Principles in Framework Design applied in Networked Robotics [⋆]

Max Reichardt [*] **Tobias Föhst** [*] **Patrick Fleischmann** [*]
Michael Arndt [*] **Karsten Berns** [*]

[*] *Robotics Research Lab, Department of Computer Science, University of Kaiserslautern, Gottlieb-Daimler-Straße, 67663 Kaiserslautern, Germany (e-mail: {reichardt, foehst, fleischmann, m_arndt, berns}@cs.uni-kl.de).*

Abstract: In recent years, we developed a considerable range of networked robotic applications based on the software framework FINROC. Such systems often require integrating diverse communication technologies, protocols, and computing platforms. In this context, beneficial concepts and approaches applied in the framework are introduced – and their positive impact on flexibility, interoperability, and development effort of resulting applications is discussed. Furthermore, applications are presented and tool support is illustrated.

Keywords: Autonomous mobile robots, Computer networks, Programming environments, Robot control, Robotics, Software engineering, Software tools, System architecture

1. MOTIVATION

Many important and innovative applications in the service robotics domain depend on computer networks. Apart from distributing robot control systems to multiple computing nodes, common use cases include teleoperation and multi-agent systems. Obtaining information on a robot's operating environment from sensor nodes and other external systems is another relevant area. It has potential to increase a robot's performance – or to reduce costs, size, and energy consumption of sensors required on the robot.

Systems to be connected are often heterogeneous with respect to computing architectures, communication technologies, protocols, and patterns. It is always possible to implement specific solutions for interconnecting two particular systems. However, this can cause considerable effort with respect to development and maintenance. Solutions that cover whole classes of systems are beneficial in this regard and also more flexible.

Robot control systems are typically based on powerful frameworks that provide a broad range of standard functionality as well as valuable development tools. They have a fundamental impact on effort required for development and maintenance of robot control systems as well as many of their quality attributes, as discussed in [Reichardt et al. (2013b)]. Interoperability and integrability are quality attributes that are particularly important in the area of networked robotics. Support for relevant communication standards has a major impact in this respect.

Naturally, an adequate network transport mechanism is of central importance for distributed applications – which have varying requirements in this regard. Robustness, low bandwidth requirements, low computational overhead, low latency, and support for quality of service (QoS) parameters are typically desirable attributes. Security is important for robot operation over the Internet. Slim solutions are required for small, embedded systems. Supported communication patterns and network topology are further aspects for consideration.

Over the years, we have developed a considerable range of networked robot control systems. They are illustrated in Fig. 1. Central areas are user interfaces for remote robot operation and smart environments – involving a notable range of applications, hardware devices, communication technologies, and standards. Some systems obtain relevant information from the Internet.

The older applications were realized in MCA2 [Scholl et al. (2001)], while newer applications are based on FINROC [1] [Reichardt et al. (2013b)]. There are decisions in robotic framework design that significantly determine how suitable they are for different types of networked applications. These, and the approaches taken in FINROC, are discussed in section 3 – while section 4 presents some applications in more detail.

2. RELATED WORK

Virtually all robotic frameworks support creating networked applications. Systems are typically decomposed into components (sometimes also called "modules" or "nodes") which may be instantiated in separate processes possibly running on different systems. They can be connected in a network-transparent way in order to create distributed robot control systems.

Network transport mechanisms have significantly differing characteristics. Some frameworks provide slim, custom

[⋆] Funding by the German Ministry of Education and Research (grant 01IC12S01W, Software-Cluster project SINNODIUM) is gratefully acknowledged.

