RoboFrame - A Modular Software Framework for Lightweight Autonomous Robots

Sebastian Petters, Dirk Thomas and Oskar von Stryk

Abstract— **The complexity of autonomous robot systems has increased dramatically in recent years. Besides an increased variety of robots, sensors, actuators, onboard computers and intelligent algorithms, the architecture of the software has gained crucial relevance with respect to the efficiency for adopting a robotic system to new hardware, new software or new tasks. For evaluation of different robot architectures and middleware not only a set of set of evaluation criterias is required. At least equally important for an evaluation is to define a representative set of the boundary conditions, i.e., different types of robots, tasks and scenarios as well as robot programmers. How the different criteria of an evaluation will be weighted also depends on these boundary conditions.**

To address the special needs of heterogeneous teams of autonomous lightweight robots, the software framework RoboFrame has been developed. Its main characteristics are platform independency, modularity and high efficiency. It is also bundled with a library of common components for robot control software, which provides much more support to the robot programmer than a robot middleware system alone.

I. Introduction

The number and types of sensors, actuators and degrees of freedom in todays robot systems is strongly increasing. More complex scenarios, including autonomous operation in a (partially) unknown environment or cooperation between multiple, heterogeneous robots require intelligent algorithms for sensor data processing, localization, motion control, and behavior control. As the link between the robot's hardware (sensors, actuators and onboard computers) and the various high level software, the software architecture gains crucial relevance. A 'good' robot software architecture must offer much more than a pure middleware, namely active support for robot system development. This includes modularity as well as portability and reusability but also support for the robot programmers through simplified interfaces, monitoring and debugging facilities and additional tools like real-time simulation for software testing without robot hardware.

To decide which software architecture may be best suited to meet specific requirements a set of evaluation criterias must be defined. However, there weighting depends on the specific boundary conditions, i.e., the specific robot hardware, the application scenario and task to be solved, and the abilities and requirements of the specific robot programmers. In this paper, we suggest a set of criterias which can be applied as needed to support the decision process to find a matching architecture. The set covers the desired platforms, robot types and control paradigms but also the support of the architecture during the development process. Further on, also a possible, induced overhead and the flexibility of the programming interface is relevant.

S. Petters, D. Thomas and O. von Stryk are with the Simulation, Systems Optimization and Robotics Group, Technische Universität Darmstadt, Darmstadt, Germany {petters, dthomas, stryk}@sim.tu-darmstadt.de

Due to a lack of suitable software architectures for autonomous lightweight systems the platform independent framework RoboFrame [1] has been developed which is also presented in this paper. It has been designed to meet the special needs of systems with challenging dynamical locomotion like small size humanoid robots or unmanned aerial vehicles which impose strong restrictions on the available payload and thus the available CPU power. Due to its modular design, the development of control software for different types of robots with a large variety of reactive and deliberative control paradigms is simplified and can be accomplished in short time. Debugging, monitoring and control of applications is supported by an extendable and also platform independent graphical user interface.

II. Software architectures for robot systems

The architecture of a software describes the components and their interaction within an application. It consists of design decisions which primarily cover aspects of non-functional characteristics like extensibility, reliability and usability. The functional components are not part of the architecture itself but are heavily affected by the architecture.

Modern robot control software uses multiple functional components for, e.g., sensor data processing, behavior control and motion control, each of them may be very complex systems themselves. The main task of the architecture for robot control software is to provide flexible and reliable communication mechanisms for data exchange between the functional parts. This should be realized without restricting the functional components in any way. For todays robot applications, mostly middleware systems are used to accomplish this task.

A software architecture for robot systems should not only provide communication mechanisms like a middleware does, but should additionally offer solutions for common problems in this specific domain. Implementations of useful components should be bundled in a library to speed up the development process by actively supporting the robot programmers.

III. Evaluation of software architectures

To evaluate different software architectures for robot systems, a set of criterias is required which must be rated on the background of specific boundary conditions. These boundary

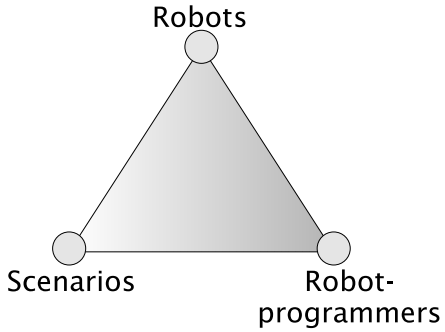


Fig. 1. Boundary conditions for the evaluation of robot software architectures and middleware.

counditions described the specific robots and robot hardware, their specific application scenarios and tasks to be solves, as well as the skills and requirements of the robot programmers (cf. Fig. 1). The weighting of each evaluation criteria depends on the specific boundary conditions.

