

A Java-based Framework for the Programming of Distributed Systems for Mobile Robots

Daniel Westhoff and Hagen Stanek

genRob GmbH, Aidlingen, Germany
{westhoff,stanek}@genRob.com

Abstract. We propose a novel concept for the programming of distributed systems for mobile robots. A software architecture is presented that eases the development of applications for mobile robots. This software architecture is based upon the Roblet-Technology, which is a powerful medium for robots. It introduces the possibility to develop, compile and execute a distributed application on one workstation. The fundamental paradigm of the Roblet-Technology is the strong use of mobile code. Using mobile code an application distributes parts of itself through the network and builds up a distributed application. Since the Roblet-Technology uses Java the development is independent of the operation system. With the feature of running programs as a distributed software, the framework allows running algorithms which need great computation power on different machines which provide this power. In this way, it greatly improves programming and testing of applications in service robotics. We provide several examples of complex applications which were developed using our framework. They all have in common that they use the Roblet-Technology to combine several independently developed software components.

1 Introduction

Robotic systems are becoming more and more complex. The number of constitutional parts that make up current robotic research platforms is increasing. A multitude of sensors can be found in these robots: tactile sensors from basic bumper switches to force and torque sensors, range measuring systems like infrared, ultra-sonic, radar and laser based sensors or vision systems including cameras as different as low-cost web-cams and high-dynamic-range cameras. On the actuator side one finds mobile robot platforms with a variety of drive systems, walking or climbing robots, robot arms with different degrees of freedom or complex multi-finger robotic hands. In service robotics all these are combined in autonomous mobile manipulators that accomplish tasks in a diversity of applications.

The field of service robotics has seen a lot of advances over the last years, but still lacks usability and robustness. We think that one reason for this is the absence of a unifying software architecture that handles the miscellaneous challenges which the software engineers encounter. These challenges vary from

the development of distributed applications to the handling of the diversities of different hardware platforms present in service robotics.

Over the last years, the research community has come to realise that the ambitious objectives of robotic research can only be reached based on solid software architectures. These architectures must support the requirements of the heterogeneous modern robot systems. Briefly summarised, the main requirements are: hardware abstraction, extendability, scalability, limited run-time overhead, actuator control, modularisation, support for networked computing, simplicity, consistency, completeness, support for multiple operating systems. [1] and [2] have conducted surveys and evaluations of existing software systems. They provide a good elaboration on the merits and demerits of these architectures.

In this paper we propose a framework that meets these challenges and enables a programmer to develop advanced applications for service robots. A main feature of the framework is the ability to integrate existing solutions to specific robotic problems. We will show that it is possible to encapsulate libraries for motion control for manipulators as well as for mobile robots. A variety of hardware devices connected to a service robot will be integrated into the architecture. A layer of abstraction will generalise the access to these devices. Thus, developed applications can be transferred to other robotic systems without changes.

The remainder of this paper is organised as follows: In section 2 an overview of existing software architectures in robotics is given. This is followed by a discussion of the merits and demerits of the existing software developments. Motivated by this, section 4 introduces our software architecture and how hardware is encapsulated by the proposed framework. In Section 6 applications are presented where the framework was applied successfully. Section 7 gives a conclusion and an outlook on future work.

2 Related Research

This section gives an overview of existing software architectures for service robots. Recently, a workshop during the 2004 conference on Intelligent Robots and Systems (IROS) tried to list the various research activities in the field of robotic middleware [3]. One year later a similar workshop was held at the 2005 Conference on Robotics and Automation [4]. The outcome of the second workshop is collected in [5]. In the following, some of the activities in the field of robotic software environments are discussed. Besides, further related research projects are stated.

The *OROCOS* project started in 2000 as a free software project due to the lack of reliable commercial robot control software [6]. It is divided into two decoupled sub-projects: *Open Realtime Control Services* and *Open Robot Control Software*. The first one is a real-time software framework for applications for machine control. The second one is a set of libraries and an application framework including generic functionality mainly for manipulators. Support of mobile robots is still in its early stages.

In 2004 the *Orca* project emerged from the OROCOS project [7]. It adopts a component-based software engineering approach using *Ice* [8] for communication and the description of interfaces. The project's goals are to enable and to simplify software reuse and to provide a generic repository of components. The use of different middleware packages for inter-component communicating is extensively discussed on the project's home page. Beside writing custom middleware, the use of CORBA and XML-based technologies is compared to Ice. Orca is available for various operating systems and compiles natively.