[1] `http://www.finroc.org`

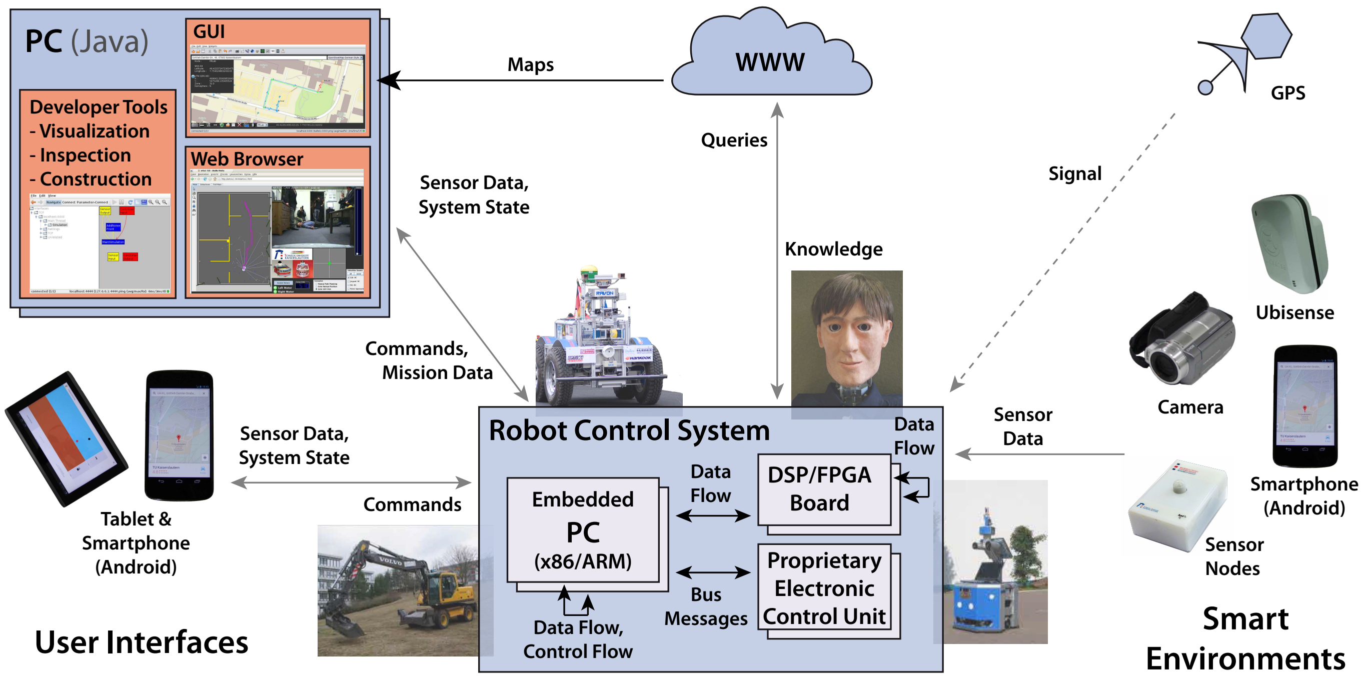


Fig. 1. Networked robotics use cases at the Robotics Research Lab. Black arrows indicate wired connections, while grey arrows indicate connections that are at least partly wireless.

solutions tailored to their requirements. Examples include the Player Project [Gerkey et al. (2001)] and ROS [Quigley et al. (2009)]. Others rely on popular middleware packets and standards. Orocos [Soetens (2006)] and OpenRTM-aist [Ando et al. (2008)], for instance, provide implementations based on CORBA. Due to the popularity of ROS, its TCP-based network transport has become a de-facto standard that several other approaches are interoperable with.

Some frameworks are independent of a specific network transport, as no solution is suitable for every application. The Player Project is among the first frameworks designed in this way. OPRoS [Jang et al. (2010)] features a notable solution with respect to middleware transparency. Few frameworks including Orocos are transport-independent, even for intra-process communication.

Interoperability with third-party systems requires supporting the standards or protocols they use. YARP is a network transport-independent approach "designed to play well with other architectures" [Fitzpatrick et al. (2008)]. [Wienke et al. (2012)] present a notable approach to achieve interoperability among systems based on different middleware standards and IDLs. If not provided by the framework, bridges are often implemented as components. The component model can be a limiting factor in this respect – e.g. if component interfaces must be static.

[Broxvall et al. (2007)] propose the PEIS middleware for smart environments involving robots. Notably, its lightweight variant is suitable for embedded systems with merely 4kb of memory.