A. Boundary conditions

1) Robots: Some architectures are restricted to one type of robots, e.g., wheeled, legged or airborne, and their specific characeristics, special types of actuators, sensors, onboard computers and other hardware. For some kind of robots and applications an architecture may be suited very well but may pose large difficulties for another kind.

The robot specific hardware also includes the onboard computing facilities and the *supported operating systems*. Sometimes the choice of an operating system is limited, due to the fact that some drivers for sensors or actuators are only available for some, but not all operating systems. Also, the operating system in use may have to be changed because of the requirement to use new robots or new robot components. If the architecture encourages platform independent programming, an application may easily be ported to a new operating system.

2) Robot programmers: The robot programmers considered here are in charge of developing the high level algorithms and software needed to solve a specific task using specific robots. They are distinguished from the developers of the robots basic hardware, software and software architecture on the one hand and the (possibly untrained) end-user doing some application programming intuitively through interaction with the robots. The software architecture and middleware should enable the robot programmers to concentrate on their main task. For this purpose, architecture and middleware should relieve them from taking care about implementation details of the communication with hardware components, like sensors or actuators, or between modules of the high level algorithms. Furthermore, platform independent programming should be encouraged and supported to enable efficient portability and adaptability. Also support for monitoring, debugging and profiling of all or parts of robot programs is required as well as the ability to efficiently interface with additionals tools like real-time simulation for software testing without robot hardware. These requirements for the robot programmers also depend on the specific robots and tasks to be solved.

3) Scenarios and tasks: Very different requirements for robot software architecture and middleware may result from the many different robot application scenarios and tasks. A few examples are mentioned below.

Industrial robots. Usually, industrial robots are stationary and may therefore be provided with arbitrary computational power and energy supplies. Because of this, efficiency considerations are less important. Also the robot hardware in use is very similar and thus does not require much flexibility from the architecture.

Service robots. Service robots must be to perform household or guidance tasks and to assist human beings in their daily life at home or at work. They mostly operate autonomously and receive orders in a high-level language. Very different actuators and sensors may be possible, thus requiring a high flexibility to prepare the robot for different applications. Also due to a moderate or large payload on wheeled platforms, quite some amount of energy supplies and computational power is available.

Ambient intelligence. In this scenario the robots are part of an environment (in human residences or offices) in which a multitude of distributed and communicating, small intelligent devices is embedded, e.g., [3]. In cooperation with the ambient intelligence, robots shall support people in their daily work or accomplish tasks autonomously or in cooperation with other systems. For this purpose, highly context-aware, adaptive and personalized systems with seamless communication interfaces are required.

Autonomous search and rescue missions. This scenario usually involves teams of heterogeneous robots with very different capabilities which cooperate with each other and with human responce forces and mission managers. Possible applications include autonomous wheel driven and airborne robots with only very limited payload. Efficiency, scalability and low communication overhead are of crucial relevance here. Also an easy integration of different intelligent algorithms and the flexibility for different control paradigms is important.

Deep space exploration. To explore deep space or other planets, reliability and increasing autonomy are relevant. Restricted power supplies requires efficient software and algorithms.

Robot soccer. In robot soccer, usually homogeneous teams of robots have to accomplish complex tasks in real time in a dynamically changing environment. To allow to focus on the development of algorithms, the robot programmer should be able to concentrate on high level algorithms for perception, planning and control and be relieved from focusing on lower level tasks. Efficiency and tools for debugging and monitoring are also of major relevance.

B. Evaluation criteria

If a set of boundary conditions of robot, scenario and programmer has been defined then specific evaluation criteria

of different architectures and middleware can be applied, for example:

Control paradigms. Some architectures explicitly support only one of the reactive or deliberative control paradigms and thus do not offer a choice. With this approach it is not possible to investigate different control paradigms for a set of boundary conditions or even change it during the lifecycle of an application.

Support for the developers. A software architecture can actively support the developers due to ease of learning and reuseability of components. The programming interface should encapsulate implementation details at different levels and thus expose only the abstraction level needed for implementing a specific task. The architecture can also provide solutions for common problems in robot control software for a specific set of boundary conditions, but should also be flexible enough to allow other solutions. Depending on the used programming languages, the architecture may also provide tools (i.e. editors, integrated development environments) to create applications.

