Universal Web Interfaces for Robot Control Frameworks

Jan Koch, Max Reichardt, Karsten Berns

Abstract— **Developers and end-users have to interface robotic systems for control and feedback. Such systems are typically co-engineered with their graphical user interfaces. In the past, a vast community of researchers has addressed issues of generality, deployment, usability, and re-usability of user interfaces. However, the support for creating graphical user interfaces in recent robotic frameworks is limited. In particular, there is typically no support for web-based teleoperation. In this work, we propose a new Java-based editor with a plugin architecture for GUI elements and communication ports. Special focus is laid on platform-independent design, easy extensibility, connectivity to different robotic frameworks, usability and deployment. The tool offers convenient creation of graphical user interfaces and can publish them over the web - making them accessible from any Java-enabled web browser.**

I. Motivation

Developers and operators of technical systems need convenient access to control interfaces and state information. In the case of - but not limited to - mobile robotic systems, this interfacing software is deployed on external networked computers or mobile devices that carry the respective graphical user interface (GUI). Usually, during development a GUI is subject to multiple changes due to ongoing feature integration and debugging. Also, different human operators show different preferences in GUI design. This leads to the demand for editor tools capable of conveniently creating and editing GUIs. If well-designed, the GUI is free of system logic and is a separate component from the system to be controlled. Developers of robotic frameworks have already spent varying amounts of effort to provide such tooling. However, these tools are specifically designed to work with their respective frameworks and have limitations regarding flexibility and deployment. In section I-A, we discuss the capabilities of recent robotic development frameworks regarding these aspects.

In this work, we propose a new Java-based GUI tool addressing the following areas:

- editor for graphical user interfaces
- independence from specific (robotic) applications and frameworks
- extensible plugin architecture
- web-based GUI deployment
- robust handling of fluctuating connection link quality

Jan Koch, Max Reichardt and Karsten Berns are with the Robotics Research Lab, Department of Computer Science, University of Kaiserslautern, Gottlieb-Daimler-Straße, 67663 Kaiserslautern, Germany {koch, m_reicha, berns}@informatik.uni-kl.de

A short look at promising application scenarios for mobile robotics shows the need for remote teleoperation. Examples include space robotics, security and surveillance, telepresence in the context of health care and assisted living, as well as robots that operate in hazardous environments such as emergency areas or industrial sites.

Today, a lot of simple devices offer built-in web servers with, however, specifically designed web interfaces. In most cases, look-and-feel is rather pragmatic and deployment is bound to the application host, as interfaces are designed for occasional parameter changes and not for continuous user interaction.

However - especially in domains where complex and rather specific software backends are in place - work has to be kept up on re-usability, extensibility and GUI editing features.

In section I-B, we give some overview on upcoming robotic products that are teleoperable over the WWW, and briefly introduce our own application scenarios. The system design of our new tool regarding plugin architecture, communication concept, GUI widgets and deployment is explained in section II. Finally, we present use cases and experiences in section III.

A. GUIs in popular robotic frameworks

Most of today's robot development frameworks provide graphical tools for management, visualization and for interaction with robots. There are in fact frameworks which already support aspects discussed in the introduction. However, there is currently no framework with a comprehensive solution. Therefore, project- or application-specific user interfaces are often implemented using common programming languages.

The popular and widespread Player Project [1], [2] provides several independent graphical tools such as *playerv* for sensor visualization, *playercam* for displaying video streams from cameras, or *playerjoy* for teleoperation. However, to our knowledge, none of the distinct features discussed in the introduction are directly supported.

At the end of 2006, Microsoft released Robotics Studio[1]. In Robotics Studio, systems are composed of services (DSS) which are executed inside of *DSS nodes*. These DSS nodes provide a web interface for managing and monitoring the services they contain. Robotics Studio is one of very few frameworks with web support. However, this web interface does not seem to be designed for teleoperation, since the content of the web pages is static. Apart from that, GUIs

[1] http://msdn.microsoft.com/robotics/

can be created using the GUI editor in the Visual Studio IDEs. However, this requires programming skills and costs significantly more effort than our approach. Furthermore, Robotics Studio's XML-based communication mechanism appears unsuitable in the context of web-based teleoperation due to its increased bandwidth requirements compared to binary encoding.

