A Comparative Overview of Robotic Frameworks: ROS and RSB

Mustafa Özçelikörs
Fachhochschule Dortmund, Fach. Info.- und Elektrotechnik
M.Sc. Embedded Systems for Mechatronics
Dortmund, Germany
mozcelikors@gmail.com

***Abstract*—Today, developers are competing against each other to provide the best robotic middleware to the robotics community. In this paper, ROS, a middleware with a huge utility and community support; and RSB, an event-driven middleware that will change our expectations of the communication architecture of robotic middleware, are explained within the context of distributed systems and robotics.**

Keywords—robotics; middleware; framework; ROS; RSB; IDL; comparison; content-based filtering; distributed system; logically unified bus; publish/subscribe.

I. INTRODUCTION

Robotics, in essence, provides useful functions to our daily lifes and to the industry. Over the past century, the scope of the robotics field has been widened significantly, since the usage of robots have been essential. Moreover, challanges in a robotic system is increased to fulfil newly defined more complex tasks. These complex tasks often require interaction with the physical world. With the help of interdisciplinary solutions (image processing, speech detection, behavior algorithms, perception algorithms, sensor fusion, and control systems) we now can overcome the 21st century tasks. However, these solutions increase the complexity of our robot's software architecture drastically and raises the load on the hardware. We therefore need software frameworks and tools that will help us reach our design goals by providing communication layers, parallelization, software packages, debugging and visualization utilities. Thus, with the help of those frameworks and tools, we are able to develop robotic applications efficiently and easily.

Since the most of the robotic applications are based on distributed systems, the requirements and challanges of distributed systems such as scalability, openness, and heterogeneity have to be fulfilled in robotic systems [2]. There are many robotic frameworks developed so far, such as ROS [1], RBS [3], YARP [4], aRDx [5], OpenRTM [6], Orocos [7], Microsoft Robotics Studio [8], which aim to fulfil different requirements and characteristics of a robotic system. For example, the robotic framework RBS is developed essentially to provide a more open platform and to overcome the challanges in cross-compiling [3]. However, none of the frameworks are yet to achieve the best characteristics a system could offer.

In this paper, an informative and comparative research on two of the robotics middleware, ROS and RSB, is presented. The main characteristics and goals, software architecture, and applications and tools of those frameworks are addressed. Additionally, the requirements and needs of a robotic system and how they are issued in ROS and RSB frameworks is discussed.

II. ROBOT OPERATING SYSTEM

ROS (Robot Operating System) is a completely open-source and flexible robotic framework that provides the some services expected from an OS (such as hardware abstraction, low-level device control, message passing between processes, and package management) which is widely used in robotic systems and applications [9][10]. Its main purpose is to create a common platform for robotics where collaborative work of robotic applications are shared and improved. It can be described as an ecosystem of robotics, where large-scale packages, libraries, and services are included [9].

One can use ROS in order to create software applications for robotics, simulate them, and associate them to low-level drivers in a structured manner. For that purpose, ROS provides support to many popular robot platforms such as Pioneer Robots [11], EvaRobot [12], Clearpath robots [13], NAO [14] and many more [15]. It also provides sensor drivers for many rangefinders (such as Lasers, Lidars and Radars), cameras, GPS (Global Positioning System), and IMU (Inertial Measurement Unit) sensors. The main goals and characteristics, software arcitecture, and applications and tools of ROS are detailed in the following part of this section.

A. Main Goals and Characteristics

In [1], when ROS was introduced for the first time, design goals of ROS was said to be being peer-to-peer (P2P), tools-based, multi-lingual, thin, and open-source. However, due to the recent developments in ROS, we could easily extend this list by adding characteristics such as distributed structure, modular design, availability of tools and services, and active community [16]. We can rearrange and extend the characteristics of ROS as follows:

1) Openness: Due to its open-source nature and package based structure, it could be said that ROS provides open services for robotic development. The ease of collaborative

improvement in ROS is definitely one of the many good reasons to use ROS for your robotic system. ROS also provides tools, and an environment to create tools; which makes it even more open. ROS is also a flexible and adaptable system due to its use of open standarts, such as BSD coding standarts, and use of standart debugging tools [17].