Testing, debugging and monitoring. For debugging and monitoring during development or operation of high level robot algorithms the architecture should provide an interface to access the relevant information. This contains the actual data of the algorithms but also the metadata about the current state of the robot program itself. For debugging purposes it is also convenient if the data arising in an example application can be recorded, modified and replayed. This allows testing and comparison of algorithms with the same reproducible input data. Suitable interfaces with additional tools like a real-time simulator for testing and debugging of robot programs or tools for profiling robot programs are also required.

Overhead. For systems with restricted payload and resulting strong limitations in power supplies and onboard computational power the overhead induced through the use of a middleware should be low. This requires highly efficient communication mechanisms and modern software engineering techniques to provide this also to the parts of the robot software contributed from the robot programmers.

Scalability. The architecture should be able to scale with growing complexity of the boundary conditions and high level algorithms required. This can be caused by more complex algorithms or by additional sensors and actuators. E.g., it should be possible to enlarge the number of robots in a team without adding more overhead for communication.

Software quality. As a base for multiple, different applications, the architecture itself should have a high software quality in means of ISO 9126-1 [2]. This standard defines criteria for evaluating the quality of software with various characteristics, i.e., functionality, usability, reliability and maintainability, and divides them into specific properties (see Table I). Even if measuring these properties is not an easy task, the standard also suggests metrics for calculation.

Availability. For further development it may be important if the source code of the software architecture is available or not. Closed source software involves the risk of lacking flexibility, e.g., if required features can not be added.

TABLE I
SOFTWARE CHARACTERISTICS AND PROPERTIES (ISO 9126-1)

Characteristics	Properties
Functionality	Suitability Accuracy Interoperability Compliance Security
Reliability	Maturity Recoverability Fault Tolerance
Usability	Learnability Understandability Operability
Efficiency	Time Behaviour Resource Behaviour
Maintainability	Stability Analyzability Changeability Testability
Portability	Installability Replaceability Adaptability

IV. EXISTING ARCHITECTURES

During the last decades, various robot software architectures have been developed, covering the special requirements of different scenarios. Three groups of architectures may be distinguished.

A. Middleware based architectures

These architectures provide a communication layer for multi purpose robot control software. The targeted robot systems are mainly equipped with high performance processing units using standard personal computers.

Miro [4] is a middleware for robot systems based on CORBA and is designed for multi processor systems. This is motivated by the observation that larger robot systems are realized as a network of computers, actuators and sensors which are equipped with own computational units.

Microsoft Robotics Studio (MSRS) [5] is a service based middleware not only restricted to but focused on robot applications. It is based on the Microsoft Compact Framework, uses HTTP as communication protocol and allows implementation of applications in all .NET programming languages.

B. Robot device interfaces

The main goal of these architectures is to provide access to sensors and actuators over the network. The communication mechanisms do not support the flexibility of the middleware systems mentioned above.

ORiN [6] provides an integrated interface to access the devices on the network. It abstracts from the device data in a way beyond the differences of manufacturers to ease targeting multi vendor environment.

Player [7] is a client / server architecture which allows access to a variety sensors over the network. Applications can be written in any programming language, requires only

network access to the sensors and thus allows the implementation of distributed or collaborative applications.

CLARAty [8] represents a framework which allows access to common sensors and actuators. Its main goal is to promote the reusability of software components. It is bundled with a vision processing library.

Orca [9] is a component-based system, which uses ICE as a middleware. It defines a set of commonly-used interfaces and provides libraries with a high-level convenient API.

C. Integrated robot control software architectures

The integrated robot control software architectures are mostly targeted to a specific type of application.

URBI [10] is a behavior interface based on a client/server architecture to control any robot or complex system. It is based on a scripted language, which can be used to connect all components – written in any other language – in a parallel and event-driven way. It furthermore provides abstractions to manipulate the data and control the flow of execution between these heterogeneous, distributed objects.

The Orocos project [11] provides a runtime environment to implement real-time and non real-time control systems. Furthermore it contains libraries to handle the modeling and computation of kinematic chains and inference in Dynamic Bayesian Networks (e.g. Kalman filters).

Saphira [12] is an architecture for robot perception, sensor data interpretation, map building and reactive planning focused on the needs of wheeled robots. Its main components are a fuzzy control system and a behavior sequencer.