The MCA2 framework [3], [4] contains a full-featured GUI editor called *MCAGUI* and the runtime monitor *MCABrowser* - both established and valuable tools in our development process. However, it requires a platform-specific backend installed on the PC on which it is deployed, and has no web support. Furthermore, MCAGUI is strongly tied to the MCA2 framework regarding communication, and is not designed for unstable network connections. The editor functionalities are quite distinct and sophisticated. The proposed tool was inspired by MCAGUI in several areas, as can be seen in section II.

The Orca framework [5] provides some graphical tools for the management of components, visualization and interaction with robots - *OrcaView2D* being the most interesting one in our context. Instead of placing them on a canvas, GUI elements can be interactively added to a two-dimensional world view. Thus, there are mainly GUI elements whose semantics are related to maps - for instance *LaserScanner2D* displaying laser scanner data. OrcaView2D is designed for extensibility, allowing further GUI elements to be added using a plugin mechanism. Concerning interoperability, Orca has support for the Player Project.

Another framework with interesting properties regarding user interfaces is ROCI [6]. It provides a web interface for accessing and monitoring the internals of a robotic application. Furthermore, it contains the *ROCI Browser*, which - among other things - allows controlling robots by specifying scripts in ROCIScript. ROCIScript is close to the natural language, and processes commands such as *Search Room* or *Go To WayPoint* [6].

Tools supporting multiple robotic frameworks as well as interoperability between different frameworks are scarce. MARIE [7], however, is a major approach to supporting and integrating components from different robotic frameworks.

B. Web-based Teleoperation

GUI solutions are usually platform-dependent, and proprietary software needs to be installed or even compiled on the host controlling the robotic system. Nowadays, however, web standards have reached a state that allows graphically rich applications to run in a web browser on almost any system without installing additional software. Therefore, it seems ideal to provide the possibility of controlling a robot directly using a web browser.

a) Recent Applications: Since the beginning of the Internet, the robotics community started investigating the benefits of web-based robot teleoperation [8]. Since then, various projects have shown possibilities such as, for instance, processing collaborative tasks [9]. In mobile robotics, we see companies from different domains on the market that already propagate telepresence and teleoperation over the Internet. On the consumer market, there are, for example, upcoming products such as Spykee[2], Rovio from WoWee[3] or ConnectR[4]. In the professional area, applications like the telemedicine system from Intouch[5] are available. At least the consumer systems are designed for a direct remote control without any intelligence for autonomous control within the robot. Some industrial robots can also be accessed using the Web. For instance, the manufacturer KUKA offers a software solution with an integrated HTTP server that can be used for off-site diagnosis and service (KUKA Teleservice)[6].

b) Assisted Living Scenario: Telepresence in assisted living or medical scenarios is currently being investigated by numerous groups [10], [11]. During our recent projects in the assisted living domain, we implemented slim mobile systems for audiovisual situation assessment and emergency detection. With the proposed solution, the mobile robots' web-interface is made accessible to first responders and family members when the intelligent environment of our home automation detects problematic conditions [12], [13]. However, the side-effects which arise with system control over the Internet need to be taken into account. As already stated in [14], limited bandwidth and unpredictable delays, in addition to different protocol stacks for direct remote control, can cause problems. We therefore prefer semi-autonomous control in terms of goal-point navigation. Operators send the robot navigation orders by clicking into a building outline as explained in section III-A.

II. System Architecture

Since extensibility is a major design goal, a simple plugin mechanism is an important part of the overall architecture. Plugins are packages that may contain GUI widgets or interfaces to either hardware devices (drivers) or to other robotic frameworks (communication). Generally, our editor is not tied to a specific framework. Rather, support for specific frameworks can be added or removed by simply adding or removing the respective plugins. This way, the set of supported frameworks and hardware can be completely customized. Fig. 1 illustrates the central architecture of our implementation.