[9] introduces the *Player/Stage* project, a client-server framework to enable research in robot and sensor systems. It provides a network interface to a variety of robot and sensor hardware and to multi-robot simulators. Multiple concurrent client connections to the servers are allowed. Client applications connect over TCP sockets. The project's server software and the simulators are limited to Unix-like operating systems.

In [10] MARIE is presented, a design tool for mobile and autonomous robot applications. It is mainly implemented in C++ and it uses the *ADAPTIVE Communication Environment* (ACE) [11] for communication and process management.

In 2002 *Evolution Robotics* introduced the *Evolution Robotics Software Platform* (ERSP) for mobile robots [12]. It is a behaviour-based, modular and extensible software available for Linux and Windows systems. The main components that are included are vision, navigation and interaction.

In December 2006 Microsoft released the first stable version of their robot software development kit *Microsoft Robotics Studio* [13]. The kit features a visual programming language to create software for robot systems, a 3D simulated environment and a runtime to execute the programs on the robot hardware. The use of the software is restricted to several versions of Microsoft Windows including Windows CE for mobile applications. It is strongly based on Microsoft's .NET framework.

In [14] a service robot for a biotechnological pilot laboratory is presented. The mobile platform of this robot is equal to parts of TASER which is presented in this paper. A seven degrees-of-freedom arm is mounted on top of the mobile platform. The system is designed to take samples from a sampling device, handle a centrifuge, a fridge and other biotechnological equipment and fulfil the complete process of sample management. It relieves the personal of the laboratory of monotonous time consuming tasks. Nevertheless it operates in a standard laboratory with standard equipment. An easy-to-use script language is proposed to define high-level work sequences. The scripts are parsed by the robot's control software and the robot fulfils the defined task. This encourages the idea of simplifying the programming of robots but lacks the flexibility of a widespread programming language including network programming for distributed systems.

3 Discussion of existing software frameworks

The main contradiction a programmer of a service robot has to deal with is the trade-off between easy operability and full flexible utilisation. Easy operability means the system should be easily programmable by non-experts. To allow this, current research investigates man-machine interfaces for natural instruction and communication. Unfortunately, most of these interfaces are only available as prototypes. But, commercial implementations of these techniques will not be available for some years to come. One compromise was to use script languages as in [14]. These languages are easy to learn and reduce the amount of knowledge needed to operate the robot. They abstract low-level details of the robot control and provide mid-level functionality. This midlevel functionality is then used to implement high-level task-oriented applications.

On the other hand script languages have serious limitations. They only allow alteration of certain parameters and sequences up to a certain point. Modifications or new tasks can only be implemented if they do not require more functionality than offered by the script language. Anything that is not possible within the functionality of the mid-level script language is therefore not possible at the task-oriented level. Such modifications would require access to low level details, but are intentionally hidden by the script languages.

As a general conclusion a framework for mobile robots should provide an easy-to-learn high-level interface for non-expert personnel, while also providing access to low-level details. This allows adding functionality that is missing in the high-level interface.

Most of the presented frameworks can be seen as component-oriented architectures and thereby try to address the above mentioned problems. The Roblet-Technology presented in section 4 is a novel component based architecture. Since its basis is one programming language the developers benefit of a unified programming environment which eases the exchange of knowledge within a project. Especially concerning the maintenance of complex robotic systems and the frequent changes of developers in scientific projects and institutions we think this will be an immense advantage over other robotic development environments.

4 Software Architecture

In this section we propose a novel software architecture to ease the development of high-level programs combining the functionality of robotic subsystems.

Many service robot systems are developed for a specific task like mail delivery, hospital service or laboratory services. In order to keep the software maintainable the low-level details of the system are hidden by a hardware abstraction layer (HAL). The task-level programs implementing the service are then based on this HAL. Using the robot for a different service often can not be done by simply writing a new task-level program, but requires additional low-level changes as well. In existing systems this cannot be done while the robot operates. We will

explain how our architecture allows easy task-oriented programming by providing high-level functionality as well as access to parts of the low-level architecture. Otherwise, adding new functionality to perform new or even only slightly changed tasks would not be possible.

