar environments as SO files. Furthermore, in the future the micro tasks will be implemented as external resources operating through a network interface. With this kind of solution the micro tasks can be located anywhere on the internet. The micro tasks are implemented as classes. An example of a base class of a micro task is presented in Appendix 1.

There are different kinds of eTasks from the point of view of their behavior. Many eTasks control some primitive actions in a robot. For example, gotoxy is a typical task. Additionally, eTasks might provide perception services for the plan. In that case they communicate with a service provider, such as an image processing unit. An information request is also a typical task for eTasks. A database located in the network is the source for the information and some eTasks interact with it. For example, these tasks might ask the database for the features of an object found with a camera. For controlling the flow of the plan there are also some special eTasks. These tasks are start, stop, and if tasks. They are hard-coded inside the plan engine but the controlling is done in the same way as in the other eTasks. The start eTask is especially important. The plan engine goes through PlanLang in order to find the start command. It indicates the beginning of the plan.

4.5.5 Micro task library

Planex holds a library of plans and micro tasks for executing the required missions. This library is basically a list of implemented micro tasks and already-configured plans. It acts as a reference for the user when configuring new plans by dragging and dropping micro tasks onto the screen of PlanGUI. The library is also used when executing plans, because it includes the

information on where the tasks are located. So the plan engine can control the implemented micro tasks through virtual communication channels, regardless of where the actual physical implementations are located. The micro tasks implemented are listed in Table 9. Detailed description is presented in Appendix 3. Furthermore, an in-depth presentation of the behavior of ILMR is presented in [Kauppi 2003]. Abbreviations in the table are: p1=First parameter value of micro task, p2=Second parameter etc. and r1=First return value of micro task etc.

Table 9. List of implemented micro tasks.

Micro Task	**Type**	**Description**
MT_ArithmeticOperation	Calculation	Make calculation between the input parameters. For example: p1=+, p2=1, p3=3 so r1=4
MT_ComparePlanexVariable	Calculation	Compare input parameters. For example: p1=1, p2=<, p3=3 so r1=true
MT_db_addObject	Database	Add object to the database. Typical objects are physical object detected with the perception system.
MT_db_addUser	Database	Store the user name specified with a color to the database.
MT_db_getObjectInfo	Database	Return information of database object.
MT_db_getUsers	Database	Return list of users in the database.
MT_db_RemoveObject	Database	Remove database object.
MT_db_setObjectPosition	Database	Set position for database object.
MT_db_typeAndNameSearch	Database	Search object from the database.
MT_ILMR_BodyPose	ILMR	Set robot's orientation.
MT_ILMR_Goahead	ILMR	Move a robot ahead or back
MT_ILMR_GoaheadAdvanced	ILMR	Move a robot ahead or back. Automatically change move mode etc.
MT_ILMR_Gotoxy	ILMR	Move a robot to a desired position
MT_ILMR_GotoxyAdvanced	ILMR	Move a robot to a desired position. Automatically change move mode etc.
MT_ILMR_Locate	ILMR	Initialize the coordinate system.
MT_ILMR_locMode	ILMR	Select a navigation mode
MT_ILMR_MoveMode	ILMR	Set a mode for movement in a robot.
MT_ILMR_ObsAvoid	ILMR	Select a behavior for a robot when it encounters an obstacle
MT_ILMR_savepos	ILMR	Save the current position to be used later.
MT_ILMR_Speak	ILMR	Speak out from the speakers on the robot.
MT_ILMR_Speed	ILMR	Set speed for a robot
MT_ILMR_Turn	ILMR	Make a robot turn to the left or right
MT_Manip_SetAngles	Movement	Move joints of the manipulator to specified position.
MT_ManipulatorRightXYZToAngles	Movement	Move arm of the manipulator to specified position.
MT_PTU_Turn	Movement	Turn the head of the robot to specific position.
MT_Laser_WaitForDistance	Perception	Measure distance using laser range finder.
MT_PS_ColorNameToColorInfo	Perception	Get color data values for selected color name.
MT_PS_detectColoredObject	Perception	Detect if there is object with specified color on the camera image.
MT_PS_detectUser	Perception	Detect if the user (from database) is on camera image.
MT_PS_LocateGripper	Perception	Measures the position of the robot's hand.
MT_PSAnalyzeColor	Perception	Analyze the color of selected point on image screen.
MT_PSFindLargestObject	Perception	Find targets area of specified color from the camera image.
MT_PSLocateObject	Perception	Measure the position of object related to the robot. Object is specified with a color.
MT_PSSick_detectUser	Perception	Detect the user using data from the SICK laser scanner and from

		camera.
MT_PTU_searchObject	Perception	Search object from the camera image while turning the head of the robot.
MT_stereo_start	Speech	Start stereo hearing.
MT_stereo_stop	Speech	Stop stereo hearing.
MT_UIspeak	Speech	Speak out from the user interface computer.
MT_Voice	Speech	Control the speech recognition and synthesizer.
MT_DebugToFile	Util	Write debug information to the text file.
MT_generateGreeting	Util	Message dialog generated to the screen.
MT_GetWopaVariables	Util	Read measurements from the robot. This micro task returns direct measurement values from the robot. This mainly used for debugging.
MT_param_copy	Util	Copy values between the variables.
MT_SetWopaVariables	Util	Send direct command to the robot. This mainly used for debugging.
MT_Sleep	Util	Delay.

5 Configuration of Tasks for Multimodal Service Robot

5.1 Introduction

Human-friendly communication is seen as a method of interaction which is similar to interaction between humans. There are several different kinds of communication methods to choose from human-robot interaction (HRI). Currently, the most common interaction method has its roots in computer science. Human-computer interfaces facilitate communication, assist in the exchange of information, and process commands and controls, among many additional interactions. Visual or text-based displays on a computer screen are the easiest way to create a user interface for robots. However, this is not the only way. Goodrich has listed the media used in HRI as follows [Goodrich, Schultz 2007]:

- Visual displays, typically presented as graphical user interfaces or augmented reality interfaces
- Gestures, including hand and facial movements and movement-based signaling of intent
- Speech and natural language, which include both auditory speech and text-based responses, and which frequently emphasize dialog and mixed-initiative interaction
- Non-speech audio, frequently used in alerting
- Physical interaction and haptics, frequently used remotely in augmented reality or in teleoperation to invoke a sense of presence, especially in telemanipulation tasks, and also frequently used proximately to promote emotional, social, and assistive exchanges.

Natural language (speech), gestures and touch are the most familiar and effective ways of communication between humans. Why not provide equally natural interfaces to communicate with robots? Using easy and friendly multi-modal interfaces between humans and robots might allow the users to concentrate more on the task at hand, rather than on the tools of the interface.

The target of the user interface is to offer easy interactive control of a versatile and complex robot in outdoor conditions. Controlling the work tasks involves mainly supervisory commands, but also the possibility of closed loop (straight) teleoperation.

5.2 Multimodal User Interface for Task Control

Humans are naturally multimodal when advising another person in his work. Typical instructions are ones like “Put that over there”, “Come here”, and “Follow me” etc., where people use both speech and pointing at the same time. The goal is not the concurrent use of communication methods (the instruction examples above) but a fluent communication process in all circumstances. Human users benefit from the multimodal user interface in robots. Configuring and controlling tasks using multimodal user interface is human-friendly and an efficient way to improve work performance. Dialog between user and robot is necessary. They may need to give instructions to each other during different phases of the execution of the task. For example, if the human and robot are carrying an item together, they need to communicate in order to cooperate better.

People are often very adaptive and change their methods of interaction fluently. For example, they might change from speech to gestures under noisy conditions. The distance between conversation partners also has its own effect. When communicating with someone nearby, speech alone is enough, but over a long distance some kind of technical device (radio communication, for example) is needed. The flexible use of different communication methods is important for fluent co-operation with service robots. The question is not how to make robots communicate like humans but how to make communication as easy as possible.

The interaction between robot and user can take place in various ways. Service robots usually work in an environment where there are no desktop computers nearby. However, operating the robot usually requires some kind of a portable computer, laptop, tablet PC, or PDA-type computer. The limitations of robots, such as insufficient ability to recognize shapes or objects with their camera systems, may require some ancillary equipment to help communication. The user can see the image from the robot’s camera, map, or state of execution of the task at hand. The user can, for example, point out the objects to be manipulated from the image or map. Traditional human-computer interfaces such as graphical user interfaces and control devices such as joysticks and mouse devices are familiar to most people. The whole interface should be appropriate combination of different kinds of interaction methods.

The interaction can also take place in the real world. The objects of interest can be marked with signs. For example, a bar code can be used to identify a box that should be lifted. Signs can also be used to determine the area where the robot is supposed to operate. In addition, the user can use gestures or show the objects to be manipulated with a pointing device, such as a distinctively colored stick. These kinds of signs and devices are very easy to use. Configuring the tasks is easy because the signs are merely carried to the working area and no other actions are needed. This is natural for human user who does not think of a location as position coordinates unlike robots do. The next section presents more details of using signs in task configuration.

5.2.1 User interface as a part of task description

Using signs as part of the task description is a good way to reduce the structure of a task under configuration. The approach is to use movable landmarks to provide relative, rather than absolute, location information. The appearance and position of the sign may also give more information. Movable and varied-looking signs are suitable for defining tasks for service robots. The signs can be used for pointing a direction, bounding an area, marking a route, or defining the location of a target. An example of using signs is presented in Figure 26. Movable signs can be used in a variety of ways. In the example pointing signs are placed on the front of the robot. They show the directions to the area to be cleaned. Driving onto the playground is prevented by a danger sign.

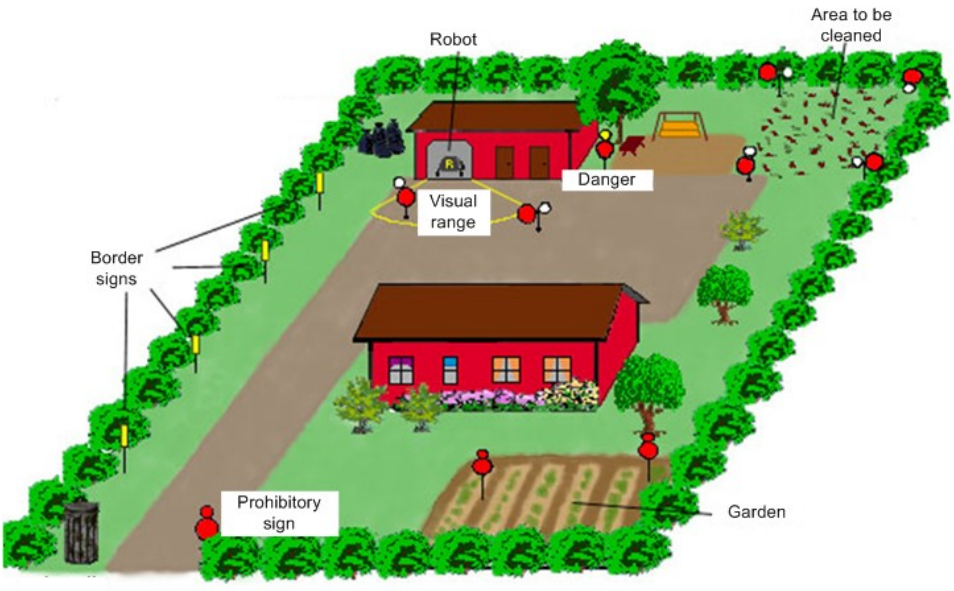

Figure 26. Example of using signs as a part of task description.

The signs can be divided into two main categories. Passive signs contain all the information in their appearance. For example, normal road signs belong to this category. In active signs the information content can be modified, for example with radio communication. Moreover, active signs can be equipped with sensors, such as a GPS receiver and compass, so that their location on a map can be determined.

The concept of using active markers placed in the operating environment for exchanging information between robots and the environment was introduced by Asama [Asama 2001]. The markers, called Intelligent Data Carriers (IDCs), are portable electronic data carrier devices. Wireless data exchange between IDCs and robots is realized through RF-ID weak radio communication. The main application area planned for the IDC devices was related to the cooperative organization of multiple robots and their operating environment, for example, information on existing free routes in an unknown environment being exchanged between the members of a robot society by means of IDCs. When a robot member detects that a route leads to a dead end, it can send the information to an IDC placed at the beginning of the route when it comes back from the dead end. Heikkilä et al. [Heikkilä et al. 2006] presents the first results of using IDCs or, as they are called, active signs for configuring work tasks for service robots. In addition to active signs, more simple passive signs are introduced as an alternative or complementary tool for passing information on the task plan of a work task from a human operator to a service robot.

5.3 Selection of Modality in the Human-Robot Interaction

People naturally and automatically select communication modality while interacting with each other. Thus human-robot interaction becomes more natural if the user has some freedom of choice with regard to modality. Basically the human user informs robots of his intentions and aims using the user interface but robots are still not sophisticated enough to fully understand the user's message. Therefore limitations exist in the selection of modality.

The modality of interaction is selected in the configuration process but it's also possible to select it while executing the plan. On the basis of his knowledge of the task, the configurator can select the most suitable modality for a task description during the configuring process. Communication modality is selected by adding micro tasks related to interaction to the plan. For example *mt_voice* is a speech recognition related micro task. Adding a mt_voice micro task to a specific phase of the plan the configurator defines that the robot should ask something and listen

out for a reply. Describing the whole dialog between the user and robot is not necessary, but the main interfaces and tools should be the ones most suitable for the task. There are also other micro tasks for multimodal interaction such as *mt_stereo_start* (for stereo hearing), *mt_ps_detect_user* (for recognize and detect user) or *mt_ps_detectColoredObject* (for detecting pointing with a scepter (see Section 5.5).

The end user has still the last word to in selecting modality of interaction. Information asked during the execution of the plan is communicated by the end user. This is provided by leaving fields of parameter values in the plan empty. For example, the task configurator has made a cleaning task for cleaning the yard. The area to be cleaned, however, is left empty in the plan description. The robot will ask for the location during the cleaning task start-up. The end user can give the location to be cleaned by clicking it on the map or by speech (saying “yard”). This is possible because the data transferred through the user interface channels are categorized in as common a way as possible. For example, ‘pose’ includes detailed information of position and direction. Furthermore, in task control, if the robot asks for the pose of something the user can select whatever interface is suitable to send the pose.

The work tasks should be configured in such a way that the parts of the task are independent of each other. In this way it is possible to bypass some parts without causing problems in the later phases. A problem might emerge if the user wants to bypass a measurement task, and later the results of the measurement need to be utilized. In this case the robot will again ask the user for guidance until the necessary preparatory phases have been executed, or the tasks have been bypassed.

The PlanGUI (see Chapter 4) interface helps the user both in the plan-configuring phase and in the execution phase. In the configuring phase possible mistakes are prevented by giving enough information to the user. In the execution phase the current state of the tasks is shown.

It is not always possible to complete the entire task with predefined sequences. Sometimes the robot might need help from the operator. If the robot encounters an assignment that it cannot complete on its own, it can interrupt the execution and wait for the user to guide it to the next phase. When the user has led the robot to complete or bypass the work phase(s) in question, the work task can continue from the next phase or another phase determined by the user.

5.4 Task Execution with Graphical User Interface

Traditional graphical user interfaces (GUI) are very powerful in many situations. Instead of carrying a laptop a user could have smaller ones such as a PDA-type computer, mobile phone or a tablet-PC with a touch screen. People are familiar with mouse and joysticks so using them an input device for the robot control would be user friendly. Mobile robotics brings added requirements for the GUIs. Map of the environment is a basic feature on the screen. Controlling tasks demands for multimodal communication methods with a GUI (mentioned in the previous section). The user should have a choice to start-up and control the execution of selected tasks. The user should have also a continuous feeling of what the robot is doing.