An extensive survey on design principles in state-of-the-art frameworks can be found in [Reichardt et al. (2013a)]. We are not aware of any other solution providing the central features presented in the next chapter in combination.

3. FINROC

Development of FINROC began in 2008 at the University of Kaiserslautern. In a systematic design approach, critical areas of design were identified and investigated – carefully considering their impact on the many relevant quality attributes of robot control systems. Apart from that, the qualities we learned to appreciate in MCA2 were preserved. Thus, application style and tools are in many ways similar.

FINROC's key features include a slim and highly modular framework core as well as an efficient, zero-copy, lock-free, real-time transport for intra-process communication. Intra-process runtime construction and suitability for up to thousands of components are further distinguishing properties. In addition, it features network transport-independence, support for multiple and extensible component models, composite components, and dynamic component interfaces. Any copyable or movable C++11 type can be used in component interfaces – enabling using domain types directly. Supported communication mechanisms include data-flow ports, service ports and blackboards – notably all provided as optional plugins (see Fig. 3). Details can be found in [Reichardt et al. (2013b)].

There are full FINROC implementations available in C++11 and Java. The latter is suitable for the Android operating system. FINROC-lite is a related lightweight C++ implementation developed by Robot Makers GmbH[2] for small embedded processors. It is currently used on Altera Nios II soft cores.

In order to increase reusability of software artifacts and to avoid framework lock-in, functionality that is not framework-dependent is implemented in independent

[2] http://robotmakers.de

libraries called RRLIBs – contributing to a separation of concerns in framework implementation and components.

3.1 Network-related Design Principles

Figure 2 illustrates a selection of quality attributes, framework design aspects, and some of their major relations in the context of networked robot control systems.

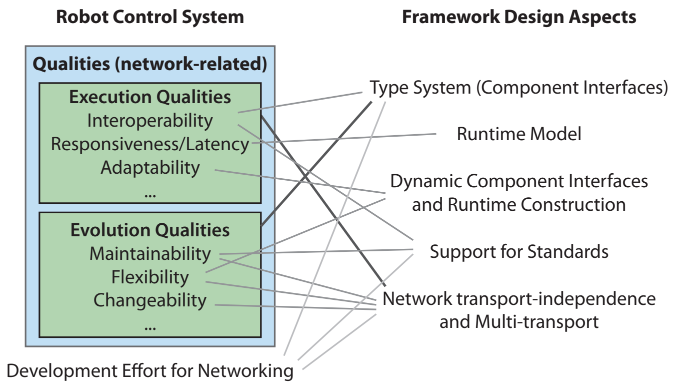


Fig. 2. Overview on network-related topics in framework design

The network transport has an impact on all network-related execution qualities of a system. We opted for network transport-independence in FINROC so that suitable transports for every application can be selected. As the choice can be changed later, this has a positive impact on several evolution qualities of the implemented system.

Figure 3 shows FINROC's slim and highly modular framework core with a selection of plugins. All plugins are optional. By selecting a suitable set of plugins, the framework can be tailored to the requirements of an application. Notably, network transports are also provided as plugins – or as components. Typically, components are used for communication with a defined set of subsystems. They are instantiated at a specific place in a robot control system and have similar characteristics to components providing hardware drivers. Plugins are used for network transports that are added to whole FINROC runtime environments.

Two such plugins are highlighted. *ros* provides interoperability with ROS components. *tcp* is a custom TCP-based protocol tailored to FINROC's requirements. It is a binary peer-to-peer protocol featuring simple quality-of-service. Furthermore, it is robust with respect to temporary connection losses and allows changing update rates for every data port individually at runtime. The implementation is based on the *Boost.Asio* library and opens only one TCP port. As our industrial partners acknowledged, it is suitable and convenient for remote maintenance over the Internet using FINROC's standard development tools. An early implementation has been used for teleoperation over the Internet [Koch et al. (2008)]. For more sophisticated networking, a plugin for integrating middleware based on the DDS standard is being developed.