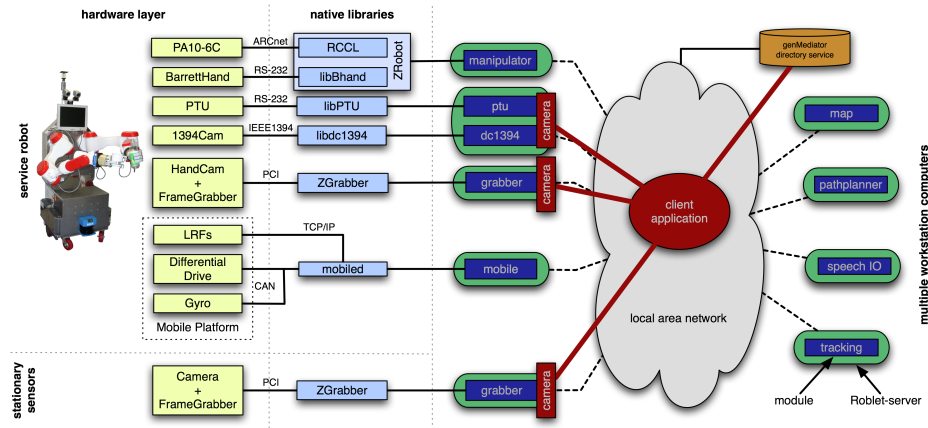


Fig. 1. The software architecture of the robot TASER at the University of Hamburg: Roblet-servers (RS) are used to provide a hardware abstraction layer. Generalisation is realised by this hardware abstraction. Distributed applications are independent of the particular hardware of the robot system. Some Roblet-servers and connections are left out for clarity. The client application is an example on how different hardware is encapsulated. The client application uses only the unit *camera* which unifies the access to cameras. The actual type of hardware is not known to the client application.

4.1 Roblets

The basics of the proposed framework are realised with Java and use Roblet-Technology, a concept firstly introduced in [15]. Roblet-Technology is a client-server architecture where clients can send parts of themselves, referred to as Roblets, to a server. The server, referred to as Roblet-server, then executes the Roblets with well-defined behaviour in case of malfunctions. Notice that not only data is transmitted between the client and server but complete executable programs. This can be compared to Java Applets but with the difference that Roblets are not downloaded but sent. Complex setups can consist of multiple client applications and Roblet-servers. A Roblet terminates if the execution of its code finishes normally or throws an exception. Exceptions are sent back to the client application. In addition, a Roblet can be terminated by a client application remotely or by the Roblet-server directly. After a Roblet terminates, the Roblet-server resets itself to a well defined state.

In section 5 we give a short example how a simple distributed application looks like when it is programmed with our framework.

Roblet-Technology is applicable to all kinds of distributed systems but it has several features that make its integration into robotic applications useful. In general, high-level applications in service robotics are mostly distributed systems. Besides one or multiple mobile robots, there are visualisation- and control applications that run on workstations in a local area network. Sometimes there is no direct access to the robot systems via keyboard, mouse and monitor but only through a wireless network. Roblet-Technology introduces the possibility to develop, compile and execute an application on one workstation. When the application is executed it will send parts of itself to available servers and spread in the local network. Roblets may send parts of themselves to other servers as well. The network communication is hidden from the programmer by the Roblet library, which simplifies the overall development. That means, the network is transparent and developing distributed applications based on Roblet-Technology is like developing one application for one workstation. Access to the remote servers is encapsulated in a client library, reducing the execution of a Roblet on the remote system to one method call.

4.2 Modules

For robotic applications we propose *modules* to extend the basic Roblet-server provided by the Roblet framework. A module is loaded when the Roblet server is started. It is meant to encapsulate a class of similar functionality.

For the robot TASER of the University of Hamburg we developed several modules. A more detailed explanation of this robot is given in section 6. One module merges the functionality of the mobile platform, a second module wraps the manipulator system including the robot arms and the hands. There are modules for the different vision systems, the pan-tilt unit, a speech module and other parts of the interaction subsystem. Figure 1 gives an overview of the main parts of the current software architecture for TASER. The system incorporates several smaller Roblet-servers and multiple client applications not shown in the figure for clarity. Notice that the map server and the pathplanning server don't run on the robot's control computer but on a workstation in the local network. This allows the integration of information gathered by multiple robots. For example, in the case of dynamic map adjustment this relieves the robot's on-board computer of some computationally expensive tasks which need no real-time capabilities.