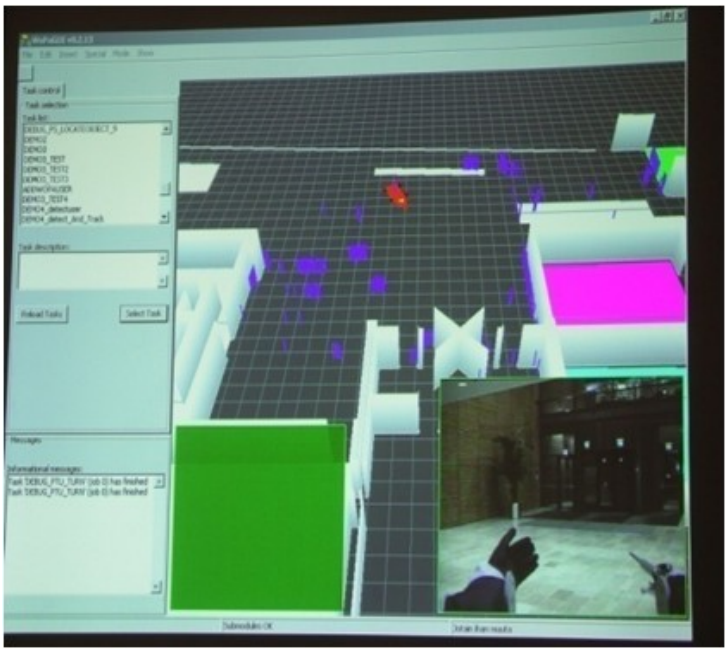

Figure 27. Screenshot of End User GUI.

The graphical user interface (see Figure 27) for the end user is a part of the platform (presented in Chapter 4) which offers many interaction methods to control tasks. Movement tasks usually require the user to point out one or more waypoints on a map, or define a trajectory that the robot needs to follow. In addition, the user may need to define an area that the robot needs to process, for example to pick up all the objects from the area in question. Perception tasks may require a definition of the properties of the objects of interest. If the properties are stored in the database, the user can give their name directly. A simple example of a named object type could be “red ball”. Another way to define the appearances or other properties of the objects is to point

them out, either on a map or in an image. The latter is usually more reliable when the objects are defined by their visual properties.

When pointing out a location on a map or a point in an image, GUI passes the coordinates to the work task. The task in Planex decides what to do with the coordinates. If a point in the image is defined, the work task probably acquires the most recent camera image and analyzes the respective pixel and its vicinity for additional information. If this information is needed in later phases, it is stored in the database.

Running the tasks allows the user to monitor the execution's progress. The blocks are highlighted and their internal state is observable in the user interface. This allows user intervention during runtime increases human-robot cooperation. If some part of the program fails, the user may suspend the execution, help the robot manually to the desired state, and continue the program from another part of the task.

5.5 Human-Robot Interaction in the Task Control Platform

Figure 28 shows communication modalities used in task control platform (presented in the Chapter 4). The communication methods are implemented and tested with the WorkPartner robot.

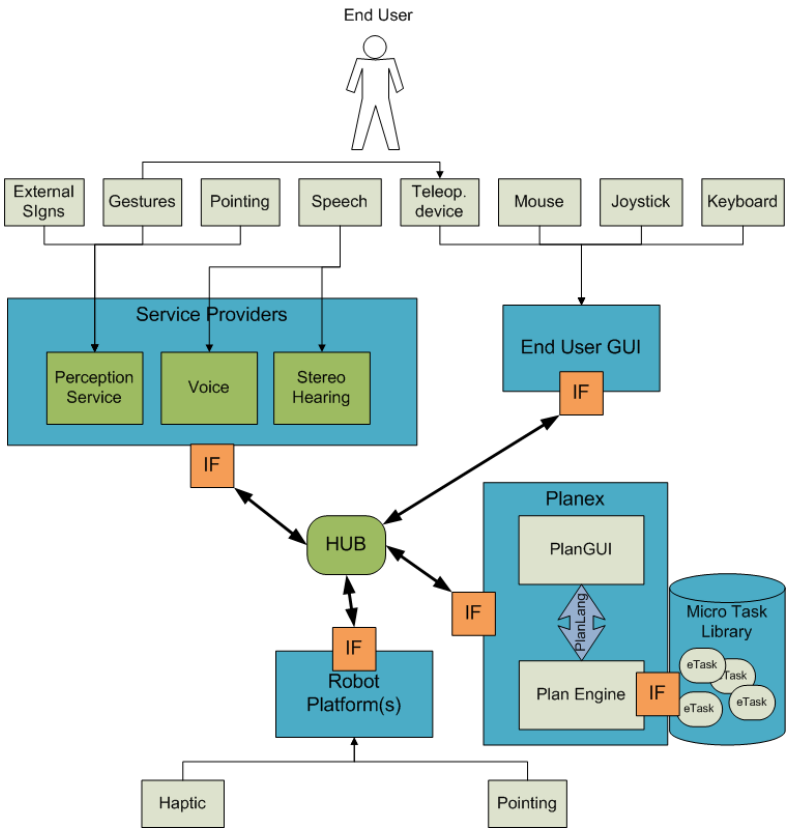

Figure 28. Communication modalities in task control platform.

The goal in WorkPartner is for most of the interaction to be human-like conversation by means of speech and gestures, so as to minimize operator hardware. WorkPartner is presented in greater detail in Chapter 6.

Multimodalities supported by WorkPartner today are (see Figure 28):

- Commercial speech recognition for user commands with a simple command set
- Stereo hearing to detect voice direction
- Speech synthesizer in the robot
- Gesture recognition either by the robot's color camera or by special mechanical wearable equipment (see Figure 29)
- Pointing either on a map screen or at a camera image, using an onboard laser distance meter or a device called a scepter (stick with a red ball) (see Figure 30)
- Haptic interface for leading the robot (see Figure 11)
- A portable sign/beacon system by which the user can communicate indirectly with the robot (see Section 5.2.1).
- Several graphical user interfaces on a computer screen controlled with mouse, keyboard or joystick.

5.5.1 Speech

Speech is a natural communication method of communication between people, and it is suitable to be used in most situations. Speech is also the most important communication method when developing human-friendly interaction between humans and robots. Recognizing and producing speech go hand in hand in humans, but when it comes on robotics these should be separated. Speech synthesizers have for many years been used in computers and their quality of output is reasonably good. Speech recognition systems have taken huge steps but there are still reliability issues. The results in recognition are quite good in office environments but in mobile robotics the noisy environment is a problem. Safety is another question in mobile robotics: for example if the user commands the robot driving towards him to "Stop" there is no doubt that this command should be recognized. Therefore robots are mostly commanded with relatively few discrete commands to maintain reliability.

The detection of who is speaking is essential for people. For example, in public places human can turn their head to the direction where the sound (speech, yell, etc.) is coming from. In

fluent communication this information on who is communicating is important. The robots should also know who is commanding them. Two human ears are effective sensors for measuring the direction of sound and this can be replicated in robots with two microphones. Stereo hearing as a part of task control is a good method for detecting the user (see Chapter 6).

5.5.2 Gestures

Figure 29. Command given by a gesture. Gesture is determined by detecting the yellow jacket and calculating the form of detection. User is also wearing a teleoperation device on his shoulders. This device can be used for direct teleoperation and for gesture recognition.

People use gestures with speech or instead of speech. Gestures are used to clarify spoken messages (for example, cross arms and say “Stop”) or spoken commands are exchanged for gestures (for example, in noisy environments or over long distances). Gestures are a good way to give commands. Commands are simple ones such as “Start”, “Stop” and “Follow”.

Gesture-based commands for robots can be given in many ways. For instance some methods are camera-based or make use of mechanical wearable devices (in Figure 29). In camera-based method a jacket with a specific color is analyzed by the camera and gesture shapes are recognized. A teleoperation device is wearable equipment used for measuring the 3D-position of wrists. Gestures can be recognized from these measured positions.

5.5.3 Pointing

People often guide by showing the direction with their hand and saying “Go over there”. Interaction by using hands and with the whole body is an easy way to communicate. Pointing is

used for indicating objects or places. People can track the direction of hands well but for robots there are limitations: the line of a hand is difficult to follow for robots. In a natural 3D environment it is difficult also for humans but they can solve the situation, for example, by using speech. Clear methods are needed to combine the spatially tied cognition of the user and robot.

Figure 30. Pointing at an object with the stick with red ball.

One way to point out objects is to use a stick with a red ball attached (in Figure 30). This method is founded on optical tracking based on color maps. The red ball is tracked like the user or gestures. The user can, for example, put the red ball near the object and say “Take this”. This method is easy to use and does not need any computer hardware to be carried by the user. However, it only works with short distances. The distance of the red ball is measured either from its size in a camera image or with a laser range finder. The exact position of the ball can be calculated based on distance.

A laser range finder on the head of the robot can also be used for pointing at objects. The user can turn the head to point at objects by using the joystick or teleoperation device. A red dot from the range finder shows where the head is pointing to. This method always gives a perfect result. Pointing from a camera image can be used over longer distances. The user simply clicks the object on the image with his mouse.

5.5.4 Haptics

Touch is one of the basic senses and an extremely important for humans. Haptic communication is the means of communicating through via touching. Touch is a good nonverbal

communication method between humans and it also provides information on surfaces and textures. Haptics in human-robot interaction is most often used as a feedback message from the robot. Force feedback, vibrations or motions enhance the sensation of actual operations whilst remotely controlling robots. Haptics can also be used for controlling the robots. Touch or tactile sensors are used to measure the pressure or force exerted by the user onto the interface. For example the user can touch the hand of the robot and give a command: ”Give me your hand. I will lead you.” In this case the forces exerted onto the robot’s arm are measured and converted to robot movement commands.

6 Experimental Verification

The proposed task control platform has been implemented and verified in a centaur-type service robot, WorkPartner, which was built at the Automation Technology Laboratory of Helsinki University of Technology [Halme et al., 2003].

6.1 WorkPartner as a Test Platform

WorkPartner (shown in Figure 31) is a centaur-like service robot for light outdoor tasks. Some possible tasks might be garden work, light forestry tasks, and guarding. The robot can be considered as being like the janitor of the future. The robot has four wheeled legs and a controllable body joint for effective hybrid locomotion and two hand manipulators for human-like tasks. WorkPartner was designed to be a multipurpose service robot. As a partner it should be capable of performing several different tasks either alone or in continuous cooperation with its master. New tasks (as well as other information) can either be loaded from the internet or an intranet or taught by the operator. The goal is for most of the interaction to be human-like conversation by means of speech and gestures, so as to minimize the operator hardware. However, different traditional and novel interfaces are being developed to help the common understanding between the robot and its operator, especially in teaching and teleoperation situations.

Figure 31. WorkPartner service robot in co-operation with people.

The WorkPartner robot is considered in more detail in [Halme et al. 2003]. The subsystems of the robot are reported in [Ylönen 2006]. Pictures, videos, and more information are available on

the project's webpage http://automation.tkk.fi/WorkPartner. [TKK Automation Technology Laboratory].

6.1.1 Mechanics

The WorkPartner robot is built on a mobile body called Hybtor (Hybrid Tractor), which is shown in Figure 32. The body of the robot has four 3 DOF legs equipped with wheels and a controllable body joint. The weight is about 270 kg and the payload about 40 kg.

Figure 32. Hybtor platform.

The mechanical parts were designed and manufactured by the Rover Company Ltd, St Petersburg, Russia. They used to make all the Soviet Union's planetary rovers, so they have extensive experience with special platforms. Because of its four wheeled legs and articulated body it has a very flexible and wide range of mobility. The locomotion system allows motion with the legs only, with the legs and wheels powered at the same time, or with the wheels only. The purpose of the hybrid locomotion system is to provide rough terrain capability and a wide speed range at the same time. The robot is equipped with a two-hand manipulator system about the size of the human upper body. The appearance and the size of the manipulator were chosen because of their similarity to what is needed for human tasks and the need for close co-operation with people. The figure and the simulator skeleton of the manipulator are illustrated in Figure 33. Simulator studies are considered in more detail in [Aarnio, Koskinen, & Ylönen 2001].

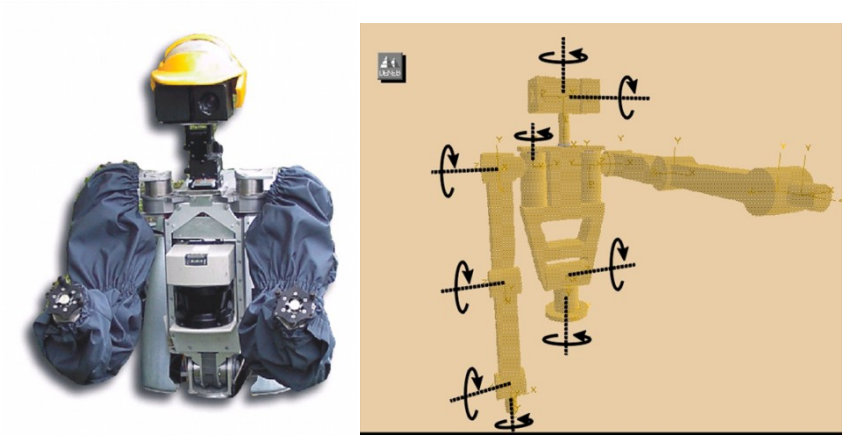

Figure 33. Manipulator of WorkPartner.

The manipulator consists of a 2-DOF (degrees-of-freedom) body, two 3-DOF arms, a 2-DOF camera, and a laser rangefinder head. All the joints of the manipulator are presented in a simulator skeleton in Figure 33. The manipulator can handle loads of up to 10 kg.

6.1.2 Power system

The hybrid power system consists of a 4-kW combustion engine and 4 batteries. The engine is a commercial 1-cylinder 4-tact type with an added catalytic converter and starter motor. The sealed led batteries supply 18Ah of energy. The engine is connected to the batteries by a commercial alternating current generator with a 48-V built-in regulator. The system is controlled by a micro-controller to ensure an optimal energy supply. With a 2-liter tank the system can supply energy for up to 3 hours of operation.

6.1.3 Electronics and computer system

The system architecture and all the subsystems of the WorkPartner robot are presented in Figure 34. The figure also presents the communication between the subsystems.

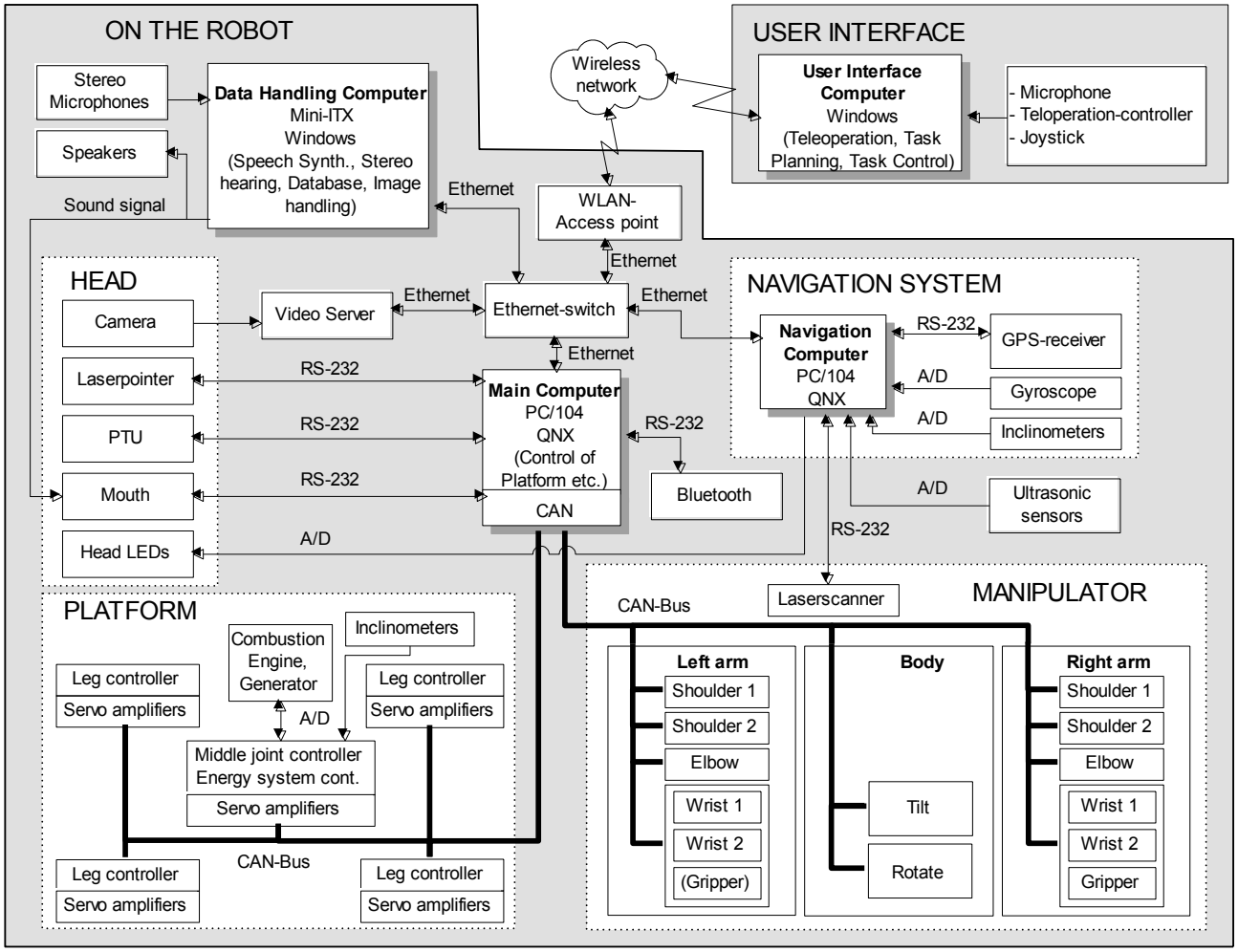

Figure 34. Subsystems of WorkPartner.