2) Heterogeneity: According to [2], heterogeneity of a system is the ability to work on a variety of hardware and software platforms. ROS is currently available for over 100 robots [15] which uses different software platforms such as Ubuntu, OS X (experimental), Android (experimental), and Arch Linux (experimental) [18]. It also provides drivers for many controllers and sensors. Moreover, ROS supports 5 main development languages which are C++, Phyton, Lisp, Octave [1] and Java [46]. It also provides several 3rd party MATLAB interaction tools using Java WebSockets and rosbridge tool. One example of this would be MATLAB-ROS Communication and Control Interface developed in [55].

ROS takes care of the cross-language development issue by using a simple interface definition language (IDL) to describe message passing between modules. IDL files are simple text files that describe the message type and name that is passed between the modules [1]. An example of IDL regarding 3-dimensional acceleration message is illustrated below [19]:

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

In this example geometry_msgs/Vector3 is another message type which includes float type vector which has x, y, and z components.

By the definition given in [2], we could easily say that the heterogeneity is a very crucial aspect of ROS, considering its platform & hardware support, software support, cross-language development solution and multi-linguality.

3) Communicating via P2P: Any robotic system built with ROS is a peer-to-peer network where every processes tend to communicate with each other instead of communicating with a central server [1][20]. In ROS, peer-to-peer connection configuration occurs in XML-RPC, which is a protocol that uses XML to encode and HTTP to transfer data [21][22]. A peer to peer network is given in Fig 1.

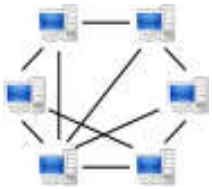

Fig. 1. A peer-to-peer network [20]

By providing this characteristic, they want to avoid having mass network traffic in the central server [1] and share resources with every single peer [20]. The mechanism in which processes find each other at runtime is called *name services*[1]. This topic will be discussed in the Architecture part of this section.

4) Easy to Cross-compile: We know gathering up dependencies and trying to build systems in Unix systems to be hard and overwhelming sometimes. ROS achieved to make a system where standalone libraries have no dependencies on ROS [1]. For building projects, they use the CMake method, which is a cross-platform building tool which compiles systems independently from compilers and platforms [23][1]. To make compiling even more straightforward, ROS uses a build system for its macros and infrastructure called *catkin*, in which CMake's ability to find dependencies and packages is used [24]. Catkin is an improvement on ROS' previous build system *rosbuild*. By using *catkin_make* command, for example, users can easily compile most of the packages automatically, without having to care about the dependency issue.

B. Software Architecture

As stated in this section, ROS uses XML-RPC protocol for its back-end, which is a protocol that uses XML to encode and transfer data within HTTP transport layer [21][22]. However, in order to transfer topic message transportation, ROS uses TCPROS [55], which uses TCP/IP sockets to deliver a message.

In order to develop software using ROS framework, one should understand how different elements in message passing structure are addressed in it. ROS uses a message-driven publish/subscribe architecture between its *node*s[1][25]. A ROS *node* is a process that can communicate with other nodes by publishing or subscribing *message*s[25], using interprocess communication. To handle the complex single-bus message passing between nodes, strings called *topic*s are created which represent the communication channels between nodes, to demonstrate in which channel the data are published and from which channel they are subscribed [26][27][1]. As it is described in part A of this section, ROS uses messages in IDL format, which can be processed by nodes easily. To illustrate this process interaction we can consider Fig 2.