Creation of further GUI widgets and framework interfaces is designed to be simple. Our fully-functional CheckBox widget, for instance, is implemented with as little as thirty lines of code. Widgets are Java classes that extend a `Widget` base class, while interfaces implement a `RobotInterface` Java interface. Specifying how such interfaces should look, and of which elements they should consist, is a critical aspect in the design. Generally, the concept needs to be sufficiently abstract, generic and flexible so that a broad range of existing frameworks and hardware can be integrated in a uniform way. On the other hand, it needs

[2] `http://www.spykeeworld.com`
[3] `http://www.wowwee.com`
[4] `http://www.irobot.com`
[5] `http://www.intouchhealth.com`
[6] `http://www.kuka.com`

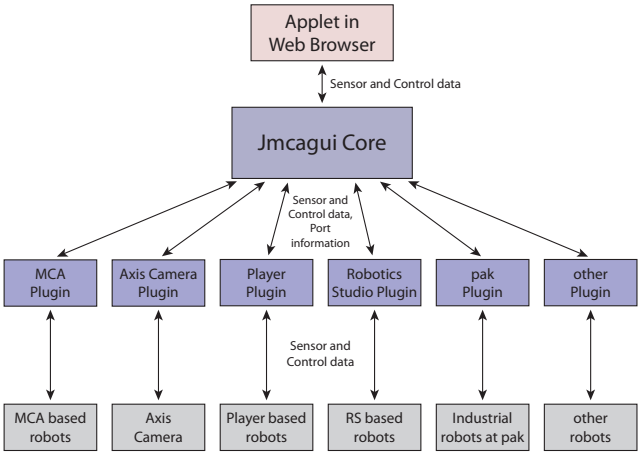

Fig. 1. Overall System Structure

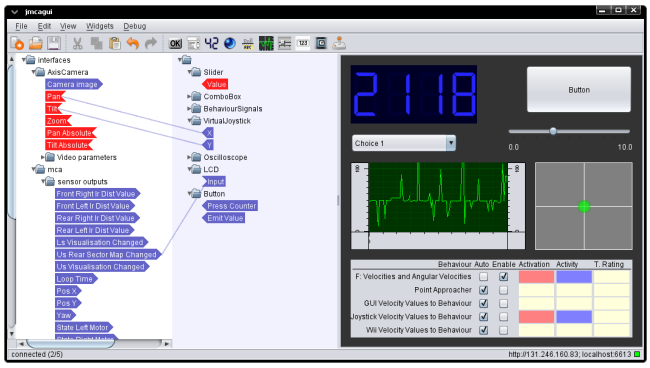

Fig. 2. Editor with connection sidebar

to be simple so that such interfaces can be implemented with reasonable effort. Our approach is presented and discussed in section II-C.

Altogether, our software can be deployed in three different variants: The editor in standalone mode, a server variant, and an applet variant that will run in a web browser (see section II-G). We apply the MVC design pattern [15] throughout our design. The variants share approximately 80% of the code, thus resulting in a small code base and keeping maintenance effort low. For the different variants, only a small set of Controller and View classes are interchanged. Notably, we actually use the same binary with different parameters for the three cases. Using MVC, we also have a clean, separate data model which can be conveniently saved and loaded using common automatic serialization mechanisms.

A. GUI Editor and Widgets

The GUI editor is designed as scalable canvas where GUI widgets may be freely arranged. The editor offers a drag-and-drop mechanism for connecting widgets with respective port interfaces. In fig. 2 we show some exemplary widgets arbitrarily arranged together with the "connection sidebar" on the left. The selected widget connector is highlighted (red), as are all ports it can be connected to.

Numerous widgets are available, including buttons, sliders, combo boxes, check boxes, labels, picture boxes, input fields, LCD-style numeric displays, or status LEDs. There is an Oscilloscope, a Text Editor, the *Virtual Joystick* widget which is often used for controlling robots, and the *Multiple Bars* widget that shows a histogram from multiple inputs. Furthermore, there is a more complex *Behaviour Signals* widget which is rather specific to MCA2 and allows displaying and modifying activation levels of behaviours.

Videos can be rendered and optionally scaled using a *Video Renderer* widget. The most complex widget to date is the *Geometry Renderer* (400 LOC). It can render an arbitrary number of objects implementing a `Paintable` interface. This interface is implemented by many inputs, including the data from our sonar and laser scanners, but also by our bitmaps and SVG vector graphics classes. We use it mainly for displaying 2D maps. An arbitrary number of predefined objects can be added to a map and moved through numerical input ports. Regarding navigation: panning, zooming, rotating, and tracking of objects is supported.