Supporting relevant standards has a positive impact on interoperability of systems. Furthermore, as standards are mature and stable, they are beneficial with respect to development effort and maintainability. However, lack of

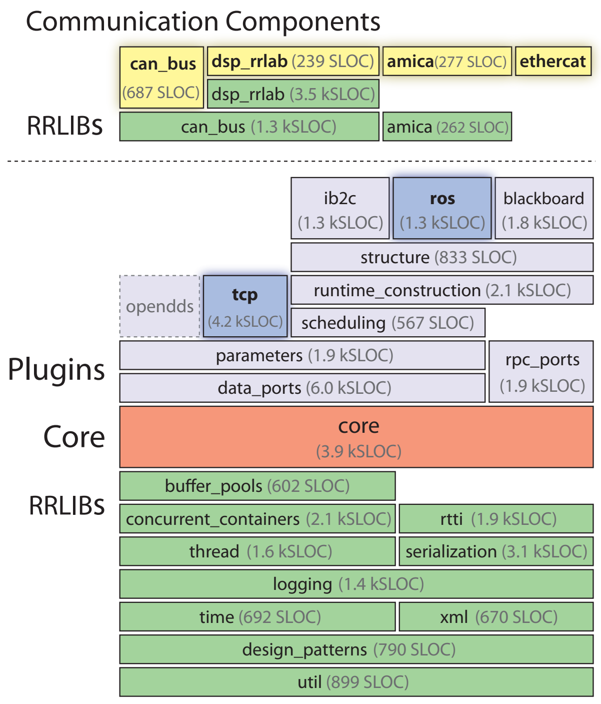
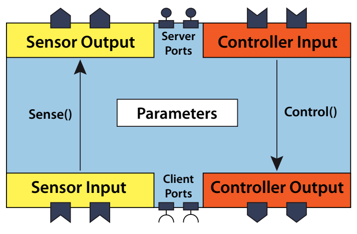


Fig. 3. FINROC's modular core with a selection of plugins and communication components

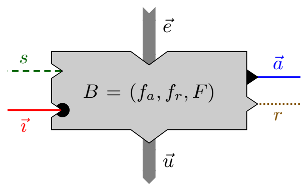
required features or heavyweight implementations can be drawbacks for homogeneous framework integration.

Robot Makers GmbH developed support for the EtherCAT real-time bus. This allows integrating standard components from automation industry. *dsp_rrlab* provides a custom, CAN-based protocol for communication between embedded PCs and DSP boards, while *can_bus* is a more generic component for exchanging messages with third-party electronic control units. *amica* provides access to wireless sensor nodes – as presented in section 4.2.

Notably, the FINROC components wrapping these protocols are thin – with often only a few hundred lines of code. In other frameworks with similar properties, it should be possible to wrap the RRLIBs with equally little effort. Most of the communication plugins and components contain bridges that attach I/Os of the protocol they contain to FINROC data ports.



(a) "SenseControlModule"



(b) iB2C behavior

Fig. 4. Two FINROC component types

Component types are also provided via optional plugins – e.g. *structure* and *ib2c*. The *structure* plugin contains FINROC's standard component types. One is the "SenseControlModule" (see Fig. 4a), which is very similar to "Modules" in MCA2. It processes sensor and controller data in separate tasks that have their own interfaces for incoming and outgoing data. In FINROC, service ports were added. *ib2c* contains components for the behavior-based architecture [Proetzsch (2010)]. Fig. 4b shows a simple behavior. These have only data flow interfaces.

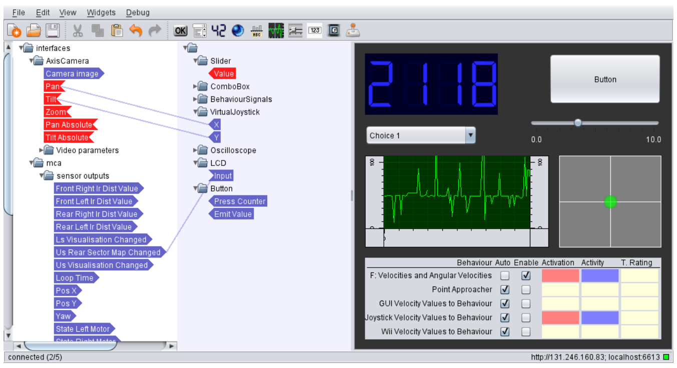
Without component plugins, FINROC can be set up as middleware only. It is available as shared library that can be linked against, in order to be easily integrable in other applications – possibly for enabling interoperability.