The main computer is the main controller for the platform. The focus of the main computer is on controlling all the modules and devices on the robot's body. It is also the communication router between the wireless network and the services on the robot platform. The main computer is a 586 PC-104 board and runs on a QNX operating system.

All the legs and the middle joints have their own control units. The leg control system is distributed around a CAN bus. Each leg has one controller (Leg Controller) based on a Siemens 167 Micro-controller and PHYTEC 167-mini-MODULE. The middle joint controller is built using the same components. The electronics also include servo controllers for the actuator motors and specially made amplifier cards for the force sensors in the legs and the hip actuators. The leg controllers communicate with the main computer via a CAN bus. The basic control strategy is that the main computer sends the xyz position of the ankle and the speed of the wheel.

The manipulator will also be connected to the CAN network. The main difference will be that the CAN network is directly connected to each joint, to each motor controller. The motor controller is built from a CAN module and a DS processor. The DSP motor controller is capable of providing position, speed, and force control. Only the head unit is a unit of its own, containing two degrees of freedom. The higher-level control software of the manipulator runs in the main computer.

6.1.4 Navigation system

The navigation system calculates the pose of the robot relative to the objects found in the close working environment. The basic navigation task is performed by detecting features in the environment and using them as beacons. The SICK laser scanner mounted on the manipulator (see Figure 33) is the main sensor for detecting features. The navigation methods used are based on simultaneous mapping and localization algorithms, which utilize fused information from an inertial sensor, wheel odometry, a laser scanner, and GPS. Selkäinaho [Selkäinaho 2002] presents the navigation system of WorkPartner in more detail.

6.1.5 Perception system

The main hardware modules of the perception system are a camera, laser range finder, laser scanner, and ultrasonic distance meters. There are also microphones for voice commands and a force sensing system for the legs.

Figure 35. The head unit of WorkPartner.

The SICK laser scanner mounted on the manipulator (see Figure 33 and Figure 34) is the main sensor for detecting obstacles while driving and for navigation. It is able to detect objects at a distance up to 50 meters. The whole horizontal scanning range is up to 180 degrees.

The camera unit mounted on the head (see Figure 34 and Figure 35) is the main sensor while finding and recognizing an object in the working environment. The camera is connected to the video server which shares the captured images over the network. The laser range finder is mounted on the head (see Figure 35). The range finder can measure a distance to the object being pointing to. The location of the object is calculated based on the joint angles between the laser range finder and the body of the robot. There are four ultra sonic sensors on the back of the WorkPartner. These sensors detect object at a distance of two meters. The ultra sonic sensors are used mainly for obstacle detection while robot is backed up.

6.1.6 Control system

Main computer

The main computer contains several software processes. The software architecture is presented in Figure 36. A description of the processes is presented in Table 10.

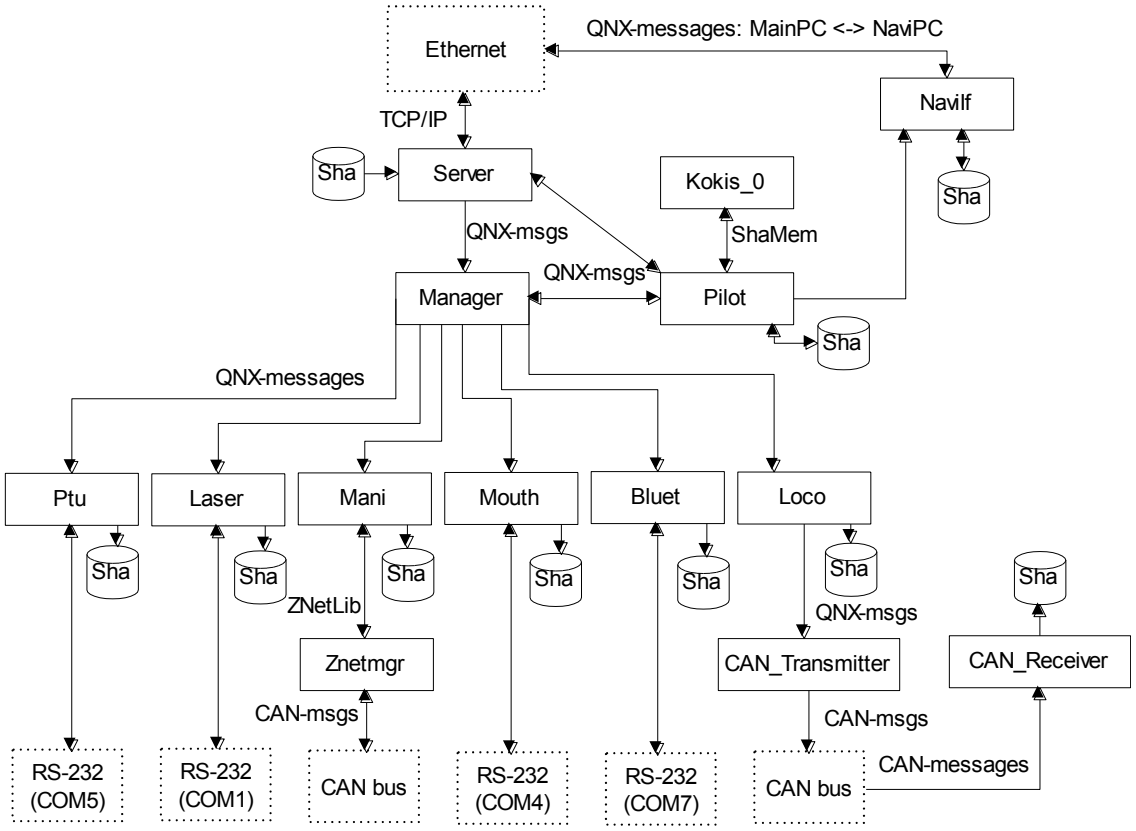

Figure 36. Software architecture in main computer.

Table 10. Processes of the main computer.

Process name (see figure 36)	**Description**
Server	Router process between on-platform Ethernet and wireless LAN. All the messages, for example, from the user interface go through this process. The communication protocol is XML messages using TCP/IP packets.
Manager	Coordination process of all the processes in the main computer. This process launch other processes and all the QNX-messages goes through this process.
Pilot	Pilot handles lower-level controls (see [Kauppi 2003]).
Kokis_0	Command interpreter for the pilot (see [Kauppi 2003]).
Naviif	Communication process between the navigation computer and main computer.
PTU	Control process for pan-tilt unit on manipulator (PTU unit is like the neck of the robot).
Laser	Control process for laser range finder on robot's head.
Mani	Control process for manipulator.
Znetmgr	Lower-level control process and CAN communication handler for manipulator.
Mouth	Control process for LED display.
Bluet	Communication handler for Bluetooth module.
Loco	Locomotion controller. The movements of the whole platform (legs and middle joint) are controlled with this process.
CAN_Transmitter	CAN transmitter for Loco process. Send commands to leg controllers and middle joint controller.
CAN_Receiver	CAN receiver for leg and middle joint controllers. Receives CAN messages (= measurements and status messages) and stores data to shared memory.
Sha	Shared memory is a common database for all the measurements and controls of the robot platform. All the processes have access to this memory area.

Data-handling computer

The data-handling computer contains more upper-level software than the main computer. In the main computer the software is focused more on controlling devices on the platform and in the data-handling computer the focus is nearer to a user interface, for example image handling, the speech system, and stereo hearing.

User interface computer

In the tests the task control architecture is running on a laptop. More details of the architecture can be found in the chapter 4. Figure 37 presents the software architecture used in the user interface computer. The modules from the figure are next presented in more detail.

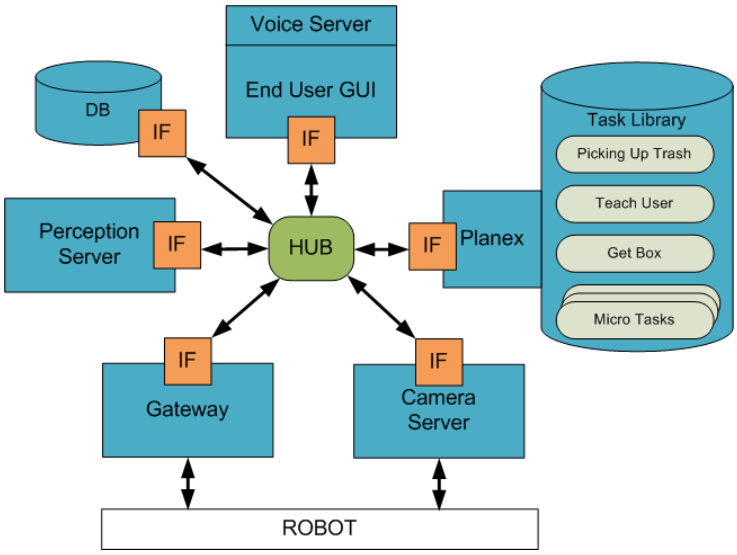

Figure 37. Architecture of User Interface Computer.

Planex

Planex is a combination of a plan configuration tool and a task execution engine. Planex holds a micro task library and offers a graphical user interface for configuring plans. There is a more detailed presentation of Planex in Chapter 4.

Voice server / End user GUI

The end user GUI is a graphical map-based user interface (see Figure 27). The end user GUI is the main module for the end users commanding the robot and the tasks. The main interface is a map-based window. In the window the user can see a map of the environment and the location of the robot. The GUI shows not only the working environment but also the common ground between the robot and the user. For example, the objects recognized by both the robot and the human are displayed on the screen. The voice server is part of the end user interface. It has a speech synthesizer and speech recognition modules. Speech SDK from Microsoft is used for recognizing speech.

Database

All the important data on the objects are stored in a database. The database is designed to represent the surrounding world for the robot. Large amounts of data have to be retrieved at once, for example in map creation, and the information that is required needs to be searched for quickly by the classification of database objects. The database is like a model of the world.

Perception Server

The perception server interprets the camera image data. It can also actively take measurements of its environment by utilizing the DoF of the robot, which can move the sensors. For example, the location of an object on the ground can be measured by pointing a pan-tilt unit-controlled laser distance meter at the object.

Gateway

The gateway is the communication router between the robot and the hub. This communicates with the *Server* process on the main computer (see Figure 36). The gateway has the real control connection to the robot and utilizes the functions realized on board the robot. For example, localization is an independent unit on board, utilizing laser odometry, wheel odometry, and GPS, and the navigation function is realized in the onboard control program. The control commands are based on ILMR [Kauppi 2003].

Camera server

The camera server loads images from the video server (see Figure 34) and tries to handle the bottleneck of wireless communication and bandwidth-hogging image transmission between the robot and the rest of the computing resources. It receives the video stream/images and broadcasts them to the wired network.

6.2 Plan of Experiments

The platform was tested with four test cases. The focus of each test was selected to verify functionalities related to the main subjects of the thesis.

Table 11 shows the test cases from the point of view of the task configurator. The focuses of the selected experiment are marked with an X. The task configurator has the main target of getting new tasks to be workable. The task configurator sketches a plan with the plan configuration tool, tests plans, and makes modifications if needed. Therefore the task design process is focused on in all the test cases. Interpretive task execution is also under test because of the debugging process of the plans. External signs are tested with a different software platform and so the focus of the test also varies.

Validation of experiments is done by comparing results to the traditional way of doing. Traditional way is programming the plan using common programming language (for example, C++)

instead of using the plan configuration methods presented in thesis. The results of test cases one to three are compared to the test cases testing external signs. PlanLang and the task control platform were not ready at that time when the external signs were tested. For this reason, the plan of test case four is configured by hard coding with C++. The focus of the test cases is to test the task configuration process, verifying to see whether different kinds of tasks are possible.

Table 11. Test cases when the user is a task configurator. The focus of the tests is marked with an "X".

Experiment	**Task Configuration Process**	**Interpretive Task Execution**	**Human-Robot Interaction**
Teaching the robot to identify the user	X	X	
Adaptation during task	X	X	
Interactive task initialization	X	X	X
External signs as a part of task description	X		X

The end user works directly with the robot using a multimodal user interface, being concerned with no longer changing the structure of the plans. In all the test cases the main focus is on the human-robot interaction when the user is an end user. A more detailed description and the results of each test case are presented in the next chapter. The experiments presented were also demonstrated to a live audience, including people on the management committee of the research project. Videos of the demonstrations can be seen on the webpage of the WorkPartner robot http://automation.tkk.fi/WorkPartner [TKK Automation Technology Laboratory].

6.3 Experiments

6.3.1 Teaching the robot to identify users

Knowledge of who the user or master is important for the robot. It should know who its commander is. This situation may cause some error condition or even dangerous situations if the robot takes a command from the wrong person. The purpose of this test is to enable the robot to recognize the user and also test the usability of the recognition process.

Teaching phase

In the tests there were two users (*user1* and *user2*) wearing jackets of a different uniform color from each other (see Figure 38). This method is selected to simplify the test, not to focus on any high-level face or shape recognition algorithm but to test the identification process only.

Figure 38. Two users, user1 (dressed as Santa Claus) and user2 (dressed in violet overalls), introduce themselves to the robot.

User1 was carrying a laptop in this phase. Both users were standing in the front of the camera, while *User1* clicked an image (on End User GUI) of the users with a mouse. These image coordinates were passed to the work task, which first analyzed the image at a specified point, and saved the appearance of that point and its vicinity to the database. A unifying name was also attached to the object, so *user1* told the system that, for example, the violet user was called *user2* and he (*user1*) was wearing a red jacket. After this phase *user1* did not have to carry the laptop computer any more.

Teaching the user was a plan configured with PlanGUI. Figure 39 presents the configured plan on a screenshot of PlanGUI. The plan contains five micro tasks: *PS_AnalyzeColor*, *PS_LocateObject*, *DB_AddUser*, *ILMR_Speak*, and *PTU_Turn*. A detailed description of the micro tasks is presented in Table 12.

