Requirements for Interoperability and Seamless Integration of Different Robotic Frameworks

Michael Arndt, Max Reichardt, Jochen Hirth, Karsten Berns

Abstract— **Many different robotic frameworks exist, each with its own advantages, disadvantages, peer groups and communities. In many cases, interoperability of different robotic runtime architectures is desired, but due to the complexity of converting between data-types and working with multiple build-systems, interoperability is hindered. This work aims to identify important design principles that – when satisfied – can enable a seamless integration of frameworks with little effort. The approach is validated in a number of real-world experiments that require data exchange between the two robotic frameworks ROS and FINROC.**

I. INTRODUCTION

In order to increase the quality of robot control systems and to reduce development effort, research on software frameworks is of major importance. To perform such research, it is often necessary for institutions to develop and maintain their own frameworks. As discussed in [1], this can be perfectly feasible. Competition for the best concepts and implementations is considered advantageous, as long as this diversity and heterogeneity does not hinder integration and reuse of software artifacts across research institutions. Every researcher should have the freedom to choose his or her favorite framework.

In the past, however, in the context of international cooperations between different institutions, the "framework barrier" problem has been encountered several times: Different institutions actually *wanted* to work on joint projects, but due to the diversity of frameworks and their inability to easily integrate with each other, things were much harder than necessary.

Although the good practice of consequently separating framework-independent software artifacts from framework-dependent code is increasingly promoted ([1], [2], [3], [4]), freely available solutions that have derived from research projects are often highly framework-dependent. In order to reduce development effort and avoid reinvention of the wheel, it would be beneficial to use and reuse them nevertheless – by creating a bridge to close the gap between various robotic frameworks, their paradigms of communication, their middlewares and their data types.

The interoperability between different robotic software components from different vendors is indeed rather poor and at an early stage of development, as noted by Smits et al. [5]. With their work and through efforts of Cote et al. [6], steps in integrating multi-vendor, multi-component software components in the robotics domain have been made. In fact, we strongly agree with their argumentations and ideas, so this work can be seen as a continuation of their foundations by extending it to other frameworks and also other paradigms of communication, as we will show that not only compatibility on a data-flow level, but also on a control-flow level can be achieved between different frameworks.

As interoperability is a quality attribute of robotic software and the possibility of realizing interoperability with little implementation effort is highly desirable, it has been investigated which practices in framework design help to achieve this.

II. DESIGN FOR INTEROPERABILITY

In recent years, we have realized interoperability between the FINROC [4] framework and several other frameworks including ROS [3], URBI [7], MCA2 [8], Player [9] and Microsoft Robotics Developer Studio[1] to a varying degree. From this experience as well as through literature research, good practices in framework design have been identified which allow making frameworks interoperable with little effort, and lead to low computational overhead of the resulting systems. These include:

- Building and Linking: Linking two frameworks in one application typically leads to the simplest solutions[2]. This requires that a framework is available as a shared library that can easily be linked against – without requiring the use of a certain build system. Furthermore, there must not be a mandatory *main()* function.
- Data Types exchanged among components: Ideally, any C++ data type can be used in component interfaces. This way, classes from framework-independent libraries such as, for instance, the PCL (Point Cloud Library) can be used directly. Notably, this reduces overhead for data conversion. Therefore, it must be possible to specify framework-specific serialization without modifying those classes. C++ operator overloading is a suitable mechanism to achieve this.
- Component Interfaces: Typically, it is easier to achieve generic interoperability for data flow than for control flow. In many cases the former is sufficient. This requires that components can have data inputs and outputs in their interfaces.

M. Arndt, M. Reichardt and K. Berns are with the Robotics Research Lab, Dept. of Computer Sciences, University of Kaiserslautern, Kaiserslautern, Germany {m_arndt,reichardt,berns}@cs.uni-kl.de

J. Hirth is with Robot Makers GmbH, Kaiserslautern, Germany hirth@robotmakers.de

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

[2]Possibly, the network transport from one of the frameworks is already sufficient. Implementing the same network protocol in both frameworks is another more laborious option.

III. Integrating Finroc and ROS

As an example of how well interoperability can be established between two robotic frameworks that fulfill the abovementioned criteria, this section shows some technical details of how ROS and FINROC are connected. Both data- as well as control-flow between the two is possible using a ROS interoperability plugin for the FINROC framework. This plugin also acts as a ROS *node*. It can on the one side communicate with FINROC components and on the other side, it connects to a ROS core, being able to publish/subscribe to topics and also being able to perform and receive requests/responses on services. The overall architecture with all possible use-cases is depicted in Figure 1.

The naive way to connect two frameworks involves the conversion between their standard data types, as indicated in the first two use-cases of Figure 1. However, complete conversions between types cause overhead, and they are not always necessary. In practice, it is much easier just to transfer native data types through the bridge and only to extract the really relevant information on demand, deeper down in other components of the framework.