The types that can be used in component interfaces are a key issue with respect to interoperability. Some frameworks apply IDLs, while others allow using e.g. C++ types directly – provided they fulfill certain requirements. These requirements typically include defining serialization. In order to use data types from third-party libraries directly, it must be possible to specify serialization without modifying classes – e.g. via C++ operator overloading or traits. A well-defined IDL has major advantages with respect to multi-language support. Bridging between two IDLs is challenging. In FINROC, we opted for native data types in component interfaces. This can avoid overhead for data conversion. Notably, types generated by IDLs may be used as well.

The ability to instantiate, connect, and delete components at application runtime (runtime construction) can have advantages for networked applications. When operating in smart environments, for instance, robots typically need to condense the available sensor information in homogeneous views of the environment. Suitable components to perform such tasks can be created as sensors are encountered. Apart from that, it simplifies implementations of communication components with dynamic sets of I/Os if ports can be added and removed from their interfaces at runtime.

Finally, the runtime model has an impact on the latency of a networked system. If incoming data is processed in a periodic loop, data will be processed with an average delay of half the cycle time. FINROC provides event-driven execution in case this is to be avoided.

3.2 Graphical Tooling



Behaviour	Auto	Enable	Activation	Activity	T. Rating
F: Velocities and Angular Velocities	☐	✓			
Point Approacher	✓	☐			
GUI Velocity Values to Behaviour	✓	☐			
Joystick Velocity Values to Behaviour	✓	☐			
Wii Velocity Values to Behaviour	✓	☐			

Fig. 5. FINGUI: flexible GUI editor

There are two major graphical tools for FINROC – both implemented in Java. FINGUI is a flexible and extensible user interface editor (see Fig. 5 and Fig. 7). Widgets are arranged on a scalable canvas – possibly in multiple tabs and windows. Similar to components, a widget's interface is a set of ports. These can be connected to any port in

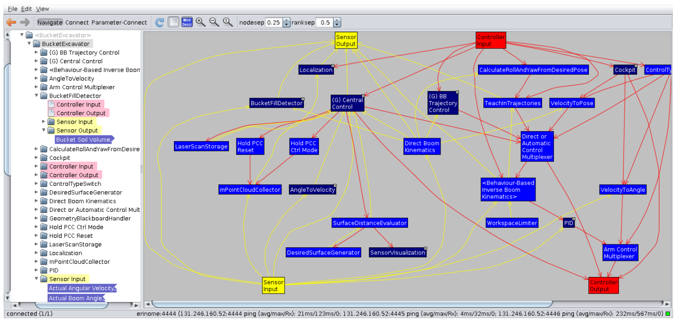


Fig. 6. FINSTRUCT: application visualization, inspection, and construction

a connected FINROC application with a compatible type. New widget types can be added via plugins. In principle, they may contain complex Java Swing applications. Generally, the FINGUI helps to minimize development effort for graphical user interfaces of predictable quality.

FINSTRUCT is a graphical tool for application visualization and construction (see Fig. 6). It enables viewing and setting the current values of all data flow ports and parameters in running FINROC applications – an indispensable help for testing, debugging, and tracking down malfunctioning components. Components can be instantiated, connected, and deleted at application runtime. As changes can be saved, it is an optional tool for live creation of robot control systems. It is used in several projects in this way. An extensible set of special-purpose views enables visualizing particular aspects of an application. There are special such views for behavior-based networks or visualization of the amount of data exchanged by components.