4.3 Units

Modules are further divided in *units*. Units are Java interfaces that are implemented within the modules. Units build the hardware abstraction layer in our framework. For example, a module encapsulates the localisation subsystem of a mobile robot and a Roblet wants to query the current *pose* estimate¹ of the robot. Then the module would implement a unit which defines a method to get

¹ A *pose* is the triple of 2D position coordinates and the robot's orientation.

the pose. On another robot there may be another localisation system encapsulated by another module. But, if the module implements the same unit, the same roblet can be executed on both robots and works without changes. Nonetheless, special features of a subsystem are made available to Roblets if module-specific units, e.g. to change special parameters of a subsystem, are implemented. Therefore a roblet has only access to units, it does not know anything about a module and a module's implementation of the interface. The whole concept is strictly object-oriented.

By introducing units, the framework is able to generalise access to similar classes of subsystems without loosing access to their special features. Additionally, units introduce a possibility of versioning into the system. If new features are integrated into a module then new units will be introduced. As long as older units are still available, all client applications and their Roblets using these old units still work. This has proved to be of great use since complex applications often consist of dozens of client applications and Roblet-servers. A transition to new units can be accomplished step by step for each client application.

Figure 2 shows a chart of the structure of a Roblet-server.

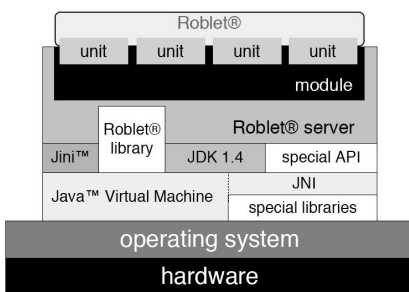


Fig. 2. The chart shows the structure of a Roblet-server and how it hides the hardware from a Roblet.

4.4 Platform Independence

There were several reasons to use Java to implement the concept of Roblet-Technology: First of all, Java virtual machines and compilers are available for a variety of different platforms from embedded system over PDAs to workstation computers. All these different systems can be found in the field of robotics. Since Java source code is compiled into bytecode, the programs can be compiled on any of these systems and executed on another system without change. Besides, Java provides a vast standard library available on all of these systems. The standard libraries include techniques for network communication like RMI or Jini used within the Roblet framework. These well-tested libraries ensure reliable

operation of the framework since they are used in millions of internet applications as well.

For the developer of a client application the view of the system is unified. He does not need any knowledge about the heterogeneous network structure, the differences between operating systems and so on. All he has to be familiar with is Java and programming becomes like programming on one single machine.

In contrast, using other programming languages like C/C++ would require the compilation of the source code for each target machine. Additional libraries, e.g. CORBA, Ice or ACE, are required for network communication, which demand additional knowledge of the programmer. Further on, these libraries may sometimes be only available for a subset of systems present in a robotic scenario. In future, the .NET framework from Microsoft may become an alternative to Java since it also compiles source code into a bytecode first. Nonetheless, to date .NET is only available for Windows platforms. The open-source projects implementing .NET for other platforms do not provide full support yet.

Since Java has no real-time capabilities, programs written within a Roblet are not intended to contain real-time control loops. There exists a specification for a real-time java virtual machine but at present no implementation [16]. The Roblet framework allows less skilled programmers to design and develop robotic applications without in-depth knowledge about the used subsystems. First experiences using the Roblet framework in lectures for graduate students have proved this.

Nonetheless, the developers of modules still must have knowledge about specific technologies they want to use. For example, if we want to encapsulate a C/C++-library that controls a hardware component the module developer will have to write a wrapper for that library using the *Java Native Interface* (JNI). That requires at least knowledge about C/C++ and Java. But, we think in future the number of developers of client applications will be much greater than that of module developers.

5 Roblet application

In this section we will give a short example how software components are used in our Roblet-Framework. We present two Java classes. The first class illustrates how Roblets are sent from the client application to a server. The second class includes the Roblet code which is executed on the server side.

Our example explains how path planning capabilities are integrated into a robotic application. We need the basic Roblet-server from the Roblet-Framework and the path planning module for our robot TASER which is loaded when the server starts running. For our client application we use the client library of the Roblet-Framework. With the client library we send Roblets to the server.