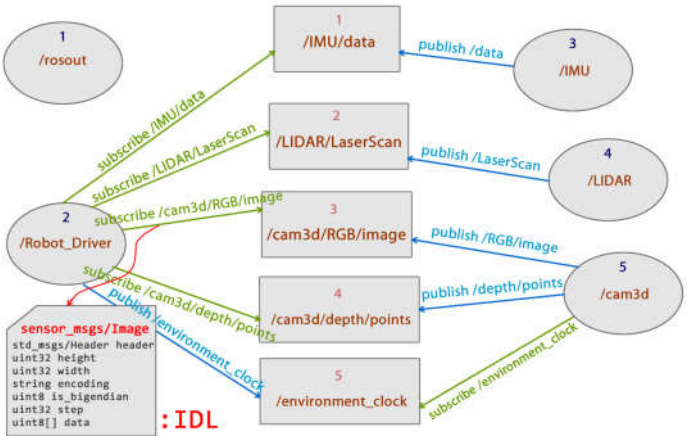

Fig. 2. An illustration of the usage of nodes and topics in ROS

In Fig 2, ROS nodes and topics regarding an example mobile robot is illustrated. Circles represent the nodes (processes) of the system whereas the rectangles represent the topics (message channels). It could be said by looking at the figure that a node can subscribe and publish to more than one topics [1]. To understand the architecture we can see that the node representing the low level architecture of the robot, `/Robot_Driver`, subscribes to sensor topics 1, 2, 3, and 4; while it is publishing the system clock to the `/environment_clock` topic, for other nodes to use. Moreover, each topic carries a type of message. For example, the topic `/cam3d/RGB/image` represents a camera image and it carries a message of type sensor_msgs/Image. Sensor_msgs is one of the many message libraries developed under common_msgs by ROS that carries message types belonging to many types of sensors [28]. ROS also includes primitive type and geometry related message collections such as std_msgs, and geometry_msgs, respectively [29].

In Fig 2, *rosout* can be described as the node ROS uses in order to log messages which is of type rosgraph_msgs/Log. Moreover, ROS uses its own environmental nodes and topics in order to complete the architecture [30]. We are able to subscribe or publish to those topics to handle the ROS environment as well. Moreover, we can also subscribe or publish to simulators (such as GazeboSim [31]) in order to simulate our robot application. With the help of this convenient access to simulator and low level components using ROS nodes, one can also create Hardware-in-the-loop (HIL) simulations using ROS [32].

In most of the distributed applications and robotic applications, it is often required to do a remote procedure call (RPC), which is a request-response protocol [33]. Previously described architecture of ROS at which messages are passed by publishing and subscribing can not satisfy the RPC protocol [1]. To do such a thing, ROS offers *service*s, which manages reply and request messages [34]. A service can only accessed by only one node at a time [1], which supports the RPC type communication. Additionally, just like topics, services have service types [34] which can be managed by using *rosservice* command.

In many applications, we often need to start more than one nodes at a time and configure their parameters. To structuralize the node initialization, ROS offers a tool called *roslaunch* which is an application that can run multiple nodes and configure ROS parameter server parameters by using an XML-based text file (called .launch file) [35]. A simple launch file in ROS would look like the following [36]:

```
   <launch>
<node name="talker" pkg="rospy_tutorials"
type="talker" />
   </launch>
```

C. *Tools and Applications*

ROS offers a set of debugging and visualization tools with which users can visualize their software structure and troubleshoot. To visualize the topic messages, *rqt_plot* could be very useful [37]. By using this tool, one can make graph related to robot positioning, velocity, image and laser data and so on. This would help when simulating and debugging the software of a robot. Moreover, another tool called *rqt_graph* allows us to visualize the work architecture, which can also be useful when debugging the computational tree of the nodes in a system [38].

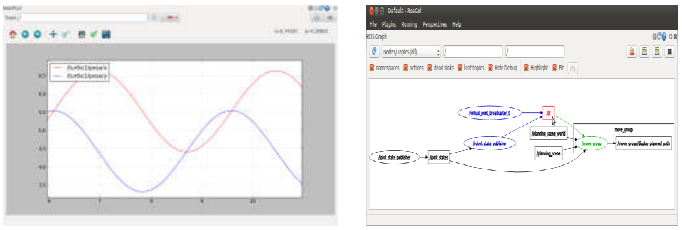

Fig. 3. rqt_plot and rqt_graph tools, respectively [37][38].

To visualize more complex 3D data, ROS offers a 3D visualization tool which is called *rviz* [39]. Rviz is mostly used in PointCloud or Image based applications, but it is very well capable of visualizing robot bases, identifying the transformation frames (*tf* [40]), directions, odometry and many more. Rviz can also import robot models, which are defined using URDF (Unified Robot Description Format). In Fig 4, visualization of position change (odometry) and depth information of a 3D sensor is given, respectively.