4. NETWORKED ROBOTIC APPLICATIONS

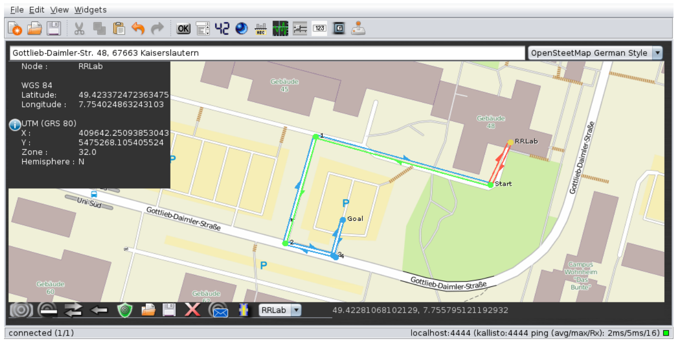
As Fig. 1 indicates, various networked robotic applications were realized at the Robotics Reasearch Lab. [Koch et al. (2008)] present the FINGUI tool in more detail – including options to integrate user interfaces in web pages. The latter was used for semi-autonomous teleoperation of a mobile indoor robot over the Internet – the robot operating in a smart environment in the context of ambient assisted living. Other examples include remote operation of commercial vehicles using mobile Android devices, or exploiting Internet services in interaction of humans with a humanoid robot.

In the following, two applications are presented in more detail: one involving user interfaces as well as Internet services, the other one dealing with wireless sensor nodes in smart environments.

4.1 Mission Planning in Aerial Images

Based on FINROC's graphical user interface editor FINGUI, a plugin to plan, visualize and analyze robotic missions has been developed. Its widget uses aerial images as well as map data from different web providers to display all information in a human readable way (see Fig. 7).

Usability of the widget is comparable to well-known map services available on the web. By typing a postal address, a geographic coordinate (e.g. latitude 49.4235 and longitude 7.7540) or a *UTM* coordinate into the address bar, the



Gottlieb-Daimler-Str. 48, 67663 Kaiserslautern

```
Node :        RRLab

WGS 84
Latitude:     49.423372472363475
Longitude :   7.754024863243103

UTM (GRS 80)
X :           409642.25093853043
Y :           5475268.105405524
Zone :        32.0
Hemisphere :  N
```

49.42281068102129, 7.755795121192932

connected (1/1)   localhost:4444 (kallisto:4444 ping (avg/max/Rx): 2ms/5ms/16)

Fig. 7. FinGUI with a topological map shown in the developed widget

location of the mission can be defined. The postal address or location description is converted to a geographic coordinate using *Google's Geocoding API*. In contrast to most of the online map services, the widget supports multiple data providers via a plugin mechanism, such as *Esri ArcGIS*, *Google Maps*, *Microsoft Bing*, *Nokia Maps* and *OpenStreetMap* as well as their different layers. Additionally, it offers a seamless switch between these providers, preserving the current section and zoom settings. Furthermore, it has a built-in caching mechanism to provide data when no Internet connection is available.

For the task of manual mission planning, a topological map represented as a directed graph can be set up by an operator. Therefore, the user defines waypoints based on the aerial imagery or the map data which can be positioned and adjusted using the mouse or by entering *GPS*/*UTM* coordinates. For graphically placed nodes, the geographic location is automatically determined based on the *GPS*-annotated images. Afterwards, the connections between these nodes can be edited using the mouse. Here, bidirectional as well as unidirectional edges are possible to be able to model one-way connections. To start the new mission, the map can be transferred to the robot in an XML-based representation using Finroc's built-in communication.

The widget was originally developed for the ravon project [Armbrust et al. (2010)], where it can also be used to update a mission by correcting or extending the topological graph. This is possible because of ravon's multi-layer architecture, allowing the navigator to replan while the behavior-based pilot steers the robot safely through the local environment. That way, by continuous updating of the graph and replanning by the navigator, the widget becomes a remote control that allows steering the robot by clicking on aerial images.

The widget also offers several options for analyzing a running mission. For instance, the robot's position and orientation can be displayed as an overlay on the aerial imagery, which can help to identify possible failures e.g. of a *GNSS* receiver or an inertial measurement unit. For a human operator the visualization within aerial images is more intuitive than interpreting *GPS* coordinates or yaw angles.