TeamBots [13] is a Java package used in research and teaching to implement reactive systems which are able to learn. Due to the usage of Java it runs on many different platforms and includes a graphical user interface for simulation purposes.

V. RoboFrame

None of the existing architectures fully covers the special needs of autonomous lightweight robots. These robots are characterized by challenges in their dynamical locomotion and stability like unmanned aerial or small and medium size humanoid robots. Their payload is quite restricted which severely restricts the computational power and power supplies available onboard. To meet these special needs, the object-oriented software RoboFrame has been developed at the authors group. *RoboFrame* is written in ANSI C++ and was designed as a software framework using modern software engineering technologies. The approach of a framework has been chosen to enable and facilitate reuse of the architecture in many different sets of boundary conditions. Each new application extends RoboFrame at predefined extension points, more precisely by means of extending (abstract) base classes. In contrast to a library, the call direction is reversed, the application specific implementation is called by the framework. RoboFrame consists of two parts: *RoboApp* is the base for any high level software running on the robot, while *RoboGui* is the base for a graphical user interface.

A. RoboApp

RoboApp has a platform abstraction layer which encapsulates all platform specific calls made to the operation system. The system dependent abstraction layer handles threading, synchronization, network and file system access, serial port communication and various other functionality. Currently the following platforms are actively supported: Linux, FreeBSD, Windows 2000, Windows XP and Windows CE 5. RoboApp has been designed to run with very low overhead to allow deployment on systems with very low computational capacity.

An application extends RoboApp by a number of user defined modules for the different tasks. Each module must not have any dependency with the other modules, instead the modules specify the demanded and provided type of data in a descriptive manner. This description is not done by an external configuration file, but in the definition of the module itself. Each type of data, which is communicated between modules, is specified as a class, which is capable of serializing itself to and deserializing itself from a byte stream.

Beside the described message based communication RoboApp also provides a black board communication approach, which is more suitable if large data structures are modified incrementally (for example mapping tasks).

A concrete robot control software creates any number of separate threads and distributes the modules over them. The execution of the modules can be based on multiple incidents: a) after a fixed timer, b) after a delay since the last execution, c) after a module receiving new input data and requesting immediate execution. Multiple modules can be added to a single thread, where they always are being executed sequentially, or distributed across multiple threads to meet the requirements from the desired scenario. Using these definitions of the application the exact timing for the execution of the threads and therewith the modules can be controlled by the framework.

Besides the thread layout the application defines unique identifiers to link a data source of one module with a data sink from another module. That means a unique identifier is used to tag a semantically message of a specific data type, for example, the raw YUV-image of a camera of the robot.

Using these descriptive informations the router, the component for dispatching all exchanged data, can dynamically route the messages between threads with their modules. The routing is not limited to local threads, but other connectors can be added to the router. This dynamic message exchange enables a very high flexibility, which adapts during runtime automatically based on the currently available data senders and receivers. The connectors demand and provide message just as modules in threads do. A common connector could be a socket connection to another application or the graphical user interface. In Fig. 2 the router is dispatching the provided data from the modules in thread A both to the module in thread B and over the socket connector to the modules in an other application or to a graphical user interface.

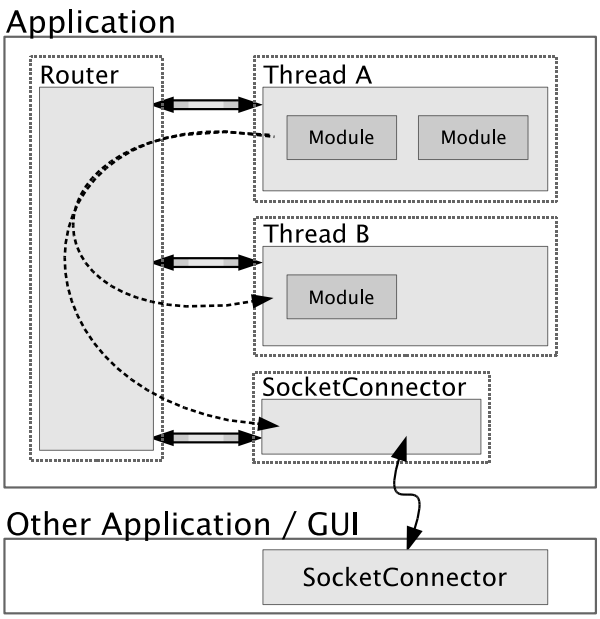


Fig. 2. Routing messages between threads or to other applications.