In order to support arbitrary data types in FINROC, operator overloading is applied to make types serializable. This enables them to be used in input, output and RPC ports. In ROS, *message traits* are employed for serialization and for generating a message out of arbitrary objects by adding checksums, type-information and more.

With this plugin, it is possible to use an arbitrary ROS message as a data type within FINROC by just *once* including a single header file, which adds serialization operators to *all* known types that represent ROS messages. On the other hand, it is also possible to create a ROS message out of an arbitrary FINROC data type[3]. In this case, however, the programmer must manually define the traits for ROS, as there are parameters which are not automatically deducible, such as the checksum. Nevertheless, it is only a matter of minutes to make a FINROC data type usable in ROS[4]. Control flow (in the form of *remote procedure calls* on the FINROC side or *services* on the ROS side) is handled using a special FINROC port called `tROSRPCPort`.

IV. Experiments

To validate the approaches that have been discussed so far, several experiments with real-world problems have been conducted – covering different use cases. Two of them are presented in this extended abstract:

A. Interacting from Finroc with a Pioneer Robot running ROS

A Pioneer P3-DX mobile robot running ROS with the `p2os` and `move_base` stacks has been used to evaluate the interaction between FINROC and ROS by using FINROC data types as ROS messages.

[3]This term refers to an arbitrary data type with FINROC serialization operators defined.

[4]Basically as described at `http://www.ros.org/wiki/roscpp/Overview/MessagesSerializationAndAdaptingTypes`

This experiment aims to enhance the results previously obtained in [10]. In that work, a wireless sensor network had been used to track a person in an office environment, in order to optimize the velocity of a mobile robot whilst maintaining safety. The previous work was conducted exclusively in FINROC; the new experiment makes use of a heterogeneous configuration, with the tracking taking place in FINROC while the robot's basic drive architecture is based on ROS. Also, this experiment focuses on the optimization of the path-planning of the mobile robot and not on velocity adaption.

The structure of the overall system, consisting of FINROC and ROS components, is visualized in Figure 2. On the FINROC side, at the lowest level, there is the hardware interface to the AmICA wireless sensor network nodes. The raw information from the sensor nodes is processed using a tracking algorithm based on a particle filter (details can be found in [10]). The result of this step is a probability distribution which indicates where people are most likely situated in the environment. This data is encapsulated in the native FINROC data type `tProbabilisticData`.

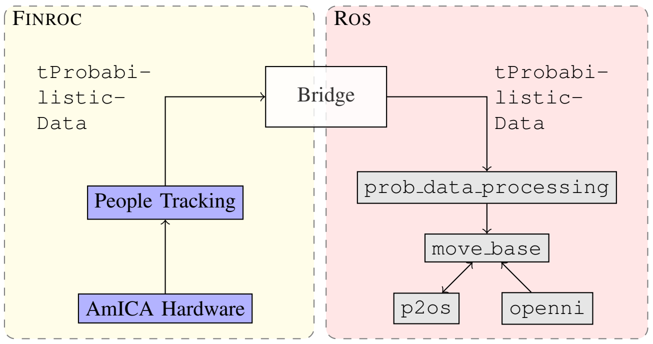

Fig. 2. Pioneer robot running ROS and using information from FINROC

Once received by ROS, a node analyzes the probabilistic data and injects information about "virtual" obstacles into `move_base` if there are people present in the environment. Now, depending on the tracking results (i.e. where people are likely situated), the robot can take different paths from start to goal. For this experiment the task of the robot was to always choose the "safest" path, even if there might be shorter ones.

B. Controlling a Commercial Finroc Robot using ROS

The mobile offroad robot VIONA (Vehicle for Intelligent Outdoor NAvigation) is a commercially available platform equipped with double-ackermann-kinematics, developed and built by Robot Makers GmbH (see Figure 3). The basic control system of the robot is realized with FINROC (see Figure 4). It includes the hardware interface, which communicates with the underlying motor controllers and sensors.

For the implementation of the high level control components, an interface is provided, which supports FINROC and ROS respectively – giving customers the opportunity to choose their favorite framework for high level robot control.