For maximum platform independence, our editor is implemented in Java without using any native libraries. Hence, Java applets are used to display the GUI in any capable web browser. GUIs are saved in a single Zip file that contains all the relevant resources, for instance embedded image files and maps. The GUI structure is encoded in XML.

B. Look and Feel

Generally, usability of our editor is designed to be similar to popular office and painting applications. Features include an undo/redo buffer, tabs, right-click menus in every area of the screen, and manipulation of multiple widgets simultaneously - moving and resizing them as well as changing their parameters. Notably, changing parameters also works with sets of very different widgets. The system analyses which parameters they have in common, and generates and arranges appropriate dialogues at runtime. GUIs in the editor can be edited and operated at the same time.

C. Ports and Communication

To support arbitrary frameworks, it is critical to provide a general infrastructure that enables exposing the different asynchronous interfaces in a simple and uniform way. Ideally, a GUI's widgets can be simply connected to different elements in these interfaces. Generally, our solution is an abstraction of what can be found in the MCA2 framework: every interface provides channels or *ports* of either incoming or outgoing data. Somewhat similar approaches can be found in [16], [6], [7] and LabView[7]. Microsoft Robotics Studio also uses the term *port* in its CCR (Concurrency and Coordination Runtime). These ports work in a similar way, but in Robotics Studio they are used to implement object-oriented interfaces in an asynchronous way. In our approach, port data can change periodically or irregularly, the latter being opportune in the case of events. Each port has a

[7] `http://www.ni.com/labview`

name, an arbitrary data type and a current value - optionally incoming values can be stored in a queue, which is useful for video or audio streams. The most common type of port is one containing a numerical value - optionally with an SI unit assigned, as recommended in [17]. This way, widgets dealing with inputs from multiple sources can scale to suitable and uniform units and display the unit dynamically aside the numerical value.

Each interface to frameworks or hardware provides a set of such ports, arranged in a tree structure for the sake of clarity - since interfaces may provide hundreds of ports. Equally, each GUI widget provides a set of ports. Assuming their types *match*, these ports can be simply connected per drag-and-drop as described in section II-A. In order to match, the data types do not necessarily need to be of equal type. Rather, it is sufficient when the type of the destination port is more general in terms of object-orientation than the one of the source port. Comparing types of objects at runtime is a native feature of the Java programming language. For example, our Geometry Renderer Widget supports all incoming objects that implement a `Paintable` interface. Any Java class which can be serialized using the standard Java serialization mechanism is suitable as a port's data type. Actually, we discovered that a limited set of such data types is already sufficient for most uses. [18] discusses the difficulty of finding suitable abstract interfaces when using object-oriented interfaces to access hardware devices - including the problem of similar but incompatible interfaces for similar classes of devices. For example, it is difficult to define an object-oriented interface that supports all the relevant features of laser scanners available on the market without becoming bloated. With the port concept, we do not have these problems, since our interfaces include less semantics. They are merely a collection of input and output channels. To continue with the example, laser scanners with special capabilities can then simply provide additional such channels, at the price that a GUI might not run unchanged with different devices. But in our context, this is clearly not required. Actually, LabView shows that these types of interfaces can work well in practice.

D. Port Monitoring

Since the web server variant of our editor already includes a simple embedded HTTP server, we also publish status information and information on the robot's ports on simple separate web pages. They allow browsing the robot's ports and also setting their values - regardless of which framework is used. This is useful for testing and diagnosis.

E. Plugins

As outlined in section II, we implemented a plugin system that allows packaging further widgets and robot interfaces, as well as import and export filters for GUIs in plugins.

Usually, widgets have some parameters to specify their look and behaviour - for instance label texts, colours, number of input ports, etc. These are simply attributes in the widget's Java class, and are automatically serialized. Dialogs allowing user modification are generated automatically at runtime - hence causing programmers no extra work. This administrative part is generalised in the widget's abstract super classes. In effect, creating a widget comes down to defining ports and parameters, drawing the user interface, reacting to input port changes and triggering output port changes. Most of our widgets have fewer than 100 lines of functional code. Widgets may be composed of Java Swing components - as most of our widgets are. If higher rendering performance is required, they may also perform custom rendering on raw image buffers.