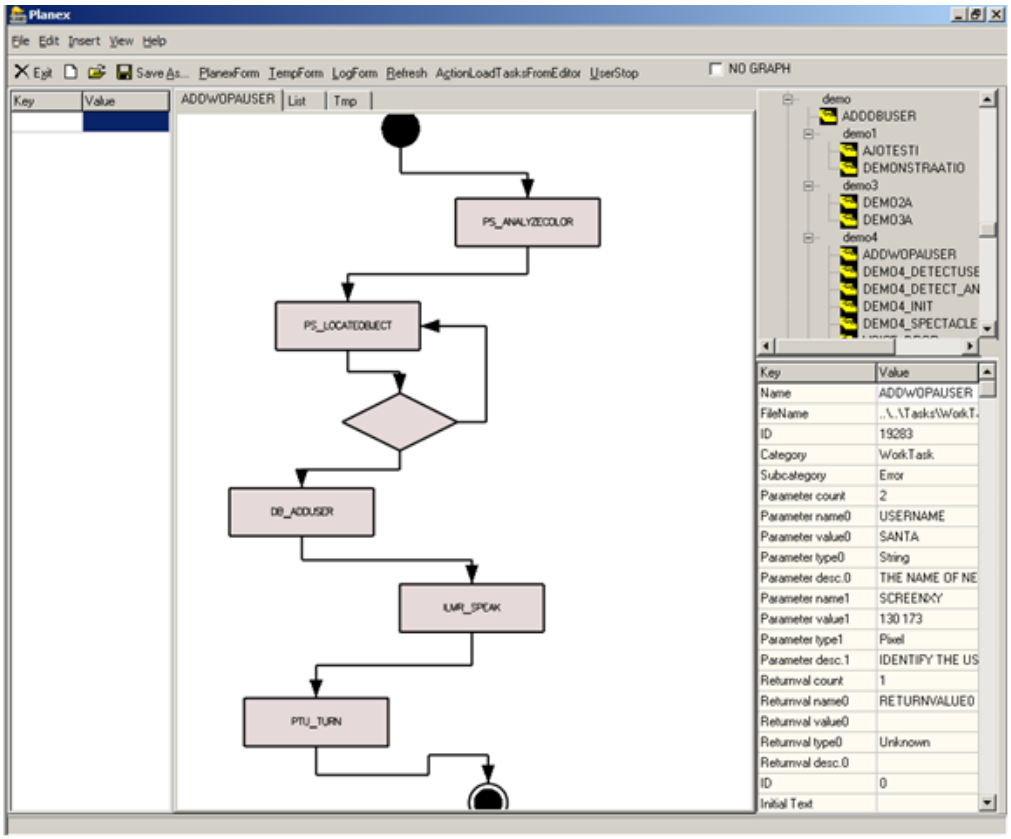

Figure 39. Plan of teaching the user depicted on a screenshot of PlanGUI.

Table 12. Description of micro tasks in the "teach the user" plan.

Micro task	**Description**
PS_AnalyzeColor	Analyzes the color (selected by the user) on the image screen. Return hue, red, green and blue values of the selected color.
PS_LocateObject	Turn the head of the robot towards the user clicked on the screen. Measures the position of the user in relation to the robot using a laser range finder. The plan continuously attempts to locate the user until found.
DB_AddUser	Asks for the name for the user and stores name and color information in the database.
ILMR_Speak	Says “User successfully added”.
PTU_Turn	Turns the head of the robot to initial position.

Because the color was used as a recognition method, it was noticed that there are several factors that weaken accuracy. Typical factors are indoor lighting, sunlight, and a color selected that is similar to the background, such as the walls. To improve user recognition accuracy, a color selection test was arranged. In the test several objects of different colors were selected. All the objects were of about human size and they were placed so as to face the robot. During the test process some colors were found to be better than others when used as a user detection factor. This suitability is presented in Table 13.

Table 13. Suitable colors for user detection.

Color	**Suitability as a recognition factor for the user**
Grey	Bad
Blue	Moderate
Red	Excellent
Violet	Excellent
Yellow	Good

Using color as a recognition factor also has some drawbacks. If there is an object with the same color as the user in the environment, the results will be rejected. In the future more sophisticated methods will be selected as a recognition factor. Some examples are face and voice recognition.

Recognition phase

Recognition of the user was tested by commanding the robot to find the user several times. The plan presented in Figure 40 was used in the test. In the recognition phase, two users, *user1* and *user2*, shouted "Hello, Wopa" (Wopa is the nickname of WorkPartner) at WorkPartner several times from different locations around the robot. The robot turned its head towards the user and said the name of the user and started to follow the user by turning its head. Because there was more than one user, the robot needed to combine the color of the user with the name of the user.

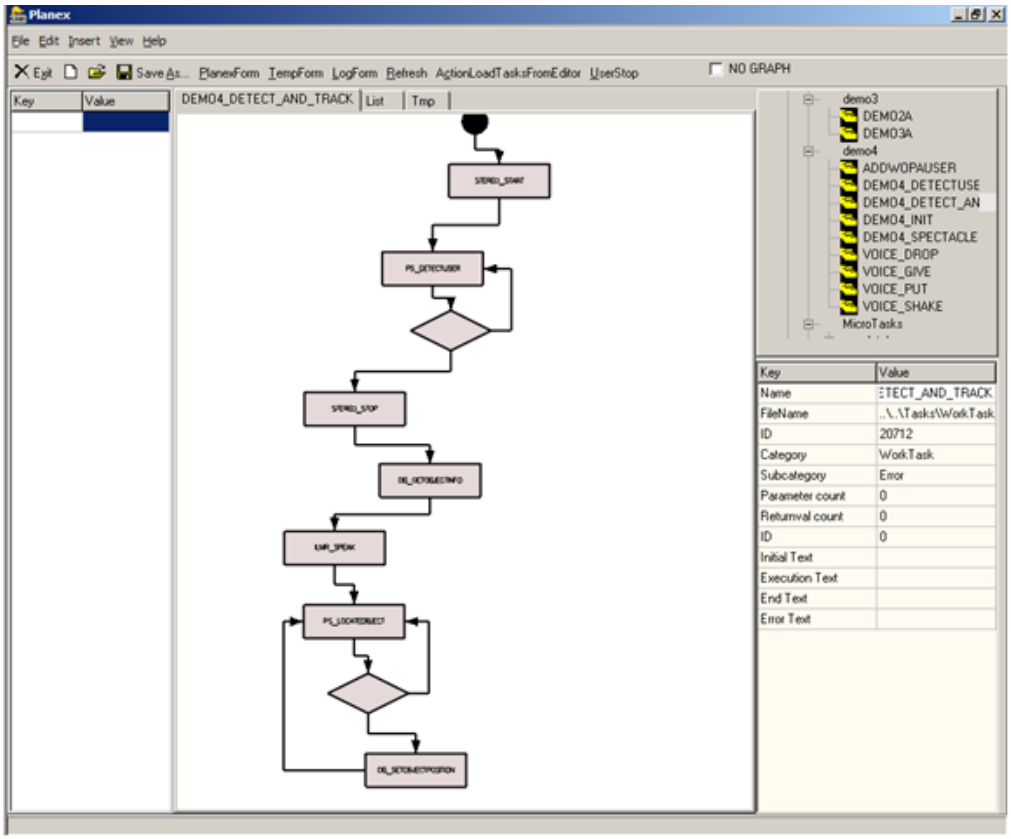

Figure 40. Screenshot of PlanGUI. Recognition of the user plan is presented.

The micro tasks applied to the plan in this test case were: *Stereo_Start*, *PS_DetectUser*, *Stereo_Stop*, *DB_GetObjectInfo*, *ILMR_Speak*, *PS_LocateObject*, and *DB_SetObjectPosition*. A detailed description of these micro tasks can be seen in Table 14.

Table 14. Description of micro tasks in "recognize and track the user" plan.

Micro Task	**Description**
Stereo_Start	Start stereo hearing. Stereo hearing system turns the head of the robot towards the sound it receives.
PS_DetectUser	Compare the user information database (saved in the teaching phase) to the view on its camera. If a successful match of the color information was found, the plan continued.
Stereo_Stop	Stop the stereo hearing.
DB_GetObjectInfo	Ask for the name of the user from the database on the basis of the color information found.
ILMR_Speak	Say “Hello, user1”, for example.
PS_LocateObject	Locate the user and turn the head of the robot towards the user while he is moving.
DB_SetObjectPosition	Add the updated position of the user to the database.

Results

The traditional way to configure this task is programming all the elements inside one application with a common programming language. Instead of using PlanLang based visual plan

configuration it needs much more time. Also, the flexibility of the plan is better than in the traditional way enabling the user to make changes to the plan easily.

6.3.2 Testing adaptation during task

In this test case the adaption during test execution was tested, the application being picking up litter. The gripper of WorkPartner was changed to a spike, shown in Figure 41. The spike is more suitable for picking up litter than a gripper. The test case could be imagined as a continuous task in which the robot should pick up pieces of litter one by one. The plan of the task was one set of micro tasks looped again and again. The configured plan is presented in Figure 44. The colored areas are drawn on the visual realization of the plan, presenting the phases of the plan in order to simplify presentation. The phases of the plan are *initialization*, *detect litter*, *locate litter*, *approach*, *locate gripper*, *pick up*, and *back to home* and also *preparation*. These phases are next described in more detail.

Figure 41. WorkPartner is picking up litter.

Initialization

Initialization contains micro tasks for setting the robot to be ready to pick up litter. This means setting the joints (manipulator and head) to their initial set points and setting up the body ready for moving.

Detect litter

Detect litter is a continuous loop in which the robot, using a camera, seeks the litter on the ground. The litter is a red foam plastic cube and the color is used as a reorganization factor. When this red cube is found the plan engine moves to the next phase.

Locate litter

Locate litter is the phase in which the position of the litter is detected. The laser range finder and camera are the sensors used in this detection. The head is turned so that the litter is seen in the middle of the camera image. The main algorithm for image handling is hue-based color filtering (aka red filter in this case). After that the distance from the head is measured with the laser range finder. The position is calculated from the measured distance and the joint angles of the head. If this phase is accepted, the routine will continue; otherwise, the plan will be restarted.

Figure 42. WorkPartner measuring the exact position of the litter. The original image and image thresholded with red.

Approach

The robot's body is driven almost to the litter in this phase. The hands are also turned so that picking it up will be possible.

Locate gripper

Measuring the exact position of the hand of WorkPartner is not possible using the internal measurements of the joints. In addition to that, visual control of the hand is used. The end of the litter spike is measured using the camera and laser range finder (using the same methods as in the *locate litter* phase). After this the hand can be moved exactly to the litter.

Figure 43. WorkPartner measuring the exact position of the gripper (or spike). The original image and image thresholded with green.

Pick up

Picking up the litter is a series of movements of the manipulator. First, the spike is moved over the litter. Then the spike is stuck into the litter and, finally, the litter is lifted and dropped into the basket on the robot.

Back to home

Back to home is the phase in which the manipulator is set to its initial position and the robot is moved slightly backwards. After this the plan is restarted to find more litter.

Preparation

Preparation is the phase which includes micro tasks for setting up the robot's state so as to be ready for the next phase.

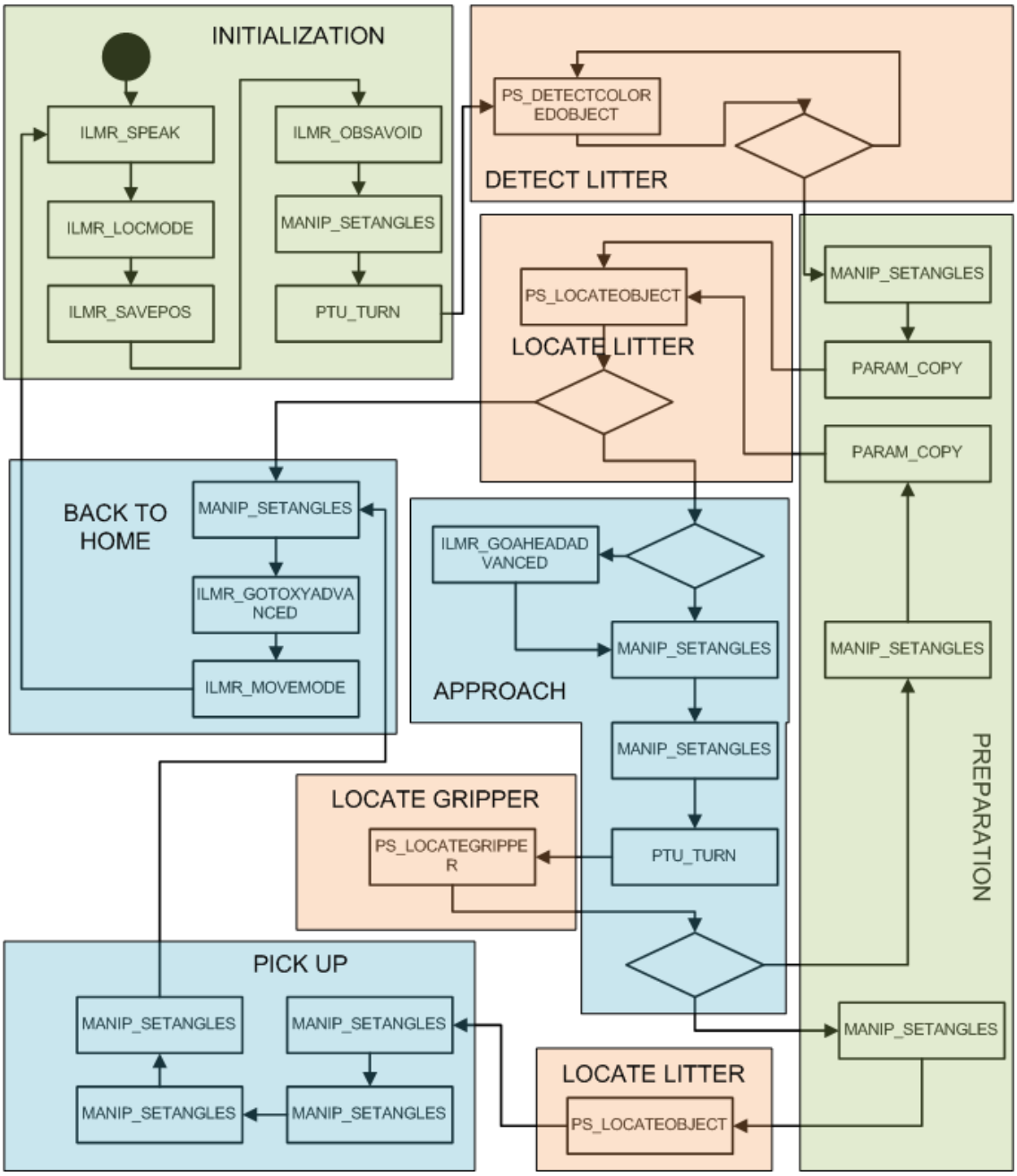

Figure 44. Edited screen shot of PlanGUI presenting Pick up litter task.

Results

Figure 45 presents the test setup where WorkPartner is starting to pick up litter items (red cubes). Figure 46 shows the trajectory is based on the robot's own navigation system so it is not absolute trajectory. Configuration of this plan in a traditional way is challenging because of the use of several sensors in parallel. Instead of the traditional way, the configuration and execution of this plan was easy to implement. The phases one to four labeled in the Figure 46 are the following:

1. Starting point. Initialization, detecting and locating the litter

2. Approaching to location of first litter item. Locating gripper and also litter more precise. Picking up the litter.
3. Driving back to starting point. Detect and locate second litter item.
4. Approach to location of second litter item and pick it up as presented in phase two. Finally drive back to the beginning point.

Figure 45. WorkPartner starts to pick up two pieces of litter. The litter items are red plastic foam cubes in front of the robot.

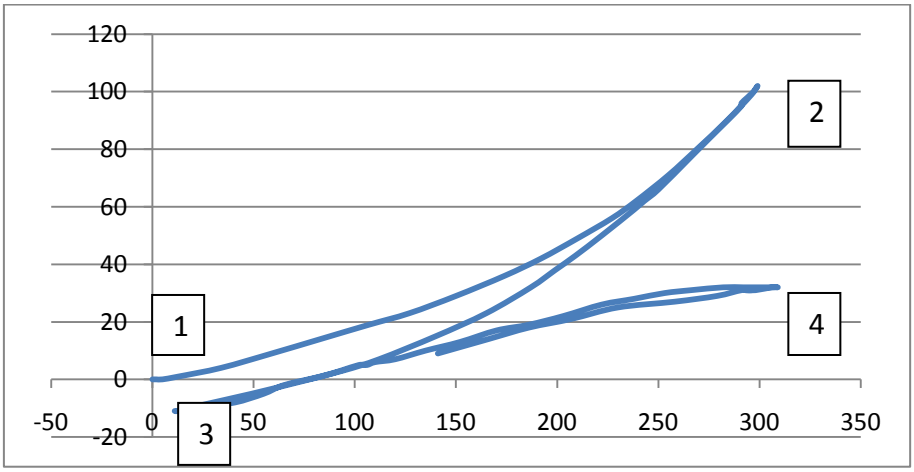

Figure 46. Trajectory of picking up two litters. Unit of the axes is centimeter.