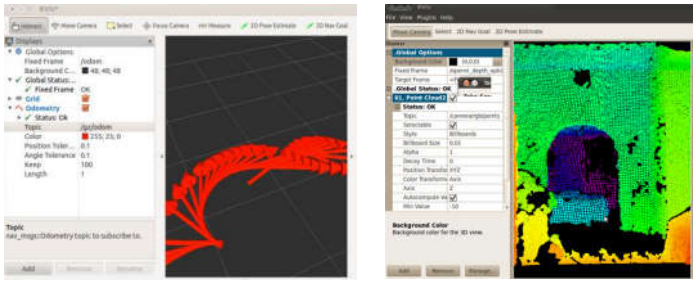

Fig. 4. Different types of Rviz visualization [41][42].

Since robots need to be tested and simulated in order to prevent material loss, ROS offers support to many simulators such as GazeboSim (3D)[31] and STDR Simulator (2D)[43]. With the help of these simulators, physical environments can be modelled, sensor behaviors can be described and the robot behaviors can be simulated. The example of an office environment built using GazeboSim is given in Fig 5 [10].

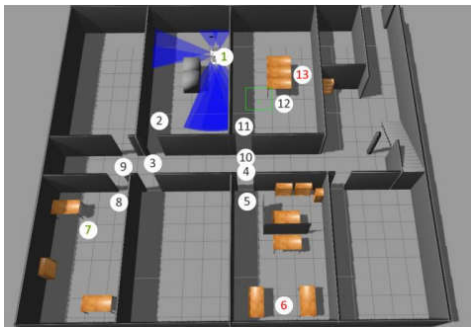

Fig. 5. A robot in an office environment built using GazeboSim (ATEKS project) [10].

III. Robotics Service Bus

Robotics Service Bus (RSB) is a robotic framework alternative to ROS, which handles its communication using a logically unified bus, and is event-driven [3]. Even though its model-based approach comes from its event-driven nature, it does not provide model based code generation tools. Rather, it uses event-driven architecture to ease the information sharing. It uses message passing using channels [44]. RSB can integrate with other middleware and is designed specifically for collaborative research [3]. In order to achieve that, RSB focused on the hierarchical structure and back-end of its architecture, in order to provide a solution to improve the existing middleware. RSB's event-driven communication resembles ROS's message-driven type-based publish/subscribe architecture [1][25], but RSB aims to present a more hierarchy in its channels [3]. In addition, RSB presents content-based filtering visible to the transport layer, aiming to optimize more [3]. We can by no means say that RSB has the huge community support that ROS does. However, contributers are starting to build up a set of utilities and tools, and the ROS/RSB bridge described in [45] provides transport-level integration with ROS Client API and RSB back-end, allowing some scripts of ROS to be integrated with RSB. Following part on this section will cover main goals and characteristics, software architecture, and tools and applications of RSB middleware.

A. Main Goals and Characteristics

RSB presents an approach to allow collaborative research by fulfiling the requirements of such systems, such as openness, and scalability [3]. Its main goals are to allow sufficient portability (openness property), introspection support, integration with other middleware, and allowing the framework to be altered for optimization (low framework lock-in) [3]. In [46], the group who are currently developing and maintaining RSB, CoR-Lab points out some of the issues related to other middleware. Most crucial issues can be listed as the following [46][3]:

- Frameworks have large footprint that does not satisfy the needs of some embedded systems.
- Insufficient integration with other frameworks
- Insufficient error handling strategies
- Unalterable framework structure (high framework lock-in) or unspecified architecture that makes them difficult to be optimized.

RSB takes these issues into account and presents a new approach to robotic frameworks, which reduces the framework lock-in, allows integration of different systems, and still provides sufficient functions for a robotic task [46][3]. The following part is dedicated to explain how some characteristics in ROS are addressed.