Listing 1.1 shows the Java class `Pathplanner` that can be used for path planning on the client side. An object of this class is created with a string containing the IP address and the port of the Roblet server with the path planning module. Instead of the IP address a hostname can be used. We could start the server on

```

import genRob.genControl.client.Client;
import genRob.genControl.client.Server;

public class Pathplanner
{
    private final Server server;

    public Pathplanner (final Client client ,
                       final String serverName)
    {
        server = client.getServer (serverName);
    }

    public Path plan (final RobotProperties properties ,
                    final Point start ,
                    final Point end)
    {
        return (Path) server.getSlot ().run
            (new PathPlanningRoblet (properties , start , end));
    }
}

```

Listing 1.1. This Java code example shows a class that can be used within a client application to question a Roblet server which provides path planning capabilities. Each time a path is requested a Roblet is send to the server. The Roblet invokes a path planning algorithm on the server and returns the answer to the client application. Some imports as well as appropriate exception handling routines are left out for clarity.

the same machine where we develop the client application. If we use port 8000 the parameter string for the constructor is `localhost:8000`.

Each time the method `plan()` is called an instance of the class `PathPlanningRoblet` is created, marshalled and sent to a slot of the server. A slot is a sandbox environment the server provides. This sandbox is a security layer that restricts the access of the Roblet code to the underlying system. Listing 1.2 shows the Java code of the class `PathPlanningRoblet`. A Roblet has to implement the Java interface `Roblet` which only specifies the method `execute(Robot robot)`. In addition, we implement the interface `Serializable` since the object will be serialised by the client library to send it over a network. On the server the method `execute()` of the Roblet is called.

First, the Roblet code queries the server for path planning capabilities. The path planning module of the server provides an implementation of the `UnitPlanner` which can be received by calling `getUnit()` on the `Robot` parameter. If the path planning module was not loaded the return value of the `getUnit()` call equals `null`. Then, we throw an exception that ends the Roblet. The presented exception handling is very basic to keep the example simple.

If the Roblet gained access to the path planning unit it computes a path from a start to a target point. Internally the path planning algorithm contacts the map server to query the current information about the environment. The start and target point were parameters of the constructor of the Roblet as well as some

```

import org.roblet.Roblet;

public class PathPlannerRoblet
    implements Roblet, Serializable
{
    private final RobotProperties properties;
    private final Point start;
    private final Point end;

    public PathPlanningRoblet (final RobotProperties props,
                               final Point start,
                               final Point end)
    {
        this.properties = props;
        this.start      = start;
        this.end        = end;
    }

    public Object execute (Robot robot)
        throws Exception
    {
        final Planner planner
            = (Planner) robot.getUnit (Planner.class);

        if (planner != null)
            return planner.plan (properties, start, end);
        else
            throw new Exception ("Planner_unit_not_provided.");
    }
}

```

Listing 1.2. The above class implements the Roblet interface. It is instantiated on a client machine. The instance is serialized and send to a server which calls the method `execute()`. The Roblet requests an implementation of the unit `Planner` to calculate an appropriate path for the robot. Some imports as well as appropriate exception handling routines are left out for clarity.

properties of the robot that will drive along the path. These member variables were serialised when we sent the Roblet to the server. Therefore, they are now usable on the server side of this distributed system. If we instantiate other objects which are not present on the servers classpath, the server notifies the client application. Then, the client application will provide the class descriptions to the server.

The path from the start to the goal point is returned to the client application. A `Path` object is return by the `execute()` method. The server sends this object to the client. On the client side the path is returned as the result of the method call `run()` on the `Slot` object.

How the path is computed depends on the properties of the robot and is out of the scope of this simple example. If there is no path that the robot can drive this is encapsulated in the `Path` object and can be queried.

This elementary example gives an insight into the Roblet-Framework. Writing two similar classes for the module that controls the mobile robot gives the developer all that he needs to drive a robot safely in our office environment. On the other side with only four small classes of Java code he creates a distributed system that connects a client application with a robot, a path planning service and a map database. The structure of such a system is visualised in figure 3.