6.3.3 Testing human-robot interaction-based task initialization

The interaction between the operator and the robot was the key point of this phase. The phase started when User1 shouted to the robot “Hello, Wopa”. The robot switched its stereo hearing on and turned its head towards the direction of the sound. The verbal command “Get box” was given to the robot. The robot interacted by asking for more details. For example: “What

kind of box?", "Where is the box?" When all the necessary information had been received, the robot started to search for the location of the user. The color that had been learned was used to identify and locate the user. The robot drove to the user and inquired about the box. The robot announced if the box with the specific color had not been found. When the correct box was found, the robot gripped it between its left and right hands and returned to the starting point. The robot knew that it needed to grab the box with two hands, because its type was 'box'. Table 15 shows the main steps of the task of getting the box.

Locating the user took place in three phases. The stereo hearing caused the robot to turn its head towards the loudest sound that could be heard. The robot was searching for possible users from the image. When one was found, it was identified and located. Then the "user" object was added to the database. Figure 47 (a) shows how these phases were realized in the task. Interaction with the user was realized with simple blocks containing speech commands and voice recognition commands.

In the next phase, the robot needed to locate the next person, in this case user2. First, the robot sought the name from the database. The database contained a description of the appearance of the corresponding person, in this case, the color of the jacket. Then the robot turned its head in the direction where it had last seen the person, and started to search for the person. If the user was not found, the entire surroundings were scanned. Figure 47 (b) shows the phases of locating the other person.

Table 15. Dialog between the user and the robot and the main steps of the 'getting the box' task.

	Robot	**Main User**
1	Stereo Hearing ON	
2		"Hello, Wopa"
3	Locate User	
4	Stereo Hearing OFF	
5	"What Can I Do for You"	
6		"Get Box"
7	"What Kind of Box"	
8		"It is red"
9	"Where Is the Box"	
10		"With user2"
11	Locate User (user2)	
12	SavePos (Home)	
13	GotoXY (user2)	
14	Locate Object (Red Box)	
15	Grab (Box)	
16	GotoXY (Home)	
17	"What Can I Do for You"	

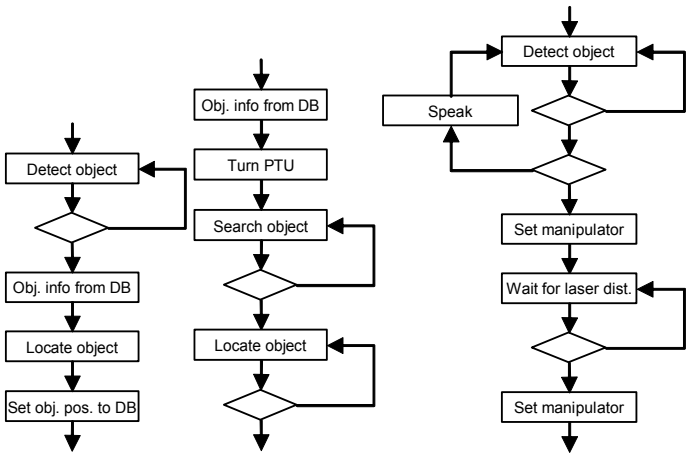

Figure 47. Three excerpts from the "Get box" task. (a) Detecting, identifying, and locating the user. (b) Identifying and locating the other person. (c) Waiting for a box, determining if it is the correct box, waiting for the box to be placed at the right distance, and grabbing the box.

Moving near the box took place in a similar way to the movement parts of the previous example. In the next phase the robot grabbed the box. First, it waited until the person showed a box. If a box with the wrong color was detected, the robot asked for the right box and continued waiting. Then it used the laser pointer to determine that the box was correctly put between its hands. When it was in the right position, the robot grabbed the box. Figure 47 (c) shows the phases of grabbing the box.

Finally, the robot drove back to the user, and asked what it should do with the box. When it returned back "home" with the box the robot waited for more commands from the operator. Some possible commands were "Shake the box", "Put the box down", "Give the box to User2", etc.

Results

The configuration of complex task interactively with the robot was clear instead of programming the whole task manually. Humans use objects and places naturally in commands. So instead of saying "goto position (2,4)" it is more natural to say "go to user2". This test case showed the benefits of human-friendly communication compared to the traditional way.

6.3.4 Testing external signs as a part of task description

Two work tasks were used to test the concept. In the first task the robot plowed an area of snow, and in the second task the robot brushed the ground. In both tasks signs were used to indicate the working area of the robot. Both active and passive signs are used in the research. The software system used in these tests is a draft version of the one presented in Chapter 4.

Case study: Passive signs

The first case was the robot plowing snow (see Figure 48). Four passive signs were used to define the working area. The signs showed the direction to the next sign, from one corner of the working area to another. Figure 49 presents the signs used in this test case. There are two spheres in the passive sign, a larger orange one and a smaller yellow one.

Figure 48. WorkPartner plowing snow. The white tubes behind the robot were used as beacons for the navigation system.

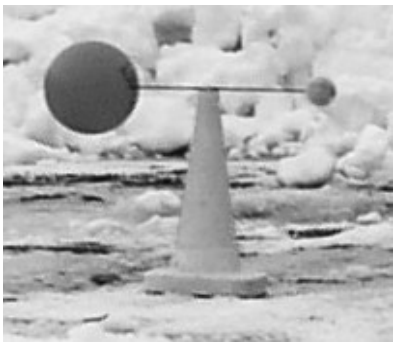

Figure 49. A passive sign placed in one corner of the working area. The sign is pointing in the direction of the next corner point. The position vector pointing from the center of the big ball to the center of the small ball defines the direction.

The plan was configured in such a way that the working area was defined using signs. In the initial test case WorkPartner asked for the estimated location of the working area. The user gave the location by clicking on a screen map. In the next phase, the robot asked how it could move to the working area. The user clicked the points of the route on the map. Finally, the robot asked when it could start up and the user accepted it.

During the execution the robot followed the path given by the user. At the end of this route the robot sought the signs one by one and calculated the working area. The calculation was based on the location of the signs. Next, the robot wandered along a route inside the working area and moved the tool, the plow, up and down. More specifically, the robot wandered to one end of the working area, lowered the tool, and started to clean the area. After reaching the other end of the area, the robot raised the tool again and moved back to the other end. After a few rounds the whole area was cleared.

The passive signs were found by using computer vision. The spheres appeared as circles in the image. The image was segmented by color. The orange and yellow circles were found from the threshold image using a circle-fitting algorithm (Figure 50). The distance to the sign was determined by measuring the size of the circle on the image plane. The 3D pose of the sign was calculated on the basis of the measured distance and the camera orientation [Suontama 2003].

Figure 50. The original image, image thresholded with orange, and the orange circle that was found.

Results

The resulting trajectory of one test round is shown in Figure 51. The trajectory is based on WorkPartner's own navigation system, so it is not an absolute trajectory. The trajectory shows the starting point, the route to the working area, and the reciprocation motion over the working area. The algorithm for calculating the route and the backwards and forwards motion is similar to that presented in [Oksanen 2007], but simpler.

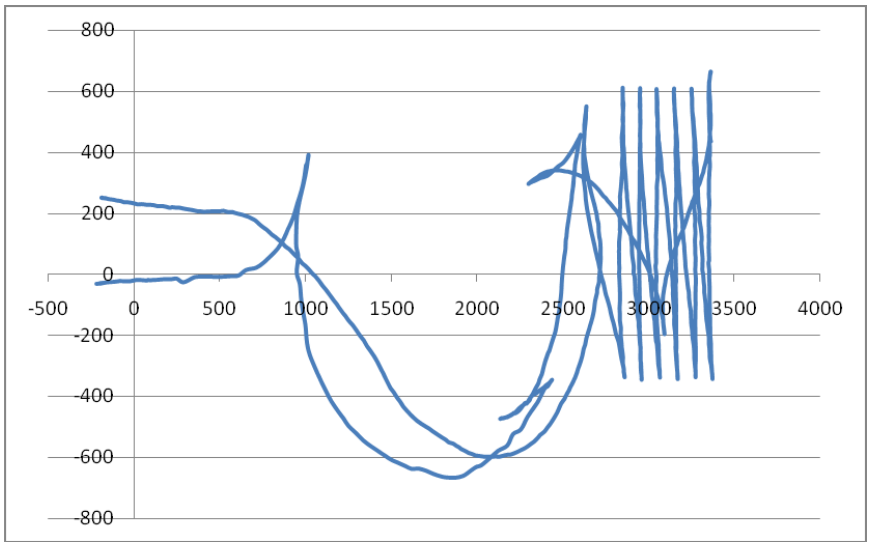

Figure 51. Trajectory of snow plowing test. The unit of the axes is centimeters.

During the execution of the task the accuracy of the image-handling method was not measured. The accuracy was measured before using signs in the work tasks [Suontama 2003]. On the basis of the results it was easier to develop the control system for the work tasks. To find out the accuracy of the image-handling method, the distances between the spheres were also measured with a laser pointer. The laser pointer is the second eye of WorkPartner and the accuracy of the pointer is about 1-2 mm. In the experiments the distances measured with the vision system and the laser pointer were compared.

The distance from WorkPartner's head (both from the camera and from the laser pointer) to the sign was transformed to a location relative to the base coordinate frame of the robot (Figure 52) by calculating the necessary kinematics transforms. Figure 53 shows the errors of the locations. The errors are the same as the divergences between the calculated locations of the sign measured by the laser pointer and the camera. In the experiments the sign was pointing about 90 degrees to the right at the front of the robot [Suontama 2003].

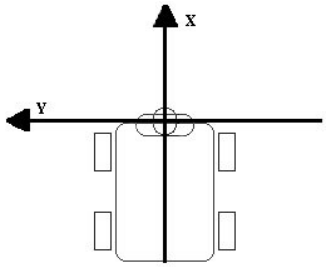

Figure 52. The base coordinate system of the robot.

The results shown in Figure 53 demonstrate that it is possible to localize the passive signs properly when the measuring distance is from 2 to 4 meters. There was more error in the X-coordinate because the measured distance from the camera is more related to the image handling than the angle of the pan-tilt unit of the robot's head. The heading of the sign was calculated on the basis of the position of its spheres. The divergence between the correct angle of the sign and the one that was calculated was from -10 to 10 degrees. The error was huge but small enough in this case.

In the experiment the accuracy of the measurements was good enough because it was not necessary to know the positions and headings of the signs exactly. The idea was to say to the robot "Go in that direction" or "I am somewhere here". This includes the same kind of information as between humans. The image handling caused most errors when the locations of the signs were being measured. Changes in lighting conditions were the most effective aspect of image handling. The task planner should take into account the limitations of the accuracy when planning the movements and routes for the robot.

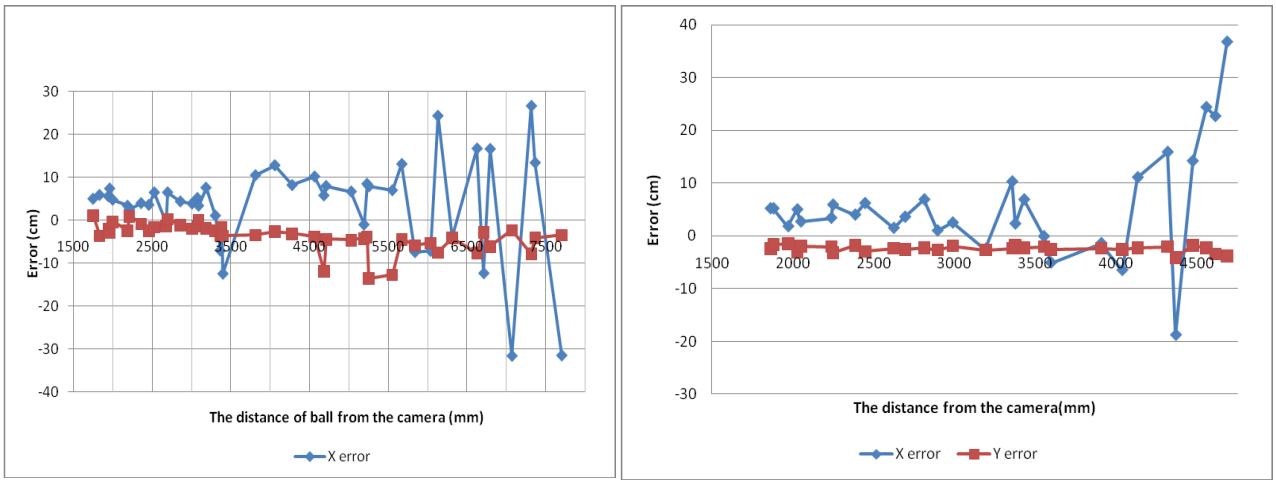

Figure 53. Accuracy of the localization (left: orange ball, diameter 22 cm, right: yellow ball, diameter 6.6 cm).

Case study: Active signs

An active sign was used in the brushing task (in Figure 54). The active sign (in Figure 55) includes a microcontroller, Bluetooth, and GPS modules. The sign was located in the center of the working area. Before starting to clean the area, WorkPartner communicated with the sign via Bluetooth. The data received from the sign included the location of the sign and the radius of the working area.

Figure 54. WorkPartner is brushing the ground.

The robot planned the brushing task on the basis of the information received in the same way as in the case with the passive signs. The only difference in the plan is how the working area is defined. There were four passive signs located in the corners of the working area. With the active sign it was located in the middle of the working area.

Figure 55. Active signs containing a GPS receiver, Bluetooth module, and a micro controller.

6.4 Conclusions

The platform was tested with four test cases. The focus of each test was selected to verify functionalities related to the main subjects of the thesis. Instead of programming the task using traditional languages (such as C++) tasks are configured using methods presented in the thesis. Based on the author's experience all the test cases were easy to configure and execute using the presented methods. It would be challenging to program the same functionalities in more traditional ways. Also, the usability of configuration would be much lower.

7 Conclusions

In this research, a novel task configuration concept for multipurpose service robots has been presented. The proposed Simultaneous Instruction Writing and Execution (SIWE) method for configuring new tasks for service robots has its roots in the time-honored master-apprentice approach. The concept presents a new method which makes it easier to configure a new task for a robot. The idea is the same as when a person tells another how a task should be performed. A novel method for executing tasks with service robots is also presented. Interpretive execution, keeping the focus on only one micro task at a time, makes it possible to modify plans during their execution. This feature makes it possible to create an automated optimization system for task configuration in the future. Multimodal interaction is an important feature for providing collaboration between humans and robots. Multimodality reduces the workload of the user by administering the task configuration and execution. Furthermore, a novel solution for using multimodal human-robot interaction (HRI) as a part of the task description is presented.

The developed software platform presents an efficient way to control service robots for accomplishing real world tasks. The platform provides an easy way to implement complex work tasks for various service robots. It defines the final plan of the work task and the building blocks used to create the needed work task. The plan is divided into subtasks called micro tasks. Those are independent modules always to be executed as a whole.

The platform offers a specification of the services and communication channels between the user and the implemented modules. All the software modules in the systems are independent and communicate with each other via a common interface. The physical location of the modules can be anywhere in the network. The common communication interface is TCP/IP-based, providing communication over the Internet, as well as in the local system. Decentralization offers, for example, external calculation power for small controllers and fluent communication in multi-robot systems.

The configuration of the plan is stored by using a so-called task composition language. This language is a representation of a plan providing a hierarchical structure of the actions of the desired work task. It also provides the elements for representing the actions needed, their internal

parameters, and the order of execution of the actions, including handling exceptions. The proposed task composition language is named as PlanLang. Instead of writing the XML based PlanLang directly, the configuration of plans is done by using visual programming. The activity diagram was selected to be the base format for the plans. Visual configuration of the plans is much faster method than textual writing. PlanLang offers the configuration of plans into a general form allowing the sharing of configured plans over a network. In the future the owners or users of robots could have an access to an internet-based service offering new plans for various tasks. Instead of configuring the plan by themselves, the users look whether the service already offers a suitable task.