Again motivated by the application in ravon – the topological map is evaluated and modified as the robot drives along its connections [Braun (2009)] – additional map information can be shown as another layer. In contrast to the previously mentioned mission setup, the evaluated map contains information regarding the drivability and the accessibility as determined by the robot's planning and navigation algorithms during the mission. The drivability is currently divided into three different groups: *executable*, *not executable*, and *speculative* if no execution information is available yet. In the visualization, these classes are indicated by different colors for edges and nodes.

4.2 Wireless Sensor Nodes

Finroc has been used together with nodes of a Wireless Sensor Network (WSN) on several occasions. One of the first applications was the optimization of a mobile robot's navigation by tracking people within a WSN using a particle filter [Arndt and Berns (2012)]. People moving in the environment were detected with the help of passive infrared (PIR) sensors on wireless sensor nodes, distributed in the target environment. The robot was able to use this data to estimate positions of people and adapt its maximum velocity according to the system state.

The work mentioned above employed nodes of the AmICA wireless sensor network platform. These nodes are based on a low-power 8 bit microprocessor (Atmel Atmega 324P), a 433/868/915 MHz radio module, and several sensors and actuators which can be mounted on each node.

As already mentioned in section 3, the communication with AmICA nodes is provided by a Finroc component.

The frames that are sent over the radio layer are the basic elements of communication in the AmICA WSN. They can be used as a native data type in the Finroc framework, allowing modules to send and receive frames.

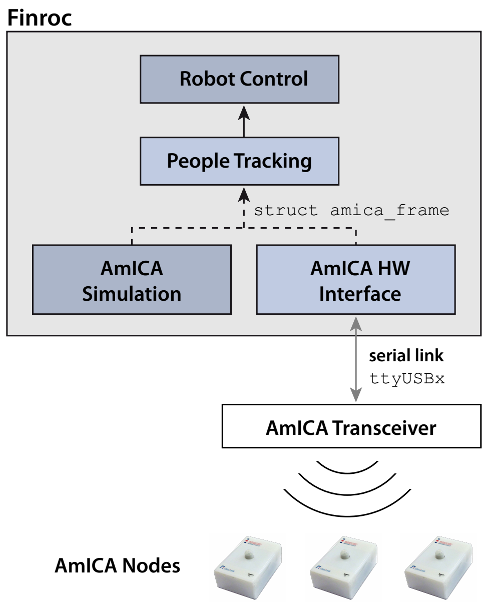


Fig. 8. Integration of AmICA wireless sensor nodes in the Finroc framework

In addition to interfacing the real hardware, experiments can also be conducted by replacing the real hardware with a simulation of the nodes. This simulation backend is realized using the simulation tool SimVis3D in Finroc.

Figure 8 provides an architectural overview of how the real or the simulated hardware can be accessed.

The simulation environment of the AmICA nodes has also successfully been used to feed node data back to the Smart Environment Platform framework TinySEP [Wille et al. (2012)], which was initially intended only to be used with real hardware. Recorded data from a real WSN has served as the base for automatically creating different simulation scenarios which have then been "played back" with the help of SimVis3D. The setup has been used to evaluate the performance of TinySEP under many different conditions [Arndt et al. (2013)].

5. CONCLUSION AND FUTURE WORK

In this paper, we discuss principles in framework design with particular relevance for networked robotic applications. The Finroc framework is an example of how they can be implemented. The successful use in a diverse range of robotic applications – notably including commercial ones – shows that these approaches are suitable and beneficial for real-world applications.

Over the last decade, we implemented a considerable collection of drivers and algorithms for robot control systems. They are available as slim and independent open source libraries (rrlibs) as well as Finroc components. In its current state, we believe that Finroc provides the necessary means to conveniently create efficient, complex robot control systems. Furthermore, it integrates well with other solutions and systems – making it a viable option for subsystems. All these approaches aim at providing reusable software artifacts that are of value in the research community also.