Creating interface plugins for hardware or frameworks can also be simple, depending on available Java support. Usually, it comes down to defining input and output ports, collecting sensor data regularly and forwarding control back to the attached framework or hardware.

F. Protocols

Wireless connections to our robots have limited bandwidth and unpredictable time delays. In the worst case, temporal connection losses occur. The same is true for any Internet connections. However, we need to transfer a lot of data - including camera video streams at preferably high resolutions and high frame rates, as well as low latency. This imposes several requirements on the networking mechanism we use for the web server communicating with the web browser GUI. First of all, it needs to be robust. When a connection fails, an operated robot should immediately switch to safe behaviour. This is done by providing default values for critical ports and by carefully monitoring round-trip times exceeding a critical threshold value. Generally, all our network packages containing data are acknowledged by the receiver. To minimize overhead, these acknowledgements are not sent immediately, but are rather appended to data that is sent anyway. Therefore our round-trip times are actually upper-bounds. With the default connection parameters they are up to 80ms higher than the actual value. If an acknowledgement for any packet is not received within by default 1500ms, a connection is considered to be temporarily down or unsuited, at least for our teleoperation scenarios. In this case, default values for critical input ports are applied. For example, velocity inputs are set to zero. To minimize the effects of latency, we use a push strategy following a subscriber mechanism. Ports that need data only occasionally may also switch to a pull strategy. Even after a complete network disconnect or restart of the host application, the GUI is automatically reconnected.

We also acknowledge data packets in order to optimize the rate at which data is sent. Two classes of port data are defined: *express* and *bulk* - with different quality of service. *Express* data (numerical sensor values, for instance) is small and should be delivered at high rates, whereas *bulk* data may be adjusted to the available bandwidth - video frames, for example, may be skipped. Many of these connection parameters - including for instance critical latency, express port update intervals or maximum number of not-yet-acknowledged bulk packets - are also published as ports and can be

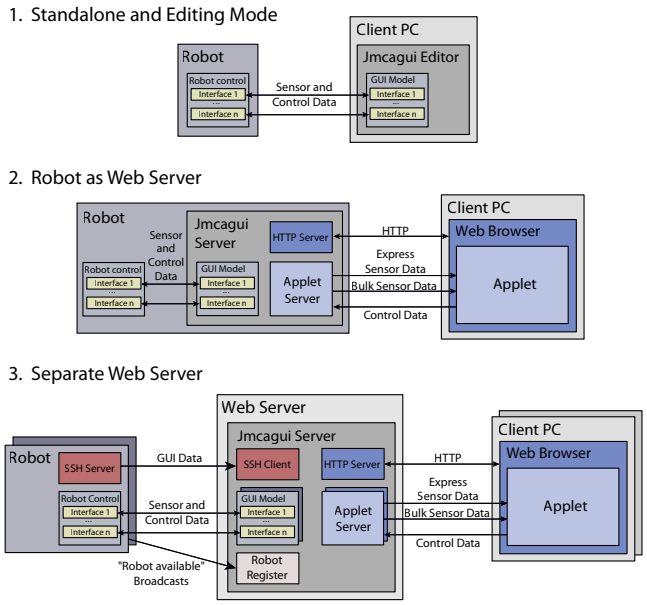

Fig. 3. Deployment

modified at runtime using the web interface. Alternatively, ordinary GUI elements such as sliders could be used to set these directly from a GUI. This can be useful for network tests, and when for some reason the built-in mechanisms do not perform well with the default parameters. For camera images we currently use an MJPEG video codec. While not optimal concerning bandwidth requirements, it introduces minimal latency, and leaving out frames is unproblematic. Apart from that, our TCP-based network mechanism groups port values in large network packets. Optionally, GZIP-based compression can be activated.

G. Deployment

Fig. 3 illustrates different variants of how a GUI can be deployed. The robot can be operated using the editor (described in section II-A) directly, as shown in the first case. This is useful, as GUIs can be designed and tested simultaneously.

In the second case, our tool is started on the robot itself - publishing the GUI as an applet using an integrated web server. Transferring data from the server to a client web browser is typically more efficient than connecting the editor natively with a robotic framework, as we can optimize data transfer (see section II-C). Regarding usability, only a standard web browser is required on a client instead of a full editor installation.