The plan engine takes care of interpreting the plan description and controlling the execution flow. The thread-based approach means that every micro task is executed inside one of the independent threads. This provides more robust execution and parallelism for the micro tasks.

The interaction between the robot and the user can be achieved in various ways. Usually, to operate a robot the user needs some kind of portable computer or something similar. He/she will use this device to see, for example, the image from the robot's camera, the map, or the state of execution of the task at hand. The user can, for example, point to objects to be manipulated from the image or from the map. The interaction can also take place in the real world. The user can use gestures or show the objects to be manipulated with a pointing device, such as a distinctly colored stick. In addition, interesting objects can be marked with signs. For example, a bar code can be used to identify a box that should be lifted. Signs can also be used to determine the area where the robot is supposed to operate.

The platform was implemented in the WorkPartner service robot and its usability and performance were studied. The structure and operation of the platform proved to be very useful. Instead of programming tasks by using traditional languages (such as C++) the tasks are configured using the methods presented in this thesis. The proposed methods were found to be very efficient and an easy way to configure tasks. Multimodal interaction between a human and the robot was found to be a straightforward and human-friendly communication method.

In the future, the platform will be implemented as a part of the GIM (the Finnish Academy's Centre of Excellence – *Generic Intelligent Machines*) architecture [Saarinen et al. 2007]. Sensing of the environment will be improved to a level where the robot will recognize both objects and humans smoothly. This will enable future work tasks to be even more complex and the performance to be even more reliable. The developed platform will become a part of the vision where humans and robots are working efficiently together at a common worksite both performing operations that they do best: robots working tirelessly improving their performances as the time passes and humans supervising and instructing them if a need arises. The analogy to the old master-apprentice relationship, so common in many professions requiring craftsmanship, is strong and well-deserved.

8 References

Aarnio, P., Koskinen, K. & Ylönen, S. 2001. *Using Simulation during Development of Combined Manipulator and Hybrid Locomotion Platform.* Proceedings of International Conference on Field and Service Robotics. pp. 287-294.

Alami, R., Chatila, R., Fleury, S., Ghallab, M. & Ingr, F. 1998. *An Architecture for Autonomy.* International Journal of Robotics Research. Vol. 17. pp. 315-337.

Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T. & Yoon, W. 2005. *RT-Component Object Model in RT-Middleware—Distributed Component Middleware for RT (Robot Technology).* Computational Intelligence in Robotics and Automation (CIRA 2005) Proceedings. 2005 IEEE International Symposium on. pp. 457-462.

Andronache, V. & Scheutz, M. 2006. *Ade - an Architecture Development Environment for Virtual and Robotic Agents.* International Journal on Artificial Intelligence Tools. Vol. 15:2. pp. 251-286.

Arai, T., Itoko, T. & Yago, H. 1997. *A graphical robot language developed in Japan.* Robotica. Vol. 15:1. pp. 99-103. 0263-5747.

Arkin, R.C., Riseman, E.M. & Hanson, A.R. 1987. *AuRA: An Architecture for Vision-Based Robot Navigation.* Proceedings of the DARPA Image Understanding Workshop.

Asama, H. 2001. *Towards emergence in distributed autonomous robotic systems--intelligent data carrier for cooperative organization of mobile robots and their operating environment.* 3rd International Conference in Field and Service Robotics. Espoo, Finland.

Breazeal, C. 2003a. *Emotive qualities in lip-synchronized robot speech.* Advanced Robotics. Vol. 17:2. pp. 97-113(17). ISSN 0169-1864.

Breazeal, C. 2003b. *Toward sociable robots.* Robotics and Autonomous Systems. Vol. 42. pp. 167-175(9).

Bruyninckx, H. 2001. *Open Robot Control Software: the OROCOS project.* IEEE Int. Conf. Robotics and Automation. pp. 2523-2528.

Calisi, D., Censi, A., Iocchi, L. & Nardi, D. 2008. *OpenRDK: A modular framework for robotic software development.* Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. pp. 1872-1877.

Chaimowicz, L., Cowley, A., Sabella, V. & Taylor, C.J. 2003. *ROCI: A Distributed Framework for Multi-Robot Perception and Control.* In Proceedings of the 2003 IEEE/RJS International Conference on Intelligent Robots and Systems, Las Vegas, NV. pp. 266-271.

Collett, T.H.J., MacDonald, B.A. & Gerkey, B.P. 2005. *Player 2.0: Toward a Practical Robot Programming Framework. Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*. Sydney, Australia. December.

Cote, C., Letourneau, D., Michaud, F., Valin, J.-., Brosseau, Y., Raievsky, C., Lemay, M. & Tran, V. 2004. *Code reusability tools for programming mobile robots.* Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Vol. 2:pp. 1820-1825 vol.2.

Côté, C., Brosseau, Y., Létourneau, D., Raïevsky, C. & Michaud, F. 2006. *Robotic Software Integration Using MARIE.* International Journal of Advanced Robotic Systems. Vol. 3:1. pp. 055-060.

Douglass, B.P. 2000. *Real-Time UML, Second Edition: Developing Efficient Objects for Embedded Systems.* Addison-Wesley. 328 p. ISBN 0-201-65784-8.

Enderle, S., Utz, H., Sablatnög, S., Simon, S., Kraetzschmar, G. & Palm, G. 2001. *Miro: Middleware for Autonomous Mobile Robots.* In Telematics Applications in Automation and Robotics.

Estlin, T., Volpe, R., Nesnas, I., Mutz, D., Fisher, F., Engelhardt, B. & Chien, S. 2001. *Decision-Making in a Robotic Architecture for Autonomy. Proceedings of 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*. Montreal Canada. June 18-21.

Ferketic, J., Goldblatt, L., Hodgson, E., Murray, S., Wichowski, R., Bradley, A., Fong, T. & Steinfeld, A. 2006. *Toward Human-Robot Interface Standards I: Use of Standardization and Intelligent Subsystems for Advancing Human-Robotic Competency in Space Exploration.* SAE 36th International Conference on Environmental Systems. July.

Fleury, S., Herrb, M. & Chatila, R. 1994. *Design of a modular architecture for autonomous robot.* Robotics and Automation, 1994. Proceedings. , 1994 IEEE International Conference on. pp. 3508-3513 vol.4.

Fong, T., Kunz, C., Hiatt, L.M. & Bugajska, M. 2006. *The human-robot interaction operating system.* HRI '06: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction. Salt Lake City, Utah, USA. March 2-3. New York, NY, USA: ACM. pp. 41-48. ISBN: 1-59593-294-1.

Fong, T., Nourbakhsh, I. & Dautenhahn, K. 2003. *A survey of socially interactive robots.* Robotics and Autonomous Systems. Vol. 42:3. pp. 143-166. ISSN 0921-8890.

Fong, T., Scholtz, J., Shah, J., Fluckiger, L., Kunz, C., Lees, D., Schreiner, J., Siegel, M., Hiatt, L.M., Nourbakhsh, I., Simmons, R., Ambrose, R., Burridge, R., Antonishek, B., Bugajska, M., Schultz, A.C. & Trafton, J.G. 2006. *A Preliminary Study of Peer-to-Peer Human-Robot Interaction.* IEEE International Conference on Systems, Man, and Cybernetics. Taipei, Taiwan. October 8-11.

Gates, B. 2006. *A Robot in Every Home.* Scientific American.

Georgia Institute of Technology. *Power of Touch: Researchers Engineer Old-fashioned "Touch" into Modern Electronic Controls.* [Homepage of Research Horizons Magazine. Writer: Gary Goettling. September 24 2004.]. [Online]. [Cited 2008, 6/3/2008]. Available at: http://www.gtresearchnews.gatech.edu/newsrelease/haptics.htm.

Gerkey, B.P., Vaughan, R.T., Sukhatme, G.S., Stoy, K., Howard, A. & Mataric, M.J. 2001. *Most Valuable Player: A Robot Device Server for Distributed Control.*

Gertz, M.W., Stewart, D.B. & Khosla, P.K. 1993. *A Software Architecture-Based Human-Machine Interface for Reconfigurable Sensor-Based Control Systems.* Proceedings of the 8th IEEE Symposium on Intelligent Control. pp. 75-80.

Goodrich, M.A. & Schultz, A.C. 2007. *Human-robot interaction: a survey.* Foundations and Trends in Human-Computer Interaction. Vol. 1:3. pp. 203-275. ISSN 1551-3963.

Gowdy, J. 1994, *SAUSAGES: Between Planning and Action*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Green, T.R.G. & Petre, M. 1992. *When visual programs are harder to read than textual programs.* Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics). Rome. CUD. pp. 167-180.

Halme, A., Leppänen, I., Suomela, J., Ylönen, S. & Kettunen, I. 2003. *WorkPartner: Interactive Human-like Service Robot for Outdoor Applications.* The International Journal of Robotics Research. Vol. 22:7/8. pp. 627-640.

Harwin, W.S., Gosine, R.G., Kazi, Z., Lees, D.S. & Dallaway, J.L. 1997. *A comparison of rehabilitation robotics languages and software.* Robotica. Vol. 15:2. pp. 133-151. 0263-5747.

Heikkilä, M., Terho, S., Hirsi, M., Halme, A. & Forsman, P. 2006. *Using Signs for Configuring Work Tasks of Service Robots.* Industrial Robot: An International Journal. Vol. 33:4. pp. 308-311. ISSN: 0143-991X.

Henning, M. 2004. *A New Approach to Object-Oriented Middleware.* IEEE Internet Computing. Vol. 8:1. pp. 66-75. 1089-7801.

IEEE. *Joseph F. Engelberger.* [Cited 2009, 8/18/2009]. Available at: http://www.ieee.org/portal/pages/about/awards/bios/2004roboticsauto.html.

IFR Statistical Department. *Worldrobotics 2007.* [Cited 2009, 8/15/2009]. Available at: http://www.worldrobotics.org/.

Immersion. *What is haptics?* [Cited 2008, 6/3/2008]. Available at: http://www.immersion.com/corporate/press_room/what_is_haptics.php.

Jadex. *Jadex BDI Agent System - Overview (About.Overview).* [Cited 2009, 8/15/2009]. Available at: http://jadex.informatik.uni-hamburg.de/bin/view/About/Overview.

JAUS group. *The Joint Architecture for Unmanned Systems.* [Cited 2008, 9/12/2008]. Available at: http://www.jauswg.org/.

Johnson, M.J., Intlekofer, K.J., Jung, H., Bradshaw, J.M., Allen, J., Suri, N. & Carvalho, M. 2008. *Coordinated Operations in Mixed Teams of Humans and Robots.* Proceedings of the 2008 IEEE International Conference on Distributed Human-Machine Systems. Athens, Greece. March 9-12.

Kaski, T. 2004. *Learning and performing skill demanding tasks with WorkPartner robot.* MSc. Automation Technology Laboratory, Helsinki University of Technology. Espoo.

Kauppi, I. 2003. *Intermediate Language for Mobile Robots: A Link between the High-Level Planner and Low-Level Services in Robots.* PhD. Department of Automation and Systems Engineering, Helsinki University of Technology. ISSN 1235-0621.

Kortenkamp, D. & Simmons, R. 2008. *Robotic Systems Architectures and Programming.* Springer Handbook of Robotics. Springer Berlin Heidelberg. pp. 187-206. ISBN 978-3-540-23957-4 (Print).

Kramer, J. & Scheutz, M. 2007. *Development environments for autonomous mobile robots: A survey.* Autonomous Robots. Vol. 22:2. pp. 101-132.

Lisetti, C.L. & Schiano, D.J. 2000. *Automatic Facial Expression Interpretation: Where Human-Computer Interaction, Artificial Intelligence and Cognitive Science Intersect.* Pragmatics and Cognition (Special Issue on Facial Information Processing: A Multidisciplinary Perspective). Vol. 8:1. John Benjamins Publishing Company. pp. 185-235. ISSN 0929-0907 (print) 1569-9943 (electronic).

Long, L.N., Hanford, S.D., Janrathitikarn, O., Sinsley, G.L. & Miller, J.A. 2007. *A Review of Intelligent Systems Software for Autonomous Vehicles.* Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symposium on. pp. 69-76.

MacKenzie, D.C., Arkin, R.C. & Cameron, J.M. 1997. *Multiagent Mission Specification and Execution.* Autonomous Robots. Vol. 4:1. pp. 29-52.

Makarenko, A., Brooks, A. & Kaupp, T. 2007. *On the Benefits of Making Robotic Software Frameworks Thin.* Erwin Prassler, Klas Nilsson and Azamat Shakhimardanov. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07) Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware. San Diego, CA. November.

Makarenko, A., Brooks, A. & Kaupp, T. 2006. *Orca: Components for Robotics.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006). Workshop on Robotic Standadization. Beijing. October 10-13.

Makatchev, M. & Tso, S.K. 2000. *Human-robot interface using agents communicating in an XML-based markup language.* pp. 270-275. ISBN 0-7803-6273-X.

Martin, C., Schreckenghost, D., Bonasso, P., Kortenkamp, D., Milam, T. & Thronesbery, C. 2003. *An environment for distributed collaboration among humans and software agents.* AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. Melbourne, Australia. 14-18 July. New York, NY, USA: ACM. pp. 1062-1063. 1-58113-683-8.

MathWorks. *Simulink - Simulation and Model-Based Design.* [Cited 2008, 9/15/2008]. Available at: http://www.mathworks.com/products/simulink/.

MCA2. *Modular Controller Architecture.* [Cited 2009, 8/15/2009]. Available at: http://www.mca2.org/.

Microsoft. *Microsoft Robotics.* [Cited 2008, 9/15/2008]. Available at: http://msdn.microsoft.com/en-us/robotics/default.aspx.

Montemerlo, M., Roy, N. & Thrun, S. 2003. *Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit.* Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on. Vol. 3:pp. 2436-2441 vol.3.

Namoshe, M., Tlale, N.S., Kumile, C.M. & Bright, G. 2008. *Open middleware for robotics.* Mechatronics and Machine Vision in Practice, 2008. M2VIP 2008. 15th International Conference on. pp. 189-194.

National Instruments. *NI LabVIEW - The Software That Powers Virtual Instrumentation - Products and Services - National Instruments.* [Cited 2008, 9/15/2008]. Available at: http://www.ni.com/labview/.

Oksanen, T. 2007. *Path planning algorithms for agricultural field machines.* Teknillinen korkeakoulu; Helsinki University of Technology. Espoo, Finland. 110 pp. 978-951-22-9079-6.

Orebäck, A. & Christensen, H.I. 2003. *Evaluation of Architectures for Mobile Robotics.* Autonomous Robots. Vol. 14:1. pp. 33-49.

Orocos group. *The Orocos Project.* [Cited 2008, 9/12/2008]. Available at: http://www.orocos.org/.

Pembeci, I. & Hager, G. 2001. *A Comparative Review of Robot Programming Languages.* CIRL lab technical report.

Perzanowski, D., Schultz, A.C., Adams, W., Marsh, E. & Bugajska, M. 2001. *Building a multimodal human-robot interface.* Intelligent Systems, IEEE. Vol. 16:1. pp. 16-21. 1541-1672.

Pokahr, A., Braubach, L. & Lamersdorf, W. 2003. *Jadex: Implementing a BDI-Infrastructure for JADE Agents.* EXP - in search of innovation (Special Issue on JADE). Vol. 3:3. pp. 76-85.

Prassler, E., Lawitzky, G., Stopp, A., Grunwald, G., Hägele, M., Dillmann, R. & Iossifidis, I. 2005. *Preface.* In: Advances in Human-Robot Interaction, eds. B. Siciliano, O. Khatib & F. Groen,Germany: Springer. pp. 1-3. ISBN: 978-3-540-23211-7.

Rosta. *RoSta: Home.* [Cited 2009, 8/15/2009]. Available at: http://www.robot-standards.org/.