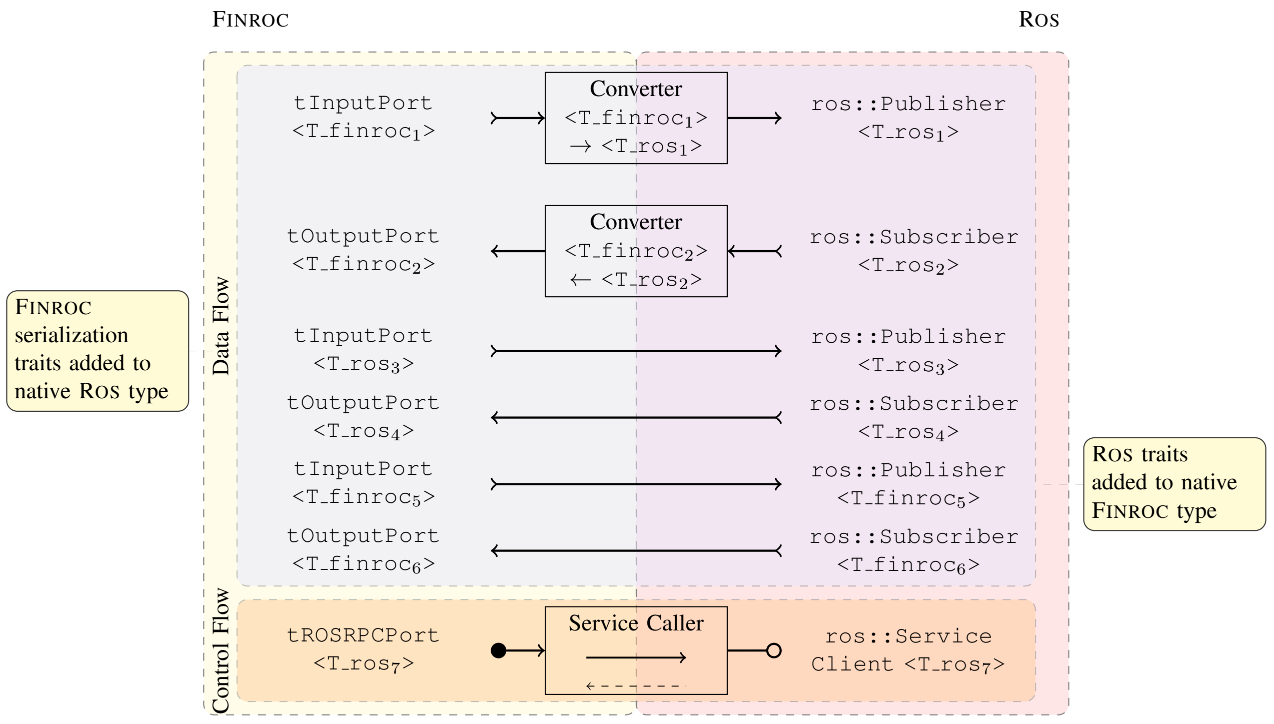

Fig. 1. Overview of the FINROC-ROS interoperability bridge

Fig. 3. Robot VIONA © Robot Makers GmbH

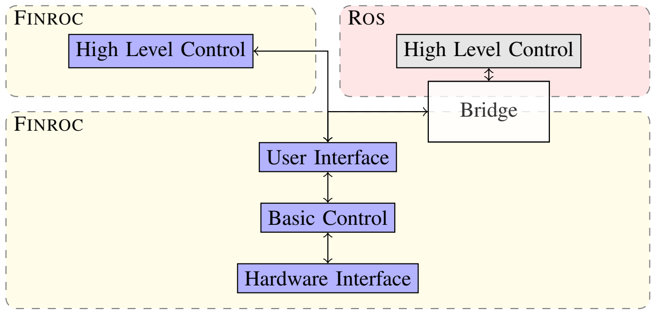

Fig. 4. Control system of VIONA with high-level control implemented either in FINROC or in ROS

REFERENCES

[1] A. Makarenko, A. Brooks, and T. Kaupp, "On the benefits of making robotic software frameworks thin," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, San Diego, California, USA, October 29-November 2 2007.

[2] "The robot construction kit," http://rock-robotics.org/.

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *IEEE Intl. Conf. on Rob. and Auto. (ICRA)*, Kobe, Japan, May 12-17 2009.

[4] M. Reichardt, T. Föhst, and K. Berns, "On software quality-motivated design of a real-time framework for complex robot control systems," in *Proceedings of the 7th International Workshop on Software Quality and Maintainability (SQM), in conjunction with the 17th European Conference on Software Maintenance and Reengineering (CSMR)*, March 5 2013.

[5] R. Smits and H. Bruyninckx, "Composition of complex robot applications via data flow integration," in *Rob. and Auto. (ICRA), 2011 IEEE Intl. Conf. on*, May 2011, pp. 5576–5580.

[6] C. Cote, 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.

[7] J.-C. Baillie, "Design principles for a universal robotic software platform and application to urbi," in *2nd National Workshop on Control Architectures of Robots (CAR'07)*, Paris, France, May 31-June 1 2007, pp. 150–155.

[8] K. U. Scholl, J. Albiez, and G. Gassmann, "Mca- an expandable modular controller architecture," in *3rd Real-Time Linux Workshop*, Milano, Italy, 2001.

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

[10] M. Arndt and K. Berns, "Optimized mobile indoor robot navigation through probabilistic tracking of people in a wireless sensor network," in *Proc. of the 7th German Conf. on Rob. (Robotik 2012)*. Munich, Germany: VDI Verlag, Berlin, May 21–22 2012, pp. 355–360.