A module sending a specific type of data is not concerned with the details, where the receiving module is located. The receiver could be in the same thread, in another thread in the same application or even in a different application running on another computer. Yet the framework allows the execution of module as if they were running in a single threaded application, because it cares for the necessary locking and synchronization.

However a module can query if a provided type of data is actually demanded by any other module. This information can be used to skip further processing when the result is not really required, which allows the algorithms to be adaptable to the specific needs at runtime. This approach allows a very low coupling of the different high level software modules defined and implemented by the robot programmer.

The modules access the demanded data using a buffer structure, with operates as a proxy of the real data objects as depicted in Fig. 3. Thereby the passing of messages between modules running in the same thread is very fast, since the costs for copying messages can be avoided. This approach reduces the overhead on a bare minimum despite the logical separation of the modules.

Due to the flexibility of the data exchange mechanism provided by the framework, the control architecture is not limited to a special behavior control paradigm. Arbitrary structures like reactive or a hierarchical-deliberative paradigms can be realized, depending on the individual connections of the user defined modules for sensor data acquisition and processing, behavior control and motion generation.

This approach also encourages the development of modules for the same functionality (e.g., object recognition or self localization) but with alternative algorithms since replacing one module with another and comparing their results is very easy and can even be done during runtime.

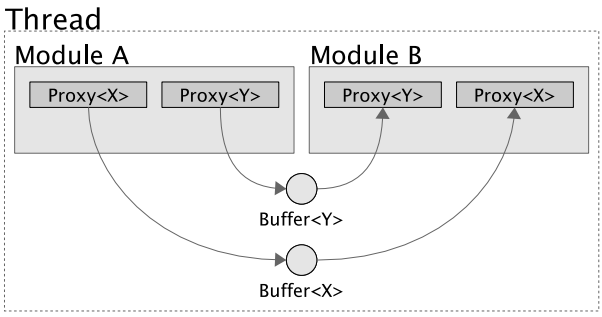


Fig. 3. Modules access exchanged data over a proxy to share data references inside of threads.

B. RoboGui

RoboGui is based on the GUI toolkit Qt[1] from Trolltech. Thus the graphical user interface (GUI) can be build on all major platforms. The GUI can connect to multiple software applications simultaneously and is used for debugging, monitoring and controling.

Since RoboGui uses the same messaging subsystem as RoboApp any message sent within the software application can be requested. The framework provides some generic dialogs to record and replay any kind of messages, to modify the timing and priority parameters of the threads and to enable and disable any module at runtime and to display logging messages from the remote application.

The dialogs in RoboGui are what the modules are in RoboApp. The graphical user interface can be extended with user defined dialogs as demonstrated in Fig. 4. Like the modules a dialog can demand and provide specific type of data using the unique identifier.

For example, the meta information provided by the messaging subsystem can be used to visualize some internal data from a module in a custom dialog, which are generated and provided in the module only when the software application is connected to the graphical user interface and the specific dialog is currently open.

VI. Components and Tools

Supplementary to the basic infrastructure RoboFrame provides tools and components to make the life of the robot program developer easier.

RoboApp includes data structures for a variant type and a hierarchy of named variant parameters. By means of them modules can easily define a set of parameters, which should be used to configure the algorithm. These parameters can be loaded from a configuration file and also altered during runtime using a generic dialog in the GUI, which speeds up the evaluation of various parameters of the module.

Moreover a logging mechanism is provided which is similar to Log4J[2]. Despite the usage of plenty debug messages throughout the source code, the performance is unharmed

[1]Qt Homepage: `http://trolltech.com/products/qt/`
[2]Log4J Homepage: `http://www.log4j.org`