Saarinen, J., Maula, A., Nissinen, R., Kukkonen, H., Suomela, J. & Halme, A. 2007. *GIMnet - Infrastructure for distributed control of Generic Intelligent Machines.* Proceedings of the 13th IASTED International Conference on Robotics and Applications Telematics 2007.

Saarinen, J., Suomela, J. & Halme, A. 2002. *The Power Oar - Mechatronic rowing assistant* . 2nd IFAC Conference on Mechatronic Systems. Berkeley, California, USA. December 9-11. International Federation of Automatic Control. pp. 428-433.

Saarinen, J., Suomela, J., Halme, A. & Pavlicek, J. 2004. *Multientity rescue system.* Telematics applications in automation and robotics. Helsinki. International Federation of Automatic Control. pp. 55-60.

Schlegel, C. & Worz, R. 1999. *The software framework SMARTSOFT for implementing sensorimotor systems.* Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on. Kyongju, Korea. pp. 1610-1616 vol.3.

Selic, B., Gullekson, G. & Ward, P.T. 1994. *Real-time object-oriented modeling.* New York, NY, USA: John Wiley \& Sons, Inc. 0-471-59917-4.

Selkäinaho, J. 2002. *Adaptive autonomous navigation of mobile robots in unknown environments.* Teknillinen korkeakoulu; Helsinki University of Technology. Espoo, Finland.

Shakhimardanov, A. & Prassler, E. 2007. *Comparative evaluation of robotic software integration systems: A case study.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sheraton Hotel and Marina, San Diego, California, USA. October 29 - November 2. pp. 3031-3037.

Simmons, R. 1994. *Structured Control for Autonomous Robots.* IEEE Transactions on Robotics and Automation. Vol. 10:1. pp. 34-43.

Simmons, R. & Apfelbaum, D. 1998. *A Task Description Language for Robot Control.* Proceedings Conference on Intelligent Robotics and Systems. October.

Suomela, J. 2004. *From teleoperation to the cognitive human-robot interface.* Helsinki University of Technology. Espoo, Finland. ISBN 951-22-7387-X.

Suomela, J. & Halme, A. 2004. *Human Robot Interaction - Case WorkPartner.* Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems 2004 (IROS 2004). Sendai International Center, Sendai, Japan. pp. 3327-3332 vol.4. ISBN 0-7803-8463-6.

Suontama, H. 2003. *Use of signs for configuring tasks for service robots.* Helsinki University of Technology. Espoo, Finland.

Terho, S., Heikkilä, M., Taipalus, T., Saarinen, J. & Halme, A. 2006. *A framework for graphical programming of skilled tasks with service robots.* Proceedings of the 9th International Conference on Climbing and Walking Robots 2006.

Thrun, S. 2004. *Toward a Framework for Human-Robot Interaction.* Human-Computer Interaction. Vol. 19:1. pp. 9-24. 0737-0024.

TKK Automation Technology Laboratory. *WorkPartner.* [Cited 2009, 8/16/2009]. Available at: http://automation.tkk.fi/WorkPartner.

URBI. *URBI - Universal Real-time Behavior Interface - Accueil.* [Cited 2009, 8/15/2009]. Available at: http://www.urbiforge.com/.

W3C. *Extensible Markup Language (XML).* [Cited 2009, 8/15/2009]. Available at: http://www.w3.org/XML/.

Waldherr, S., Romero, R. & Thrun, S. 2000. *A Gesture Based Interface for Human-Robot Interaction.* Autonomous Robots. Vol. 9:2. pp. 151-173. ISSN 0929-5593.

Verma, V., Estlin, T., Jónsson, A., Pasareanu, C., Simmons, R. & Tso, S.K. 2005. *Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences.* International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS).

Verma, V., Jónsson, A., Pasareanu, C. & Iatauro, M. 2006. *Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations.* American Institute of Aeronautics and Astronautics Space 2006 Conference.

Verma, V., Jónsson, A., Simmons, R., Estlin, T. & Levinson, R. 2005. *Survey of Command Execution Systems for NASA Spacecraft and Robots.* "Plan Execution: A Reality Check" Workshop at the International Conference on Automated Planning & Scheduling (ICAPS).

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R. & Das, H. 2001. *The CLARAty architecture for robotic autonomy.* March 10-17. pp. 1/121-1/132 vol.1.

Ylönen, S. 2006. *Modularity in Service Robotics - Techno-Economic Justification through a Case Study.* Helsinki University of Technology. Espoo. 78 pp. 13 978-951-22-8502-0.

Appendices

Appendix 1: Base Class of Micro Task

An example of a base class of a micro task.

```
class TMTClass{
public:
        void (*eventCallback)(TMTEvent *event, void *callerObj); // The callback function pointer
        void *callerObj;    // Pointer to the object that registers the callback
        TConfig *Config;  // Pointer to config information      TMTClass(void);
        TMTClass(TConfig *C);
        virtual ~TMTClass(void);
        void  Log(string msg);
        int  virtual Init(TSingleTaskParameters &parameters); // Methods for micro tasks
        int  virtual Execute(void);
        int  virtual Stop(void);
        int  virtual Pause(void);
        int  virtual Continue(void);
        int  virtual UserStop(void);              // User wants to stop the execution of the task
        void virtual getSimulationInput(TSingleTaskParameters &pin);
        int  SendToWif(string target, string cmd); // Sending message to wif
        int  virtual WifResponse(int msgtype, int code, string source, string msg); // Receive message from wif
        string GetTaskName();
        TMTEvent *event;
        bool Running;      void setConfig(TConfig *C);
        void setCallback(void (*func)(TMTEvent *event, void *callerObj), void *obj);      // Use to register the callback
protected:
        int state;
        string taskName;
        TParameters initparameter;
private:
};
```

Appendix 2: Example of PlanLang

```
<?xml version="1.0"?>
<Plan category="WorkTask" taskName="ADDWOPAUSER" id="19283">
  <Parameter type="String" value="SANTA" name="USERNAME" description="THE NAME OF NEW WOPA USER"/>
  <Parameter type="Pixel" value="130 173 " name="SCREENXY" description="IDENTIFY THE USER FROM CAMERA
IMAGE"/>
  <ReturnValue type="Unknown" value="" name="RETURNVALUE0" description=""/>
  <Visualization uiBlockType="MicroTask" blockName="" blockID="0" uiPositionX="0" uiPositionY="0" InitText=""
ExeText="" EndText="" ErrorText=""/>
  <MicroTask MicroTaskName="STARTSTATE" category="Unknown" id="0" priority="Normal"
returnValue="STARTSTATE RETURNVALUE" currentValueDescription="">
    <Visualization uiBlockType="Start" blockName="STARTSTATE" blockID="0" uiPositionX="141" uiPositionY="21"
InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="PS_ANALYZECOLOR" category="Calc" id="32" priority="Normal" returnValue=""
currentValueDescription="">
    <Parameter type="Pixel" value="SCREENXY" name="SCREENXY" description="ROW AN COL FROM THE CAMERA
IMAGE"/>
    <Parameter type="Double" value="5" name="RADIUS" description="RADIUS OF THE AREA"/>
    <Parameter type="Double" value="3" name="VALUEMIN" description="MINIMUN BRIGHTNESS (V IN HSV)"/>
    <Parameter type="Double" value="255" name="VALUEMAX" description="MAXIMUM BRIGHTNESS (V IN HSV)"/>
    <ReturnValue type="Struct" value="" name="COLORINFO" description="HUE R G B - VALUES IN STRING"/>
    <Visualization uiBlockType="MicroTask" blockName="PS_ANALYZECOLOR" blockID="8986" uiPositionX="218"
uiPositionY="114" InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="DB_ADDUSER" category="Calc" id="26" priority="Normal" returnValue=""
currentValueDescription="">
    <Parameter type="String" value="USERNAME" name="USERNAME" description="THE NAME OF  THE USER"/>
    <Parameter type="Struct" value="COLORINFO" name="COLORINFO" description="THE COLOR INFO OF  THE USER -
USED IN DATA ASSOCIATION"/>
    <Parameter type="Position" value="LOCATEDPOS" name="USERPOSITION" description="THE CURRENT POSITION
OF  THE USER"/>
    <Visualization uiBlockType="MicroTask" blockName="DB_ADDUSER" blockID="8712" uiPositionX="-17"
uiPositionY="416" InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="ILMR_SPEAK" category="ILMR" id="0" priority="Normal" returnValue=""
currentValueDescription="MT_ILMR_SPEAK - NOW WITH UNLIMITED INPUTS!">
    <Parameter type="String" value="USER" name="MT_ILMR_SPEAK_IN" description="INPUT FOR MT_ILMR_SPEAK"/>
    <Parameter type="String" value="USERNAME" name="MT_ILMR_SPEAK_IN" description="INPUT FOR
MT_ILMR_SPEAK"/>
    <Parameter type="String" value="SUCCESFULLY ADDED" name="MT_ILMR_SPEAK_IN" description="INPUT FOR
MT_ILMR_SPEAK"/>
    <ReturnValue type="Integer" value="" name="ISOK" description="THE STANDARD RETURN VALUE OF THE TASK"/>
    <Visualization uiBlockType="MicroTask" blockName="ILMR_SPEAK" blockID="91700" uiPositionX="160"
uiPositionY="516" InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="PTU_TURN" category="Calc" id="18" priority="Normal" returnValue=""
currentValueDescription="TURNING HEAD">
    <Parameter type="Double" value="0" name="PTUPAN" description="PTU PAN VALUE IN PTU'S OWN UNITS"/>
    <Parameter type="Double" value="-500" name="PTUTILT" description="PTU TILT VALUE IN PTU'S OWN UNITS"/>
    <ReturnValue type="Integer" value="" name="ISOK" description="THE STANDARD RETURN VALUE OF THE TASK"/>
    <Visualization uiBlockType="MicroTask" blockName="PTU_TURN" blockID="49476" uiPositionX="-2"
uiPositionY="635" InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="FINALSTATE" category="FlowControl" id="1" priority="Normal"
returnValue="FINALSTATE RETURNVALUE" currentValueDescription="STOPPING WORK TASK">
```

```
   <Visualization uiBlockType="Stop" blockName="FINALSTATE" blockID="1" uiPositionX="232" uiPositionY="724"
InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="PS_LOCATEOBJECT" category="Calc" id="33" priority="Normal" returnValue=""
currentValueDescription="LOCATING">
   <Parameter type="Struct" value="COLORINFO" name="COLORINFO" description="HUE OF THE COLOR"/>
    <Parameter type="Double" value="10" name="HUEVARIATION" description="VARIATION OF HUE"/>
   <Parameter type="Double" value="30" name="MINSATURATION" description="MINIMUM SATURATION"/>
   <Parameter type="Double" value="255" name="MAXSATURATION" description="MAXIMUM SATURATION"/>
   <Parameter type="Double" value="1000" name="MINAREA" description="MINIMUM AREA IN PIXELS"/>
   <Parameter type="Double" value="70" name="UNTIL" description="SEARCH UNTIL"/>
   <Parameter type="Double" value="6000" name="TILTLIMIT" description="MAXIMUM TILT VALUE"/>
   <Parameter type="Double" value="-8000" name="MINPAN" description="MINIMUM PAN VALUE"/>
   <Parameter type="Double" value="8000" name="MAXPAN" description="MAXIMUM PAN VALUE"/>
   <Parameter type="Bool" value="1" name="TURNLASERON" description="SET IF THE LASER IS USED"/>
   <Parameter type="Bool" value="1" name="LeaveLaserOn" description="Set if the laser is left on"/>
   <Parameter type="Double" value="-20" name="XOFFSET" description="X OFFSET OF MANIPULATOR"/>
   <Parameter type="Double" value="30" name="YOFFSET" description="Y OFFSET OF MANIPULATOR"/>
   <Parameter type="Double" value="0" name="ZOFFSET" description="Y OFFSET OF MANIPULATOR"/>
   <Parameter type="String" value="" name="Region3D" description="3D region of interest"/>
   <Parameter type="Bool" value="0" name="ManyTargets" description="Many targets?"/>
   <ReturnValue type="Double" value="" name="WOPAX" description="X COORDINATE IN WOPAS COORDINATES"/>
   <ReturnValue type="Double" value="" name="WOPAY" description="Y COORDINATE IN WOPAS COORDINATES"/>
   <ReturnValue type="Double" value="" name="WOPAZ" description="Z COORDINATE IN WOPAS COORDINATES"/>
   <ReturnValue type="Position" value="" name="LOCATEDPOS" description="THE POSITION OF THE LOCATED
OBJECT"/>
   <ReturnValue type="Bool" value="" name="ISFOUND" description="WAS THE OBJECT FOUND"/>
   <ReturnValue type="Struct" value="" name="ANGLES" description="MANIPULATOR ANGLES"/>
   <ReturnValue type="Double" value="" name="DISTANCE" description="DISTANCE TO BE MOVED FORWARD"/>
   <ReturnValue type="Bool" value="" name="ISZERO" description="IS THE DISTANCE ZERO"/>
   <Visualization uiBlockType="MicroTask" blockName="PS_LOCATEOBJECT" blockID="3114" uiPositionX="31"
uiPositionY="222" InitText="" ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <MicroTask MicroTaskName="If" category="FlowControl" id="4" priority="Normal" returnValue="If returnvalue"
currentValueDescription="Starting If">
   <Parameter type="Integer" value="ISFOUND" name="ifinput" description=""/>
   <ReturnValue type="Integer" value="" name="ifoutput" description=""/>
   <Visualization uiBlockType="If" blockName="If" blockID="9" uiPositionX="100" uiPositionY="317" InitText=""
ExeText="" EndText="" ErrorText=""/>
  </MicroTask>
  <Connections>
   <Connect From="0" FromPos="3" To="8986" ToPos="1"/>
   <Connect From="8712" FromPos="3" To="91700" ToPos="1"/>
   <Connect From="49476" FromPos="3" To="1" ToPos="1"/>
   <Connect From="91700" FromPos="3" To="49476" ToPos="1"/>
   <Connect From="8986" FromPos="3" To="3114" ToPos="1"/>
   <Connect From="3114" FromPos="3" To="9" ToPos="1"/>
   <Connect From="9" FromPos="2" To="3114" ToPos="2"/>
   <Connect From="9" FromPos="3" To="8712" ToPos="1"/>
  </Connections>
</Plan>
```

Appendix 3: List of Implemented Micro Tasks

MT_ArithmeticOperation	Calculation

Make calculation between the input parameters. Following operators are used for calculation. The result(s) of calculation is stored to return value(s) (=output).
Usage:
input0: operator
input1: variable #1
input2: variable #2
...
output0: result #1
output1: result #2

Operator	**Inputs**	**Outputs**
SET or COPY	1..20	same as inputs: outputN = inputN
+ or SUM	1..20	1 : output0 = input0 + input1 + ... + inputN)
- or SUB	2	1 : output0 = input0 - input1)
or PROD	1..20	1 : output0 = input0 input1 * ... * inputN
/ or DIV	2	1 : output0 = input0 / input1
MOD	2	1 : output0 = input0 % input1
XOR	2	1 : output0 = input0 ^ input1
AND	2	1 : output0 = input0 & input1
OR	2	1 : output0 = input0 | input1
SQRT	1..20	same as inputs : outputN = SQRT(inputN)
SQUARE	1..20	same as inputs : outputN = inputn inputN
SIN	1..20	same as inputs : outputN = SIN(inputN)
COS	1..20	same as inputs : outputN = COS(inputN)
TAN	1..20	same as inputs : etc...
ASIN	1..20	same as inputs
ACOS	1..20	same as inputs
ATAN	1..20	same as inputs
ATAN2	2	1 : output0 = ATAN2(input0, input1)
MIN	1..20	1 : output0 = MIN(input0, input1, ..., inputN)
MAX	1..20	1 : output0 = MAX(input0, input1, ..., inputN)
MEAN	1..20	1 : output0 = MEAN(input0, input1, ..., inputN)
NORM	1..20	1 : output0 = SQRT(input0*input0 + input1*input1 + ... + inputN*inputN)
DEGTORAD	1..20	same as inputs : outputN = inputN pi/180
RADTODEG	1..20	same as inputs : outputN = inputN 180/pi

Parameters	1	String	Operator
	2	Double	1st input value
	3-20	Double	Input values 2-20.
Return values	1	Double	1st result value
	2-20	Double	2nd result value
	3	Double	Result values 3-20

MT_ComparePlanexVariable			Calculation
Compare input parameters: Usage: input0: first number input1: condition input2: second number output0: - true: 1 - false: 0 **condition** **output0 = 1 (true), if:** < or LESS input0 < input1 > or GREATER input0 > input1 = or == or EQUAL input0 == input1 != or NOT EQUAL input0 != input1			
Parameters	1	Double	First value to be compared
	2	String	Condition (see above). Operator used for comparison.
	3	Double	Second value to be compared
Return values	1	Double	Result (true or false)

MT_generateGreeting			Util
Generate greeting message to user. The message form is "Hello USER! Nice to see you". USER is changed the name of the user. Input parameter of the micro task is identification number of database object (aka id of the user). For example detect_User micro tasks returns the id of the user.			
Parameters	1	Object	Database ID of the user. The user for whom the greeting is made.
	2	String	Strings are added to greeting: param2 + username + param3 + param4 + … +param6
	3-6	String	Strings are added to greeting: param2 + username + param3 + param4 + … +param6
Return values	1	String	Greeting message to user.