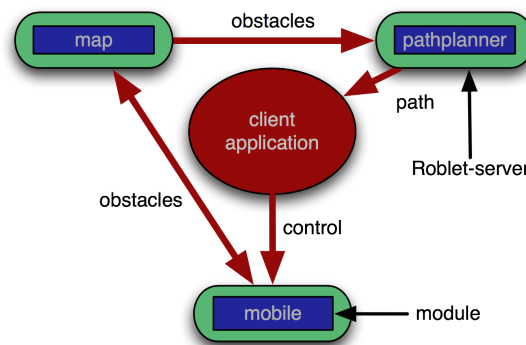


Fig. 3. The flow chart visualises the structure of the distributed system described in appendix 5. A roblet is sent by the client application to the path planner. The pathplanner contacts the server with the map data and receives the obstacles currently stored in the map. After that the pathplanner calculates a path and sends the answer back to the client application. Then, the client application could contact the mobile robot to control its motions. The mobile robot uses the map for localisation purposes. In addition, it sends information about newly detected obstacles to the map server.

A roblet can establish more advanced network communication channels between the server and the client. It may start threads which open sockets or use the Remote Method Invocation provided by Java's standard libraries. A Roblet ends when the method `execute()` finishes. If a thread is started by a Roblet

the thread will stay alive in the slot on the server until the thread ends. Therefore, Roblets may only be needed to distribute code onto a number of different servers when the application starts. These Roblets establish network connections to the client application. Roblets can send Roblets themselves to other servers and thereby create complex distributed system structures. As a result, each client application can create the communication network that is most appropriate. One may choose to use XML data for communication, another may compress all data before transmission and a third one may implement special encryption algorithms to increase security.

6 Applications in Service Robotics

In this section we will describe two applications which emphasise the capabilities of the proposed architecture. The applications show TASER when it accomplishes high-level tasks using a combination of its various components. TASER is a multi-modal service robot located at the institute TAMS of the University of Hamburg.

6.1 Interaction with the environment

The first example is a combination of localisation, planning of paths, object manipulation and interaction where the robot is instructed to operate a light switch. An operator chooses a light switch and commands the robot via speech commands or an interactive dialogue to operate it. The application uses various Roblet-servers shown in figure 1.

First, a Roblet on the Roblet-server for the speech IO informs the client application that a light switch is to be operated by the robot. Then the position of the light switch which is stored as a point of interest in a map is requested from a map server. This Roblet-server encapsulates a database in which map elements like obstacles and points of interest are stored. Multiple applications can get, alter or add map elements of the database concurrently through this Roblet-server. Then, a Roblet is sent to the pathplanning server to get a path to the light switch. The Roblet sends a new Roblet to the robot. There, the new one drives the robot to a suitable position at the light switch, so that the arm can reach the switch. The position of the arm relative to the switch is obtained from a method call to the arm-operations library which is provided by the corresponding Roblet-server. By solving the kinematic chain, the robot computes a position and orientation suitable to operate the switch. After this position has been reached by the robot arm, a Roblet tries to fine-position the manipulator in front of the light switch with the hand camera by visual servoing. A separate Roblet-server for the hand camera provides positioning errors calculated on the observed images. When the arm is centred in front of the switch, an approach move is made by the arm which is force controlled by sensor input of the BarrettHand. The sensors of the hand are precise enough to stop the movement of the arm when the finger touches the switch. In the final step the finger operates the switch. By using a

final movement of individual fingers, even switches like a double-switch can be operated independently.

6.2 Grasping and Transportation

The second example is given by the task of object grasping and transport. The user can advise the robot to fetch and carry objects lying on a table via an interactive dialogue. Each source of information about humans interacting with the robot is encapsulated into its own Roblet-server and can thereby be employed by Roblets. The robot plans its path to the object using the Roblet-server for pathplanning. After reaching a position suitable for object grasping, the robot tries to identify the object by means of object recognition. In case of ambiguities the interaction system is used with other Roblet-servers to resolve the situation. For example, if the robot cannot distinguish objects on the table, it uses the active vision system to recognise pointing gestures and gaze to resolve the position the manipulator of the robot is intended to move to. Additionally, the user can teach the robot new grasping motions and grasps [17].

When the object is successfully recognised, the robot selects a suitable grasp for the object from an internal grasp database and executes it. After grasping the object the robot moves the manipulator back into a safe transporting position. If the transport position has influence on the security outline around the robot, this outline is modified and a path to where the object is to be placed will be calculated based on the new outline. When the final position has been reached the robot will set down the grasped object and is available for new tasks again.

7 Conclusion

The presented software architecture enables the building of high-level applications for service robots using standard components for robots as well as specialised hard- or software. We proved it with the application of our framework to the service robot TASER at the University of Hamburg. The software architecture of the robot based on the Roblet-Technology is a powerful medium for robots. The feature of running client programs as a distributed software offers the possibility to run algorithms which need great computation power on different machines which provide this power. The type of communication, e.g. encrypted or compressed communication, can be changed during runtime. Each client application can use its individual and appropriate type of communication.