Future technical work will include the integration of further middleware standards in Finroc plugins – such as DDS, RT Middleware [Ando et al. (2008)] or OPC UA. Current projects target service robot operation in further smart environments, namely production facilities and construction sites.

REFERENCES

Ando, N., Suehiro, T., and Kotoku, T. (2008). A software platform for component based rt-system development: Openrtm-aist. In S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. von Stryk (eds.), *Simulation, Modeling, and Programming for Autonomous Robots*, volume 5325 of *Lecture Notes in Computer Science*, 87–98. Springer Berlin / Heidelberg.

Armbrust, C., Braun, T., Föhst, T., Proetzsch, M., Renner, A., Schäfer, B.H., and Berns, K. (2010). RAVON – the robust autonomous vehicle for off-road navigation. In Y. Baudoin and M.K. Habib (eds.), *Using robots in hazardous environments: Landmine detection, de-mining and other applications*. Woodhead Publishing Limited. ISBN: 1 84569 786 3.

Arndt, M. and Berns, K. (2012). Optimized mobile indoor robot navigation through probabilistic tracking of people in a wireless sensor network. In *Proceedings of the 7th German Conference on Robotics (Robotik 2012)*, 355–360. VDI Verlag, Berlin, Munich, Germany.

Arndt, M., Wille, S., de Souza, L., Rey, V.F., Wehn, N., and Berns, K. (2013). Performance evaluation of ambient services by combining robotic frameworks and a smart environment platform. *Robotics and Autonomous Systems*, 61(11), 1173 – 1185.

Braun, T. (2009). *Cost-Efficient Global Robot Navigation in Rugged Off-Road Terrain*. Verlag Dr. Hut. ISBN: 978-3-86853-135-0.

Broxvall, M., Seo, B.S., and Kwon, W.Y. (2007). The peis kernel: a middleware for ubiquitous robotics. In *Proc. of the Workshop on Ubiquitous Robotic Space Design and Applications, in conjunction with the International Conference on Intelligent Robots and Systems (IROS)*. San Diego, USA.

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1), 29–45.

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

Jang, C., Lee, S.I., Jung, S.W., Song, B., Kim, R., Kim, S., and Lee, C.H. (2010). Opros: A new component-based robot software platform. *ETRI Journal*, 32, 646–656.

Koch, J., Reichardt, M., and Berns, K. (2008). Universal web interfaces for robot control frameworks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nice, France.

Proetzsch, M. (2010). *Development Process for Complex Behavior-Based Robot Control Systems*. Verlag Dr. Hut. ISBN: 978-3-86853-626-3.

Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A.Y. (2009). ROS: an open-source robot operating system. In *Proceedings of the Workshop on Open Source Software in Robotics, in conjunction with the IEEE International Conference on Robotics and Automation (ICRA)*. Kobe, Japan.

Reichardt, M., Föhst, T., and Berns, K. (2013a). Design principles in robot control frameworks. In *Informatik 2013*, Lecture Notes in Informatics (LNI). Springer, Koblenz, Germany.

Reichardt, M., Föhst, T., and Berns, K. (2013b). On software quality-motivated design of a real-time framework for complex robot control systems. *Electronic Communications of the EASST*, Software Quality and Maintainability(60). `http://journal.ub.tu-berlin.de/eceasst/article/view/855`.

Scholl, K.U., Albiez, J., and Gassmann, B. (2001). Mca-an expandable modular controller architecture. In *3rd Real-Time Linux Workshop*. Milano, Italy.

Soetens, P. (2006). *A Software Framework for Real-Time and Distributed Robot and Machine Control*. Ph.D. thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

Wienke, J., Nordmann, A., and Wrede, S. (2012). A meta-model and toolchain for improved interoperability of robotic frameworks. In *SIMPAR2012 - Simulation, Modeling, and Programming for Autonomous Robots*. Springer Berlin / Heidelberg, Tsukuba, Japan.

Wille, S., Shcherbakov, I., de Souza, L., and Wehn, N. (2012). Tinysep - eine kompakte plattform für ambient assisted living. In *Technik für ein selbstbestimmtes Leben (AAL 2012)*.