In the example shown in fig. 4, the robot's web server is accessible on port 4444. When deployed behind a firewall, the robot can be accessed by either opening this port, using an SSH tunnel or by setting up a VPN. However, if the firewall is not to be levered, a generic instance of our server can be installed separately - possibly integrated in a central web server (case 3). In this case, the GUI data is still stored on the robots and the generic server retrieves it via SSH for publication. Also, only a single connection between the generic server and the robot has to be established, regardless of the number of GUI clients connected - thus keeping wireless bandwidth usage static.

III. Summary

We presented a new tool for the creation, integration and deployment of graphical user interfaces. The unique combination of benefits of our tool is the convenient creation of interfaces, the flexible deployment and the powerful extensibility characteristics. As mentioned in section I-A, current frameworks do not offer these features in combination. Although we use our tool in the area of mobile robotics, it is useful for any kind of operable devices that have ethernet access, a plugin for interaction presumed. Based on web technology, our tool is suitable for quickly establishing an operational environment.

A. Usage

Currently, we support the MCA2 framework and provide basic connectivity to the Player Project as well as to Microsoft Robotics Studio. Furthermore, two industrial robots using an Adept Controller at the Department of Mechanical Engineering at our university can be controlled. These are an Adept 550 Table-Top Scara and a Sysmelec Cartesian micro assembly robot. The controller runs the V+ operating system.

Our mechanism for dealing with fluctuating connection conditions has proven valuable in practice. However, regarding general system concepts with web-based teleoperation, it is our opinion that the operator should control the robot with high-level commands instead of guiding the system directly, as link quality can never be completely guaranteed. In our application scenario (see section I-B and fig. 4) the operator has two choices: 1) He moves the mobile system with the joystick widget arranged at the bottom right of the GUI. If the wireless connection fails, the robot will apply default values, by in this case setting the speed command to zero. 2) If the operator chooses to control the robot indirectly in an autonomous mode by clicking on the 2D representation of the living environment - thus specifying goal coordinates - the robot will drive independently to the specified position even if the connection temporarily fails. The GUI also contains a camera widget and a geometry renderer widget displaying a map together with the robot's sensor distance data. Furthermore, there are LED widgets for the motors' activation state, a button for toggling this state, an LCD widget indicating the robot's control cycle period in milliseconds, and so on.

B. Experimental Features

We implemented experimental features such as reprogramming a robot during operation with an embedded interpreter on our robot. A Text Editor widget connected to the robot's Interpreted Text Port can be used to execute

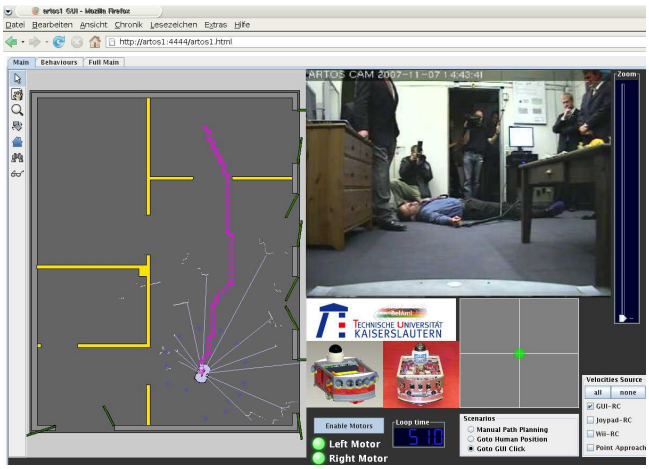

Fig. 4. Teleoperation in Emergency Scenario

arbitrary programmes in this module. Similarly, we offer a widget with a complete BeanShell[8] console and interpreter embedded. This allows executing arbitrary Java statements from the GUI. It has some predefined functions that make it easy to, for example, apply a sine signal to a port. This can be useful for testing.

C. Outlook

As the general architecture of our editor is finalised, the next steps include the integration of further robotic frameworks and hardware, as well as adding more graphical widgets.

Integration of a generic Java3D-based visualisation plugin will enhance our tool, since basically all robotic frameworks offer simulators with 3D content. So far, we embedded a specific 3D application widget displaying two industrial robots requiring the Java3D extension to be installed on respective web clients. Also, plugins for frameworks and hardware support may use native libraries in the future if required. Notably, this does not limit the platform independence of the applet variant, which runs on any Java-enabled web browser without requiring additional permissions.