The convenience of used proved in several lectures we gave where student were able to build applications for our robot. Some of these application are used in our daily routines when we work with the robot.

One next step will be to implement further software to improve the usability of the robot system and create a toolbox of reusable program parts. In this step the variety of the high-level functions like object grasping and multimodal interaction will be increased. Furthermore, the possibilities of autonomous navigation and map building will be extended.

Another step will be the port of some modules to other robot platforms. With this we can show that the hardware abstraction provided by *units* is reliable, when we do not need changes in the client applications.

Additionally we try to integrate components of other robotic software environments like these of section 2. Our framework will help to connect the various great efforts that are carried out in all these projects.

References

1. A. Orebäck and H.I. Christensen: *Evaluation of Architectures for Mobile Robotics*. Autonomous Robots, vol.14(1), pp. 33-49, Springer, Netherlands, 2003.
2. J. Kramer and M. Scheutz: *Development Environments for Autonomous Mobile Robots: A Survey*. Autonomous Robots, vol. 22(2), pp. 101-132, Springer, Netherlands, 2007.
3. *Workshop on Robot Middleware towards Standards*, International Conference on Intelligent Robots and System (IROS'04), Sendai, Japan, 2004, <http://www.is.aist.go.jp/rt/events/20040928IROS.html>.
4. *Workshop on Principles and Practice of Software Development in Robotics*, IEEE International Conference on Robotics and Automation (ICRA'05), Barcelona, Spain, 2005.
5. Davide Brugali (ed.): *Software engineering for experimental robotics*, Springer tracts in advanced robotics, Springer, Berlin, Germany, 2007.
6. H. Bruyninckx: *Open robot control software: the OROCOS project*, Proceedings of the IEEE 2001 International Conference on Robotics and Automation (ICRA'01), Vol. 3, pp. 2523–2528, Seoul, Korea, 2001, <http://www.orocos.org>.
7. A. Brooks, T. Kaupp, A. Makarenko, A. Orebäck, S. Williams: *Towards Component-Based Robotics*, Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05), Alberta, Canada, 2005, <http://orca-robotics.sourceforge.net>.
8. M. Henning: *A new approach to object-oriented middleware*, Internet Computing, Vol. 4, Nr. 1, pages 66–75, 2004.
9. B.P. Gerkey, R.T. Vaughn, K. Stoy, A. Howard, G.S. Sukhatme, M.J. Mataric: *Most Valuable Player: A Robot Device Server for Distributed Control*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01), pages 1226–1231, Wailea, Hawaii, 2001.
10. C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brousseau, C. Raievsy, M. Lemay, V. Tran: *Code Reusability Tools for Programming Mobile Robots*, Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04), pages 1820–1825, Senda, Japan, 2004.
11. D.C. Schmidt, D.F. Box and T. Suda: *ADAPTIVE — A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment*, Concurrency: Practice and Experience, Vol. 5, Nr. 4, pp 269–286, 1993.
12. N. Karlsson, M.E. Munich, L. Goncalves, J. Ostrowski, E. Di Bernado, P. Pirjanian: *Core Tehnologies for service Robotics*, Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04), Senda, Japan, 2004.
13. Microsoft Robotics Studio: <http://msdn.microsoft.com/robotics/>
14. T. Scherer: *A mobile service robot for automisation of sample taking and sample management in a biotechnological pilot laboratory*, University of Bielefeld, Ph.D Thesis, 2005, <http://bieson.ub.uni-bielefeld.de/volltexte/2005/775/>.

15. D. Westhoff, H. Stanek, T. Scherer, J. Zhang, A. Knoll: *A flexible framework for task-oriented programming of service robots*, Robotik 2004, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, VDI-Berichte (ISBN 3-18-091841-1), Munich, Germany, 2004.
16. The Real-Time Java™ Expert Group: *The Real-Time Specification for Java (RTSJ)*, 2002, <http://rtsj.dev.java.net>.
17. M. Hüser, T. Baier, J. Zhang: *Learning of demonstrated Grasping Skills by stereoscopic tracking of human hand configuration*, To Appear, IEEE International Conference on Robotics and Automation, Orlando, Florida, May 2006.