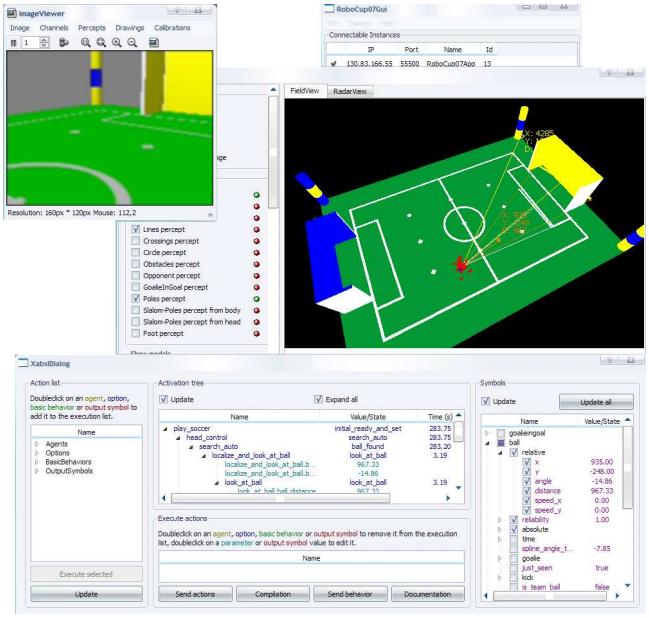


Fig. 4. Example of application of RoboGui showing several application specific dialogs for monitoring the current world model and state of the robot behavior control.

Fig. 5. Selected robot systems running applications based on RoboFrame in the authors research group. Upper row: humanoid robot Lara (left) actuated by artificial shape memory alloy muscles, several humanoid, legged and wheeled robots (middle), autonomous humanoid robot Bruno (right). Lower row (from left): robotic boot, robotic offroad vehicle, new four-legged robot.

due to configurable selective output capabilities. These logging messages can also be reported to a remotely connected graphical user interface.

Another example, which supports profiling a robot software application, is the chronometer hierarchy, which measures the timing of code blocks. It is not intended to replace full powered profiling tools, but gives the developer a rough overview for which software modules which part of the CPU time is spent. For threads and modules these chronometers are added automatically, but modules can use additional chronometers to measure different aspects of their algorithms. The measuring of any chronometer can be enabled in a dialog of the GUI, which displays the time, average time, frequency etc.

RoboGui provides a dialog which is capable of recording any kind of exchanged messages and saves them in a log file. These messages can be played-back or reproduced step by step for example to debug a module with exactly the same input data again and again.

Additional tools and high level packages: Beside these small components integrated into the framework, there are also a couple of additional tools and libraries for developing robot programs consisting of modules and corresponding dialogs, which are quite common and used in many application.

One of the common packages is dedicated for developing autonomous robot behavior decisions based on provided informations using hierarchical state machines. The module integrates the engine of the Extensible Agent Behavior Specification Language XABSL [14] for executing the implemented behavior. Beside that a dialog enables remote debugging of the hierarchical state machines and to inspect the input and output data. With the GUI, specific state machines can be triggered manually and even a new behavior can be uploaded and executed during runtime.

Beside this their is a number of packages providing algorithms and monitoring dialogs in the domains of image processing, self localization using stochastic methods and world modeling. A bunch of modules exist which use hardware specific drivers to read sensor data from devices like YUV-cameras, RGB-cameras, laser range scanners etc. Such building blocks are realized by separate libraries so that they can easily be plugged into any application using RoboFrame.

VII. Applications of RoboFrame

The described software architecture RoboFrame has been and is currently being used in multiple scenarios with different heterogeneous robots (see Fig. 5).

A. RoboCup - Humanoid Robot League

A special scenario for humanoid robots in a dynamically changing environment is given in RoboCup[3], the annually held world wide competition for cooperating teams of autonomous soccer robots in different leagues. Because of its publicly defined setting and wide acceptance it may serve as an example for a set of specific boundary conditions as discussed in Sect. III-A. Our team, the Darmstadt Dribblers[4], is participating in the humanoid robot league with our 55cm tall humanoid robots (Bruno [15], Fig. 5). The fully autonomous robot is actuated by 21 rotary electric motors and equiped with onboard cameras and inertial sensors. In order to enable a high performance in bipedal locomotion with respect to walking speed and postural stability the payload for onboard computer and batteries is highly restricted. Since the life cycle of recent hardware components (servo motors, cameras, inertial sensors, onboard computer) is short the main components of the robot's hardware are heavily modified or replaced with newer models approximately every

[3]RoboCup Homepage: `http://www.robocup.org`
[4]Darmstadt Dribblers Homepage: `http://www.dribblers.de`