1) Openness: Openness is a system's ability to be easily adaptable, portable, flexible, and interoperable which means that it should be easily changable, satisfy certain standarts, run on different hardware and software platforms, be easily managable and operate with other systems from the same scope [2][3]. To address these properties, RSB brings many new approaches to the robotic framework area. First of all, RSB uses the RST project [47] for its message definitions, which is a type specification that uses IDL externally, to have a standart data type in its applications [3][47]. By using IDL externally, they provide a system which is not limited to RSB only. Secondly, RSB provides interoperability support by implemented bridge software with other robotic middleware such as ROS-RSB bridge [3]. Finally, in order to overcome the portability issue, RSB introduces dependency graph which does prevent it from having multiple layers of dependencies. Instead, RSB has a dependency graph that has standart libraries [3], which ease the drudgery of cross-compilation.

2) Scalability: A systems ability to scale means that the stability of them should be independent from the number of processing nodes. In order to scale the size of the systems, efficient processing is crucial [3]. RSB includes three different transports which are Spread-based transport, network-based transport (using TCP), and in-process transport [3]. In order to manage complex distributed systems, RSB's Spread-based transport provide wide area group communication system [48]. As for the performance, although benchmark tests in [3] showed that ROS has the most efficiency in all of the tested middleware, RSB claims that their roundrip performance is sufficient for robotic applications.

3) Heterogeneity: RSB provides full support for C++, Java, Phyton, Lisp, and partial support for Matlab [46], and most of the RSB packages are available for operating systems such as Linux, MacOS, and Windows; which shows that RSB is able to run on many hardware and software platforms [49].

4) Interospection Support: RSB provides an interospection namespace [50] and interospection based tools to make us able to keep track of the communication, which are used to log and replay the data streams with a common format [3]. These tools are used when analyzing the timing of the communication [3].

B. Software Architecture

As mentioned, RSB presents an event-driven and message-oriented communication architecture where a logically unified bus which is established across several transport layers and has channels according to hierarchy [46]. In Fig 6, RSB API is given, where content-based message filtering, event processing, and Converters are located in Extension-Point API, and high level communication and infrastructure are located in User- Level API. RSB's architecture is structured using three models.

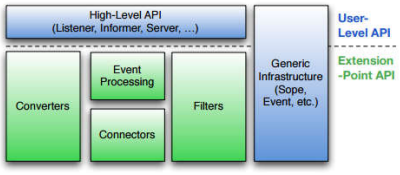

Fig. 6. RSB API [46].

1) Event Model: Events are actions that trigger data exchange [3][46]. An event has many components such as an ID, a payload, a destination scope (reciever of event notification), a causal vector; and an event comes with a meta data class attached to it [44]. As an analogy, we can relate the terms of the event model to a timed automata. Payload of an event is expression of actual transmitted data, whereas the causal vector represents the expression of the emitted (sent) trigger [3]. Meta data class, however, is responsible for providing timestamps to make the event traceable [3].

2) Notification Model: The Notification Model of RSB describes the data transmission and reception between events [3]. Units which handle event sending are called *Informers*, whereas units which handle event reception are called *Listeners* [3]. We could relate informers and receivers to ROS' publisher and subscriber nodes. *Participant*s, which participate in data transmission and reception first has to connect to the so-called logically unified bus via a *Channel.* This channel is realized by many *Connector*s which represent the lowest level in RSB communication which exchange *Notification*s in a transport layer (see Section III Part A) [3]. Connectors in the system uses *Converter*s to serialize (or deserialize) the event to exchange a *Notification*s on different transport layers [3]. As eloquently illustrated in Fig 7 by [3], we can think of *Notification*s as the event descriptions in transport layers.

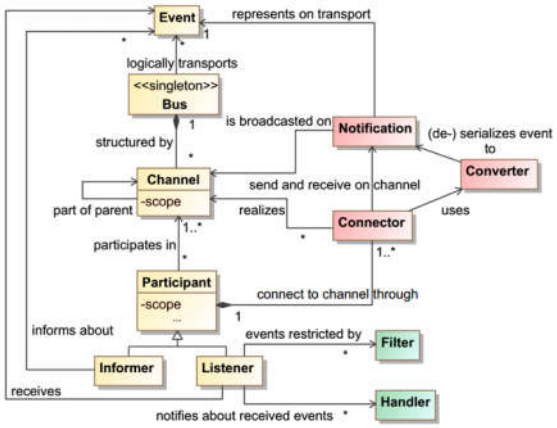

Fig. 7. Conceptual overview of RSB model [3].