Increased rendering performance is expected with the release of Java SE 6 Update N. Concerning the assisted living scenario we mentioned, we are engaged in tests with the local emergency and medical services to evaluate usability of the GUI.

IV. Acknowledgements

The presented work is funded by the Rhineland-Palatinate State Ministry of Science, Education, Research and Culture, by the Fraunhofer-Society, and by the German Federal Ministry of Education and Research.

[8] http://www.beanshell.org

References

[1] B. Gerkey, R. Vaughan, K. Sty, A. Howard, G. Sukhatme, and M. Mataric, "Most valuable player: A robot device server for distributed control," in *Proc. of the IEEE/RSJ Internatinal Conference on Intelligent Robots and Systems (IROS)*, Wailea, Hawaii, October 2001, pp. 1226–1231.

[2] T. H. J. Collett, B. A. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Australasian Conference on Robotics and Automation (ACRA)*, Sydney, Australia, December 5-7 2005.

[3] K.-U. Scholl, J. Albiez, B. Gassmann, and J. Zöllner, "Mca - modular controller architecture," in *Robotik 2002, VDI-Bericht 1679*, 2002.

[4] J. Koch, M. Anastasopoulos, and K. Berns, "Using the modular controller architecture (mca) in ambient assisted living," in *3rd IET International Conference on Intelligent Environments (IE07)*, Ulm, Germany, September 24-25 2007.

[5] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Orca: A component model and repository," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed. Berlin / Heidelberg: Springer - Verlag, April 2007, vol. 30.

[6] A. Cowley, L. Chaimowicz, and C. J. Taylor, "Roci: Strongly typed component interfaces for multi-robot teams programming," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed. Berlin / Heidelberg: Springer - Verlag, April 2007, vol. 30.

[7] C. Coté, D. Letourneau, and C. Ra, "Using marie for mobile robot component development and integration," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed. Berlin / Heidelberg: Springer - Verlag, April 2007, vol. 30.

[8] K. Goldberg, M. Mascha, D. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, "Desktop teleoperation via the world wide web," in *IEEE International Conference on Robotics and Automation (ICRA)*, Nagoya, Japan, May 1995, pp. 654 – 659.

[9] K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, and G. Smith, "Collaborative teleoperation via the internet," in *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, USA, April 2000.

[10] F. Michaud, P. Boissy, H. Corriveau, A. Grant, D. L. M. Lauria, R. Cloutier, M. A. Roux, and D. Iannuzzi, "Telepresence robot for home care assistance," in *AAAI Spring Symposium on Multidisciplinary Collaboration for Socially Assistive Robotics*, Palo Alto, USA, March 2007.

[11] A. Tapus, M. Matarić, and B. Scassellati, "The grand challenges in socially assistive robotics," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, March 2007.

[12] M. Becker, E. Ras, and J. Koch, "Engineering tele-health solutions in the ambient assisted living lab," in *21st International Conference on Advanced Information Networking and Applications (AINA)*, Niagara Falls, Canada, 2007.

[13] J. Koch, C. Armbrust, and K. Berns, "Small service robots for assisted living environments," in *VDI/VDE Fachtagung Robotik*, Munich, Germany, June 11-12 2008.

[14] P. Fiorini and R. Oboe, "Internet-based telerobotics: Problems and approaches," in *International Conference on Advanced Robotics (ICAR)*, Monterey, USA, 1997.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 1997.

[16] A. C. Domínguez-Brito, D. Hernández-Sosa, J. Isern-Gonzalez, and J. Cabrera-Gámez, "Coolbot: A component model and software infrastructure for robotics," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed., vol. 30. Berlin / Heidelberg: Springer - Verlag, April 2007.

[17] B. A. MacDonald, G. Biggs, and T. H. J. Collett, "Software environments for robot programming," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed. Berlin / Heidelberg: Springer - Verlag, April 2007, vol. 30.

[18] R. Vaughan, B. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Las Vegas, Nevada, USA, October 27-31 2003, pp. 2121–2427.