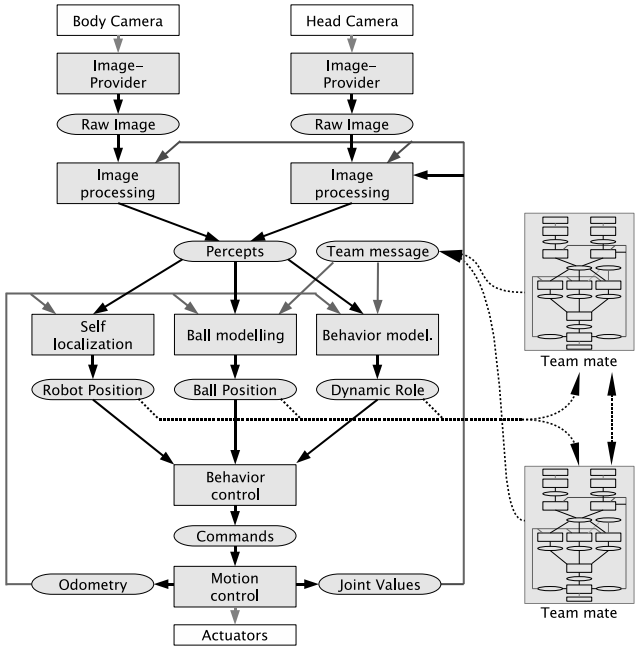


Fig. 6. Application layout used at RoboCup 2007 (rectangles describe modules, ellipses represent exchanged messages).

two years. The main parts of the high level robot algorithms are developed by graduate students in practical courses and theses.

Due to these constraints the following criteria are crucial in this scenario:

- the demand for a low overhead is directly implicated by the very restricted payload capabilities,
- the short life cycle of hardware components makes an architecture necessary with can easily integrate new types of robots and components and support different operating systems to enable the reuse of already developed modules of high level algorithms,
- the duration of the employment of the programmers requires that they can quickly understand the architecture and use the provided interfaces.

Based on the flexible communication methods it was not difficult to establish high-level abilities like a team communication between the teammates, which enables the fusion of the individual world model informations and can be used for dynamic role assignment between team mates during a soccer game.

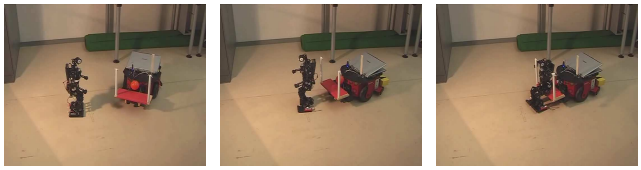
In Fig. 6 the internal structure of the modules and their exchanged messages are shown to give an example of a real world application layout based on RoboFrame.

B. The new four-legged robot

A new, open and modular platform for research in autonomous, four-legged robots[5] is being developed at the authors group in cooperation with Hajime Research Institut[6].

[5] `http://www.thenewrobot.com`

[6] Hajime Research Institut `http://www.hajimerobot.co.jp`



(a) Communication about robot position (b) Autonomous boarding (c) Completed boarding

Fig. 7. Excerpt of cooperation of autonomous humanoid robot and wheeled robot for joint mission completion [16].

This four legged robot is designed to offer a large variety of research and development opportunities to the advanced robot programmer which can hardly be found in any other robot platform currently available. Developing the software for a demo application for the first robot prototype consisting of different walking styles which are controled using LAN or WLAN and a remote PC with a joystick was just a matter of days due to the use of the described software architecture and reusable components. A PC104 under Windows CE has been used as onboard computer.

C. Cooperation of heterogeneous robots

Another project deals with the cooperation of strongly heterogeneous, autonomous robots. In a case study, an autonomous humanoid robot with a Pocket PC under Windows CE as onboard computer is cooperating with a wheeled Pioneer 2DX robot with an AMD K6 II 400 MHz processor under Linux as onboard computer (cf. Fig. 7, [16]). The subtasks which are required for mission completion are dynamically assigned based on the capabilities, which are necessary and available to fulfill the job. To our knowledge this case study is the first successful cooperation between an autonomous humanoid and a wheeled robot.

D. Real-time HW/SW-in-the-loop simulation

The framework itself does not feature any kind of simulation. However, it can be interfaced with a large variety of robot simulators, e.g., a humanoid robot simulator based on a multi robot simulation framework MuRoSimF [17]. To interface an application based on RoboFrame with the simulation the already introduced communication methods are used.

Instead of having physical hardware components providing sensor information other modules are used instead to provide corresponding data but from simulated hardware components. The same technique is used for data which is used to control actuators etc. For the modules which are handling the processing the exchange of the data source is fully transparent since the type of communicated data remains unaffected.