When dealing with hierarchy in its channels, RSB uses the *Scope* approach, compared to ROS' topics. [3]. With this approach, the implication is that a scope could only be visible to higher hierarchial scopes and to the scope itself in the channel [46]. Fig 8 illustrates the hierarchy in channels, which shows that notifying `/radar/accel/` will appear to `/`, `/radar`, `/radar/accel`, but not to `/radar/accel/x`.

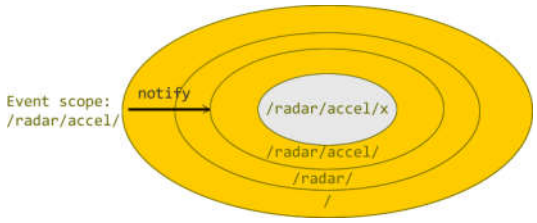

Fig. 8. Event scope example

3) Observation Model: Observation model of RSB is the model that provides a solution to unrelevant event reception and also to event handling [3]. Event restriction is done by *Filter*s attached to the Listeners [3]. In this model, *Filter*s are responsible for restricting the events by running series of content-based filtering algorithms. *Handler*s, however, are used simply to handle event listening, i.e in order to act upon a received event, we use Handlers as callback functions [3].

C. Tools and Applications

RSB provides some utilities such as *logger*, *introspect*, *send*, *call* that runs from the command line [51]. These tools are used for logging real-time data, analysing RSB systems, sending events, and calling server methods, respectively. Additionaly, *web* tool allows users to inspect an RSB system by providing a web interface [51]. RSB also provides time synchronization via command line. The *rsb_timesync* command could be used to synchronize different scopes of a system [51].

As stated before, RSB provides bridge tools for other frameworks which would make us run some third party tools created for other frameworks [46]. Moreover, RSB provides support for some third party applications such as Vicon, ISR, XTT and Pamini to handle operations such as motion detection, speech recognition, task handling and modeling [46].

So far, most of the RSB's functionality was used and tested in a couple of applications using the robots NAO [14], Oncilla (AMARSi project) (seen in Fig 9) [52] and a kinetic robot manipulator [53][3]. Moreover, RSB is being used for educational purposes at Bielefed University [3].

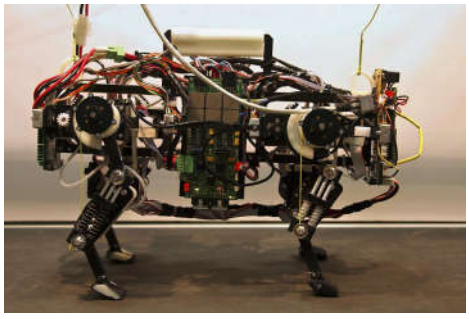

Fig. 9. The AMARSi Project (using Oncilla) [52].

IV. Comparison And Summary

This section is dedicated to summarizing the differences between ROS and RSB, two middleware that are in demand and was discussed in sections II and III. Table 1. shows the differences between middleware ROS and RSB compared by using different properties. In the table, it is seen that both middleware have strong sides and drawbacks. At the first glance, it is seen that the communication side of RSB looks more advanced, whereas the ROS shine out when it comes to the utility support.

Table 1. ROS-RSB Comparison Table (Table derived from [46][3][15][54]) - (*:experimental)