MT_Sleep			Util
Holds the plan execution to specific period of time.			
Parameters	1	Double	Time delay in seconds.
Return values	1		No return values

MT_DebugToFile			Util
Write debug information to the text file.			
Parameters	1	String	1st line of debug message
	2	String	2nd line of debug message
	3-8	String	Lines 3-8 of debug messages
Return values	1		No return values

MT_GetWopaVariables			Util
Read measurements from the robot. This micro task returns direct measurement values from the robot. This is used mainly for debugging.			
Parameters	1	String	Name of the measurement 1.
	2	String	Name of the measurement 2.
	3-5	String	Names of the measurements 3-5.
Return values	1	String	Value of the measurement 1.
	2	String	Value of the measurement 2.
	3-5	String	Values of the measurements 3-5.

MT_SetWopaVariables			Util
Send direct command to the robot. This used mainly for debugging.			
Parameters	1	String	Name of the variable 1.
	2	String	Value of the variable 1.
	3,5,7,9,11,13	String	Names of the variables 2-7.
	4,6,8,10,12,14	String	Values of the variables 2-7.
Return values	1	Integer	0=not OK, 1=OK

MT_param_copy			Util
Copy values between the variables. - Possible to change the names of variables - Possible to make duplicates of variables. For example saving parameter value for later comparison Output values are same as inputs: outputN = inputN			
Parameters	1	Unknown	1st input value
	2	Unknown	2nd input value
	3-20	Unknown	Input values 3-20
Return values	1	Unknown	Equal to 1st input value
	2	Unknown	Equal to 2nd input value
	3-20	Unknown	Equal to input values 3-20

MT_stereo_start			Speech
Starting the stereo hearing. Stereo hearing is external service. This micro task just sends a startup command for the service.			
Parameters	1		No parameters
Return values	1		No return values

MT_stereo_stop			Speech
Stopping the stereo hearing. Stereo hearing is external service. This micro task just sends a stop command for the service.			
Parameters	1		No parameters
Return values	1		No return values

MT_UIspeak			Speech
Speaks sentence from end user GUI with speech synthesizer. This micro task is suitable for the user if he does not hear speech from the robot (long distance for example).			
Parameters	1		Sentence to be spoken.
Return values	1		No return values

MT_Voice			Speech
Control the behavior of voice service. Both speech recognition and synthesize is used. Speech is synthesized on the robot.			
Parameters	1	String	The control command for voice. Can be 'color', 'who', 'task', 'confirm', 'object', 'stop', 'attention' for speech recognition or 'speak' for speech synthesizer. For example use of ‘color’ command requests the voice service to ask and wait color word said by the user.
	2	String	Sentence to be spoken if the control command is ‘speak’.
Return values	1	String	Recognized text - nothing in case of speak.
	2	Integer	In case confirmation the value is 1 == Yes, 0 == No

MT_db_addObject			Database
Add object to the database. Typical objects are physical object detected with the perception system.			
Parameters	1	String	Type of the object. This textual type “box” or “litter” for example.
	2	String	Name of the object.
	3	Position	Position of the object
	4	Integer	Geometry type of the object (ball=1, cube=2, cone=3, torus=4 or teapot=5)
	5	Struct	Color information for the object. Hue Red Green Blue values
	6	Double	Length (diameter) of the object
	7	Double	Height of the object
Return values	1	Object	Object ID (-1 in case of error)

MT_db_addUser			Database
Store the user name specified with a color to the database.			
Parameters	1	String	Name of the user.
	2	Struct	Color information of the user [Hue R G B]
	3	Position	Current position of the user.
Return values	1	Integer	1 if OK, 0 if error.

MT_db_getObjectInfo			Database
Return information of database object.			
Parameters	1	Object	Object ID to get from the database.
Return values	1	String	The name of the object.
	2	String	The type of the object.
	3	Position	The position of the object.
	4	Struct	The color information of the object.

MT_db_getUsers			Database
Return list of users in the database.			
Parameters	1		No parameters
Return values	1	Integer	Number of users in the database.
	2	Object	ID of the user.
	3-20	Object	ID of the user.

MT_db_RemoveObject			Database
Remove database object.			
Parameters	1	Object	ID of the removed object
Return values	1	Integer	1=OK, -1=not OK

MT_db_setObjectPosition			Database
Set position for database object.			
Parameters	1	Object	ID of the object that is changed.
	2	Position	The new position for the object.
Return values	1		No return values

MT_db_typeAndNameSearch			Database
Search object from the database.			
Parameters	1	String	The control for the search: type, name or and
	2	String	The type of searched object.
	3	String	The name of the searched object.
Return values	1	Integer	The number of objects found
	2	Object	ID of object found.
	3-20	Object	ID of object found.

MT_Manip_SetAngles			Manipulator
Move joints of the manipulator to specified position. Named angles: HOME, PREPARETOGRAB, BASKET, LEAN, FRONTRELEASE, FRONTGRAB, GROUNDGRAB, GROUNDRELEASE, SHAKE1, SHAKE2, ZERO, TRASNPORT, GIVE, GO_DOWN, DOWN_RELEASE, HANDS_DOWN_FRONT, TOP_OF_OBJECT Angles structure: (mrot)angle(!mrot) Set joint of manipulator's body rotation. (angle in degrees) (minc)angle(!minc) Set joint of manipulator's body inclination. (mrro)angle(!mrro) Set joint of manipulator's right arm (shoulder) inclination. (mrin)angle(!mrin) Set joint of manipulator's right arm (shoulder) rotation. (mrel)angle(!mrel) Set joint of manipulator's right arm elbow. (mwri)angle(!mwri) Set joint of manipulator's right wrist inclination (mwrr)angle(!mwrr) Set joint of manipulator's right wrist rotation. (mlro)angle(!mlro) Set joint of manipulator's left arm (shoulder) inclination. (mlin)angle(!mlin) Set joint of manipulator's left arm (shoulder) rotation. (mlel)angle(!mlel) Set joint of manipulator's left arm elbow. (mwli)angle(!mwli) Set joint of manipulator's left wrist inclination (mwlr)angle(!mwlr) Set joint of manipulator's left wrist rotation.			
Parameters	1	String	Manipulator angles structure or named angles.
	2	String	Grip or ungrip
	3	Double	Delay before changing the gripper state
Return values	1	Integer	0=not OK, 1=OK

MT_ManipulatorRightXYZToAngles			Manipulator
Move arm of the manipulator to specified position.			
Parameters	1	Integer	X coordinate of right hand
	2	Integer	Y coordinate of right hand
	3	Integer	Z coordinate of right hand
	4	Integer	Angle between ground and gripper
	5	Integer	Use negative elbow angle?
Return values	1	String	Rotation angle of manipulator's body
	2	String	Rotation angle of right shoulder
	3	String	Inclination angle of right shoulder.
	4	String	Inclination angle of right elbow.
	5	String	Inclination angle of right wrist.

MT_PTU_Turn			PTU
Turn the head of the robot to specific position. (PTU = pan-tilt unit)			
Parameters	1	Double	PTU pan value in PTU's own units.
	2	Double	PTU tilt value in PTU's own units.
Return values	1	Integer	0=not ok, 1=OK

MT_Laser_WaitForDistance			Perception
Wait until (or timeout) something is between smallest and biggest distances.			
Parameters	1	Double	PTU pan value in PTU's own units.
	2	Double	PTU tilt value in PTU's own units.
	3	Double	Timeout in seconds.
	4	Double	Smallest accepted distance.
	5	Double	Biggest accepted distance.
Return values	1	Bool	Was the object found (true or false)

MT_PS_ColorNameToColorInfo			Perception
Get color data values for selected color name.			
Parameters	1	String	The name of the color.
Return values	1	Struct	Struct [Hue R G B] of the color.

MT_PS_detectUser			Perception
Detect if the user (from database) is on camera image.			
Parameters	1	Double	Allowed variation in Hue.
	2	Double	Minimum saturation.
	3	Double	Maximum saturation.
	4	Double	Minimum allowed area.
Return values	1	Object	The database ID of the user if the user was found (else -1).
	2	Pixel	The center of the segmented user.
	3	Position	Pan and tilt value – to where the PTU should be turned if the object is centered.
	4	Integer	Indicates if the user is found from image.

MT_PS_detectColoredObject			Perception
Detect if there is object with specified color on the camera image.			
Parameters	1	Struct	The hue values (integer) that are searched [hue0 hue1…9
	2	String	The wanted colors as string (red, green, blue, violet)
	3	Double	Allowed variation in Hue
	4	Double	Minimum saturation
	5	Double	Maximum saturation
	6	Double	Minimum allowed area.
Return values	1	Integer	True (=1) if there is large enough object of searched color.
	2	Integer	The hue of found object.
	3	Struct	The color information of the object [Hue R G B]
	4	Position	Pan and tilt value – to where the PTU should be turned if the object is centered.
	5	String	The color of the object that was detected.
	6	Integer	Tells if this object was wanted color object.

MT_PS_LocateGripper			Perception
Measures the position of the robot's hand.			
Parameters	1	Struct	Specify the color: [HueMin HueMax SatMin SatMax ValMin ValMax]
	2	Struct	The variation of distance
Return values	1	Double	X-offset of the gripper.
	2	Double	Y-offset of the gripper.
	3	Double	Z-offset of the gripper.
	4	Double	Result of the operation (ok=1, not OK=0)

MT_PSAnalyzeColor			Perception
Analyze the color of selected point on image screen.			
Parameters	1	Pixel	Row and column from the camera image.
	2	Double	Radius of the area.
	3	Double	Minimum brightness (V in HSV)
	4	Double	Maximum brightness (V in HSV)
Return values	1	Struct	Color information [Hue R G B]

MT_PSFindLargestObject			Perception
Find targets area of specified color from the camera image.			
Parameters	1	Struct	Color information of the object that is located [Hue R G B]
	2	Double	Variation of hue
	3	Double	Minimum saturation.
	4	Double	Maximum saturation.
	5	Double	Minimum area in pixels.
Return values	1	Double	X coordinate of object center.
	2	Double	Y coordinate of object center.
	3	Double	Z coordinate of object center.
	4	Double	Tilt value that moves the object to center of image.
	5	Double	Area of object in pixels.
	6	Integer	Was the object found (OK=1, not OK=0)

MT_PSSick_detectUser			Perception
Detect the user using data from the SICK laser scanner and from camera.			
Parameters	1	Double	Allowed variation of hue
	2	Double	Minimum saturation
	3	Double	Maximum saturation
	4	Double	Minimum area in pixels
Return values	1	Object	The DB id of the user IF the user was found (else -1)
	2	Pixel	The center of the segmented user
	3	Position	Pan and Tilt value - to where the PTU should be turned if the user is centered
	4	Position	The position of the user in WorkPartner’s coordinates
	5	Double	The distance to the user based on the sick
	6	Integer	Indicates if the user is found from image

MT_PTU_searchObject			Perception
Search object from the camera image while turning the head of the robot.			
Parameters	1	Object	Object id to get from the database
	2	Double	Allowed variation of hue
	3	Double	Minimum saturation
	4	Double	Maximum saturation
	5	Double	Minimum area in pixels
	6	Double	The time in ms that the PTU is waited to turn
Return values	1	Integer	Indicates if the object was found from image
	2	Position	The position of the found object

MT_PSLocateObject			Perception
Measure the position of object related to the robot. Object is specified with a color.			
Parameters	1	Struct	Hue of the color
	2	Double	Allowed variation of hue
	3	Double	Minimum saturation
	4	Double	Maximum saturation
	5	Double	Minimum area in pixels
	6	Double	Search until
	7	Double	Maximum tilt value
	8	Double	Minimum pan value
	9	Double	Maximum pan value
	10	Bool	Set if the laser range finder is used
	11	Bool	Set if the laser is left on after tracking
	12	Double	X offset of manipulator
	13	Double	Y offset of manipulator
	14	Double	Z offset of manipulator
Return values	1	Double	X coordinate of object in WorkPartner’s coordinates
	2	Double	Y coordinate of object in WorkPartner’s coordinates
	3	Double	Z coordinate of object in WorkPartner’s coordinates
	4	Position	The Position of the located object
	5	Bool	Was the object found
	6	Struct	Manipulator angles
	7	Double	Distance to be moved forward

Note. ILMR micro tasks are basically external services for intermediate level. So it is sending ILMR commands handled on the robot.

MT_ILMR_BodyPose			ILMR (See [Kauppi, 2003]]
Set robot's orientation.			
Parameters	1	String	Input for the BodyPose command (park, wheel, rolk)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Goahead			ILMR (See [Kauppi, 2003]]
Move a robot ahead or back			
Parameters	1	String	Distance to go ahead (in meters)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_GoaheadAdvanced			ILMR (See [Kauppi, 2003]]
Move a robot ahead or back. Automatically change move mode etc.			
Parameters	1	Double	Distance to go ahead (in meters)
	2	Double	Speed of moving (m/s)
	3	Bool	Determine the sign of speed from distance
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Gotoxy			ILMR (See [Kauppi, 2003]]
Move a robot to a desired position			
Parameters	1		X,Y of target location
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_GotoxyAdvanced			ILMR (See [Kauppi, 2003]]
Move a robot to a desired position. Automatically change move mode etc.			
Parameters	1	String	X,Y of target location
	2	Double	Speed of moving
	3	Double	Distance to stop in advance
	4	Bool	Use intelligent obstacle avoidance
	5	String	What is said to person
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Locate			ILMR (See [Kauppi, 2003]]
Initialize the coordinate system.			
Parameters	1	String	Location of current position [x,y,dir]
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_locMode			ILMR (See [Kauppi, 2003]]
Select a navigation mode			
Parameters	1	String	Input for locmode command (automatic, gps and odo for example)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_MoveMode			ILMR (See [Kauppi, 2003]]
Set a mode for movement in a robot.			
Parameters	1	String	Input for movemode command (park, wheel, rolk)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_ObsAvoid			ILMR (See [Kauppi, 2003]]
Select a behavior for a robot when it encounters an obstacle			
Parameters	1	String	Input for obsavoid command (ON, OFF or STOP)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_savepos			ILMR (See [Kauppi, 2003]]
Save the current position to be used later.			
Parameters	1	String	Input for savepos command (name for the current position)
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Speak			ILMR (See [Kauppi, 2003]]
Speak out from the speakers on the robot.			
Parameters	1	String	Sentence to be speak out.
	2	String	Sentence to be speak out.
	3-20	String	Sentence to be speak out.
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Speed			ILMR (See [Kauppi, 2003]]
Set speed for a robot			
Parameters	1	String	Input for the speed command. (speed in m/s).
Return values	1	Integer	0=not OK, 1=OK

MT_ILMR_Turn			ILMR (See [Kauppi, 2003]]
Make a robot turn to the left or right			
Parameters	1	String	Input for the turn command. (turning angle, curvature etc.)
Return values	1	Integer	0=not OK, 1=OK