Another ongoing project is to evaluate different approaches to humanoid robot behavior control using humanoid robots simulated in real-time. Therefore RoboFrame and MuRoSimF were chosen because of the quick learnability and flexible exchange of modules, which allows rapid prototyping and comparison of different algorithms.

VIII. Summary

For evaluation of software architectures and middleware for autonomous robots it is proposed in this paper to use a set of criteria but on the background of boundary conditions consisting of specific robots, scenarios, tasks and robot programmers. Therefore, there will not be a single set of criteria and boundary conditions that fits all relevant cases. However, representative sets of these yet have to be developed and agreed upon.

Furthermore, the platform independent software framework RoboFrame has been presented, which addresses the special needs of heterogeneous teams of lightweight autonomous robots. The successful application of RoboFrame in a variety of different scenarios for different robots demonstrated its efficiency and flexibility. On top of RoboFrame, a library of common components for robot control software has been implemented, which simplifies the development of new applications and actively supports the robot programmers. A plain middleware system alone does not offer these valuable feature to the robot programmer.

References

[1] M. Friedmann, J. Kiener, S. Petters, D. Thomas, O. von Stryk: *Modular software architecture for teams of cooperating, heterogeneous robots*, In Proc. IEEE Intl. Conf. on Robotics and Biomimetics (ROBIO), Kunming, China, Dec. 17-20, 2006, pp. 613-618

[2] ISO Standards. Software engineering - Product quality - ISO/IEC 9126-1. International Organization for Standardization, 2001

[3] B. Gates: *A Robot in Every Home*, Scientific American, January 2007, http://www.sciam.com/article.cfm?chanID=sa006&colID=1&articleID=9312A198-E7F2-99DF-31DA639D6C4BA567

[4] H. Utz, S. Sablatnög, S. Enderle, and G. K. Kraetzschmar: *Miro middleware for mobile robot applications*, IEEE Trans. on Robotics and Automation, vol. 18, no. 4, pp. 493497, August 2002

[5] Microsoft, Microsoft robotics studio, http://msdn2.microsoft.com/en-us/robotics/default.aspx

[6] M. Mizukawa, H. Matsuka, T. Koyama, T. Inukai, A. Nodad, H. Tezuka, Y. Noguch, and N. Otera: *Orin: Open robot interface for the network*, In SICE, pages 925928, IEEE Press, 2002

[7] B. Gerkey, R.T. Vaughan, and A. Howard: *The playerstage project: Tools for multi-robot and distributed sensor systems*, In International Conference on Advanced Robotics (ICAR), pages 317323. IEEE Press, 2003

[8] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim: *CLARAty: An architecture for reusable robotic software*, In SPIE Aerosense Conference, Orlando, Florida, April 2003

[9] A. Makarenko, A. Brooks, and T. Kaupp: *Orca: Components for robotics*, In International Conference on Intelligent Robots and Systems (IROS), pages 163168. IEEE Press, 2006

[10] J.-C. Baillie: *Urbi: towards a universal robotic low-level programming language*, In International Conference on Intelligent Robots and Systems (IROS), pages 820825, IEEE Press, 2005

[11] H. Bruyninckx: *Open robot control software: the Orocos project*, In: IEEE International Conference on Robotics and Automation (ICRA), pages 25232528. IEEE Press, 2001

[12] K. Konolige and K. Myers: *The Saphira Architecture for Autonomous Mobile Robots*, SRI International, 1996

[13] T. Balch: TeamBots - www.teambots.org, 200

[14] M. Lötzsch, M. Risler and M. Jüngel: *XABSL - A pragmatic approach to behavior engineering*, In: Proc. IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), pp. 5124-5129, October 9-15, 2006

[15] M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, O. von Stryk: *Versatile, high-quality motions and behavior control of humanoid soccer robots*, In: IEEE/RAS International Conference on Humanoid Robots (Humanoids), Workshop on Humanoid Soccer Robots, pp. 9-16, Dec. 2006, Genua, Italy

[16] J. Kiener and O. von Stryk: *Cooperation of Heterogeneous, Autonomous Robots: A Case Study of Humanoid and Wheeled Robots*, In: IEEE Intl. Conf. on Intelligent Robots and Systems (IROS), 29 October - 2 November, San Diego, 2007, to appear

[17] M. Friedmann, K. Petersen, O. von Stryk: *Tailored Real-Time Simulation for Teams of Humanoid Robots*, In: International RoboCup Symposium, July 9-10, Atlanta (USA), 2007 (Robot World Cup Soccer Games and Conferences)