	ROS	**RSB**
Approach	Message-driven Publish/Subscribe	Event-driven, Message-oriented
Topology [46]	1:1	m:n
Transport Layer	HTTP (XML-RPC), and TCP (TCPROS)	Spread-based, TCP-based, in-process transport
Channel Hierarchy	No	Yes
Centralized [46]	Yes (master)	No
Filtering	Yes (message filters)	Yes (content-based event filters)
IDL type message	Yes	Yes, external IDL (RST project)
Alterability	Low	High
OS Compatibility	Ubuntu, OS X *, Arch Linux *, Android *	Linux, OS X, Windows
Programming Language Support	C++, Python, Lisp, Octave, Java [46], Matlab (3rd party apps, rosbridge)	C++, Phyton, Lisp, Java, Matlab (via Java) [46]
Roundtrip Performance [3]	Sufficient, Higher than RSB	Sufficient [3]
Logging/Bagging Tools	Yes, rosout and rosbag	Yes, rsb logger
Simulator Support	GazeboSim, STDR, and more	(Not known)
Robot/ Sensor Support	At least over 100 robots/sensors [15][54]	(Not known)
Community Support	High	Low
# of Tools Provided	Relatively High	Average
Resources	ROS API, ROS Wiki, ROS Exchange	Only RSB API

V. Conclusion

Over the past decade, many robotic middleware were developed and offered to engineers. The open-source software community have also contributed to the progress and development of robotic middleware greatly. On one hand, RSB brings lots of innovations to the future of robotic field i.e model-driven approach with high optimizability, content-based event filtering and communication hierarchy. On the other hand, one might consider ROS to meet the requirements of his system as it provides drivers and interfaces for many robots, sensors and simulators that are supported by a huge community. As the robotic systems become more complex, there is no doubt that the competition in this field will improve existing middleware and bring new ones. Moreover, the evolution of the cyber-physical systems such as robotic systems will allow new approaches and methods to arise. Therefore, it is important that engineers make a decision on the middleware or tools they use by evaluating the existing solutions, considering drawbacks of each solution and choosing a solution that provides the optimal use for their application.

Acknowledgement

This study is completed as a student research project for Distributed and Parallel Systems lecture, under Master's program Embedded Systems for Mechatronics in Dortmund University of Applied Sciences and Arts.

References

[1] M. Quigley, B.Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A.Ng, "ROS: an open-source Robot Operating System," in ICRA Workshop on Open Source Software, 2009.

[2] R. Hobo, http://www.remcohobo.nl/DIA/distsysreq.pdf , 2004.

[3] J. Wienke, S. Wrede, "A Middleware for Collaborative Research in Experimental Robotics", IEEE/SICE International Symposium on System Integration (SII), 2011.

[4] P. Fizparick, E. Ceseracciu, D. E. Domenichelli, A. Paikan, G. Metta, and L. Natale, "A middle way for robotics middleware", Journal of Software Engineering for Robotics, 2014.

[5] T. Hammer, B. Bauml, "The highly performant and realtime deterministic communication layer of the aRdx software framework", 16th Advanced Robotics (ICAR), 2013.

[6] G. Biggs, T. Kotoku, "Tutorial workshop I: SICE 2011 OpenRTM-aist tutorial", SICE Annual Conference (SICE), 2011.

[7] The Orocos Project, http://www.orocos.org/

[8] Microsoft Robotics Developer Studio User Guide, https://msdn.microsoft.com/en-us/library/bb648760.aspx

[9] About ROS, http://www.ros.org/about-ros/

[10] M. Özçelikörs, A. Çoşkun, M.G. Say, A Yazıcı, U. Yayan and M. Akçakoca, "Kinect Based Intelligent Wheelchair Navigation with Potential Fields", INISTA 2014 International Symposium on Innovations in Intelligent Systems and Applications, 2014.

[11] Adept Mobile Robots, http://www.mobilerobots.com/Mobile_Robots.aspx

[12] EvaRobot, http://www.evarobot.com, 2015.

[13] ROS Wiki, "ClearPath Robotics", http://wiki.ros.org/ClearpathRobotics

[14] NAO Robots, https://www.aldebaran.com/en

[15] ROS Wiki, "Robots Using ROS",http://wiki.ros.org/Robots

[16] Is ROS for Me?, http://www.ros.org/is-ros-for-me/

[17] ROS Wiki, "ROS developer's guide", http://wiki.ros.org/DevelopersGuide

[18] ROS Wiki, "ROS Installation", http://wiki.ros.org/ROS/Installation

[19] ROS Wiki, "geomertry_msgs", http://wiki.ros.org/geometry_msgs

[20] peer-to-peer (Definition), http://searchnetworking.techtarget.com/definition/peer-to-peer

[21] G.G. de Rivera, R. Ribalda, J. Colas, J. Garrido, "A generic software platform for controlling collaborative robotic system using XML-RPC",IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2005.

[22] CMake Project, https://cmake.org/

[23] ROS Wiki, "Catkin Conceptual Overview", http://wiki.ros.org/catkin/conceptual_overview

[24] ROS Wiki, "Understanding Nodes", http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

[25] Core Components, http://www.ros.org/core-components/

[26] ROS Wiki, "Topics", http://wiki.ros.org/Topics

[27] ROS API Documentation, "sensor_msgs", http://docs.ros.org/jade/api/sensor_msgs/html/index-msg.html

[28] ROS API Documentation, http://docs.ros.org/jade/api/

[29] ROS Wiki, "rosout", http://wiki.ros.org/rosout

[30] GazeboSim, http://gazebosim.org/

[31] S. Ayasun, A. Monti, R. Dougal, and R. Fischl, "On the Stability of Hardware in the Loop Simulation", http://vtb.engr.sc.edu/vtbwebsite/downloads/publications/hil_imacs02.pdf

[32] What is Remote Procedure Call?, http://searchsoa.techtarget.com/definition/Remote-Procedure-Call

[33] ROS Wiki, "Services", http://wiki.ros.org/Services

[34] ROS Wiki, "roslaunch", http://wiki.ros.org/roslaunch

[35] ROS Wiki, " XML", http://wiki.ros.org/roslaunch/XML

[36] ROS Wiki, "rqt_plot", http://wiki.ros.org/rqt_plot

[37] ROS Wiki, "rqt_graph", http://wiki.ros.org/rqt_graph

[38] ROS Wiki, "rviz", http://wiki.ros.org/rviz

[39] ROS Wiki, "tf", http://wiki.ros.org/tf

[40] Wheeliebot: A Server Controlled Video Streaming Indoor Navigation Robot, http://thewebblog.net/portfolio/projects_wheeliebot.php

[41] Kinect on ROS, http://estanciaitesm.blogspot.de/

[42] ROS Wiki, "STDR Simulator", http://wiki.ros.org/stdr_simulator

[43] RSB 0.13.0 Documentation, "Concepts", http://docs.cor-lab.org/rsb-manual/trunk/html/concepts.html

[44] RSB 0.13.0 Documentation, "Ros Integration", https://code.cor-lab.org/projects/rsb/wiki/RosIntegration

[45] CoR-Lab, "RSB-Robotics Service Bus A Lightweight Event-driven Middleware", https://code.cor-lab.de/projects/rsb/repository/rsb-talks/revisions/master/entry/overview/talk.pdf, 2011.

[46] CoR-Lab, "RST Project", https://code.cor-lab.de/projects/rst

[47] Y. Amir and J. Stanton, "The spread wide area group communication system", Tech. Rep. CNDS-98-4, Center for Networking and Distributed Systems, The Johns Hopkins University, 1998.

[48] RSB 0.13.0 Documentation, "C++ Installation", http://docs.cor-lab.org/rsb-manual/trunk/html/install-cpp.html

[49] RSB 0.13.0 API, "rsb::interospection", http://docs.cor-lab.org/rsb-cpp-api/trunk/html/namespacersb_1_1introspection.html

[50] RSB 0.13.0 Documentation, "Tools", http://docs.cor-lab.org/rsb-manual/trunk/html/tools.html

[51] AMARSi Oncilla, https://www.amarsi-project.eu/oncilla

[52] M. Rolf and J. J. Steil, "Continuum Kinematics Simulation of the Bionic Handling Assistant", in IEEE Int. Conf. on Robotics and Automation, (St. Paul, Minnesota, USA), IEEE, 2012, submitted.

[53] ROS Wiki, "Sensors supported by ROS", http://wiki.ros.org/Sensors

[54] MATLAB-ROS Communication and Control Interface, http://thewebblog.net/portfolio/projects_matlabrosinterface.php

[55] ROS Wiki, TCPROS, http://wiki.ros.org/ROS/TCPROS