R5-COP

Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems

Middleware Assessment (initial)

Viatcheslav Tretyakov, Nikolai Ensslen (SYN),
Sönke Michalik, Jan Wagner, Rainer Buchty (TUBS)

Project	R5-COP	**Grant Agreement #**	621447
Deliverable #	D31.11	**Dissemination Level**	PU
Contact Person	Viatcheslav Tretyakov	**Organization**	SYN
E-Mail	vtretyakov@synapticon.com	**Date**	27.02.2015

Document History

Document History			
Ver.	**Date**	**Changes**	**Author**
0.1	29.01.2015	Initial document creation	Viatcheslav Tretyakov
0.2	26.01.2015	First batch of metrics added	Viatcheslav Tretyakov
0.3	02.02.2015	More metrics added	Viatcheslav Tretyakov
0.4	11.02.2015	Input added	Sönke Michalik
0.5	17.02.2015	Editing document	Jan Wagner
0.9	24.02.2015	Formatting, Typesetting	Rainer Buchty, Jan Wagner
1.0	01.03.2015	Input added	Sönke Michalik
1.1	10.03.2015	Modifications requested from review	Sönke Michalik

Note: Filename should be
"R5-COP_D##_#.pdf", e.g. „R5-COP_D91.1_v0.1_TUBS.pdf"

Fields are defined as follows

1. **Deliverable number *.***
2. **Revision number (should not be required with LaTeX ... use versioning!)**
   - **draft version** **v**
   - **approved** **a**
   - **version sequence (two digits)** ***.***
3. **Company identification (Partner acronym)** ***

Table of Contents

List of Figures

List of Tables

List of Acronyms

Abbreviation	**stands for**
μC/OSII	real-time deterministic multitasking kernel for microprocessors
API	Application Programming Interface
BSD	Berkeley Software Distribution License
CAN	Controller Area Network
CORBA	Common Object Request Broker Architecture
EtherCAT	Ethernet for Control Automation Technology
GPL	General Public License
HAL	Hardware Abstraction Layer
HTTP	Hypertext Transfer Protocol
LGPL	"Lesser" General Public License
MARS	Multi-Agent Robotic Systems
MAS	Multi-Agent systems
RDF	Resource Description Framework
ROS	Robot Operating System
RSF	Robotic Software Framework
RTAI	Real Time Application Interface
RTLinux	Real-time Linux
SLAM	Simultaneous Localization and Mapping
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language

1 Introduction

1.1 Summary (abstract)

Robotic systems require significant interaction and coordination of hardware and software elements. In the context of software engineering, the concept of middleware earned a very strong role in the entire software development process. In robotic applications the use of a middleware can help improving the organization, the maintainability and the efficiency of the code that controls the robot.

Robotic systems are usually complex systems built on many different hardware and software components, as sensors and actuators as well as planners and control algorithms. In general, on each robot runs a software that is responsible for reading sensors data, extracting the needed informations from them, computing the sequence of actions to accomplish a given task and controlling the actuators to execute the actions. Using a custom approach, there will be a single monolithic application that will handle all these tasks, making code maintenance hard and preventing every form of code reuse and sharing between different projects. Such a scenario, with many hardware and software components that needs to communicate and collaborate to reach a goal, is exactly where a middleware can help improving the organization, the maintainability and the efficiency of the code. The whole application can be structured into many little concern separated tasks, as ”get a sensor reading”, ”extract features from some data”, ”drive the motors to some speed”. Different components can exchange data using a common communication channel provided by the middleware, using interfaces that are consistent between different applications. In this way, it becomes really easy to share and reuse code among different projects, or change an algorithm to get some functionality as it is only necessary to keep the same interface. As an example, if you need to switch from a proximity sensor to another, it is possible to write a new component that share the same interface and update it without modifying the rest of the application. This concept can be extended to large and complex applications, in which using a middleware can clearly improve the overall code organization and reduce the programming effort.

Why use middleware?

Middleware is useful for a number of reasons when designing large, distributed systems. Some relevant aspects:

- **Portability:** middleware offers a common programming model across programming language and/or platform boundaries, as well as across distributed systems by providing a uniform communication API. Thanks to this, it is possible to make cooperative applications developed in different languages (e.g. Java and C++) and executed on different operating systems (e.g. Windows and Linux) without any specific effort by the programmer.
- **Reliability:** middlewares are developed and tested separately from the final application. This allow the application programmer to abstract low-level aspects and use (and re-use) a well-tested library.
- **Managing complexity:** low-level aspects could be managed by suitable libraries that abstract specific operating system or hardware aspects. This simplifies development and reduces the probability of errors.

1.2 Purpose of this document

D31.11 will show metrics and criteria to quantitatively assess middlewares. Aspects like modularity, composability, configurability, reusability, real-time capability, scalability and stability will be considered.

1.3 Partners involved

Partners and Contribution	
Short Name	**Contribution**
SYN	Main contributor to this report
TUBS	Consulting

2 Middleware Assessment Metrics

To assess middlewares in terms of modularity, composability, configurability, reusability, real-time capability, scalability and stability it is required to define a range of specific metrics.

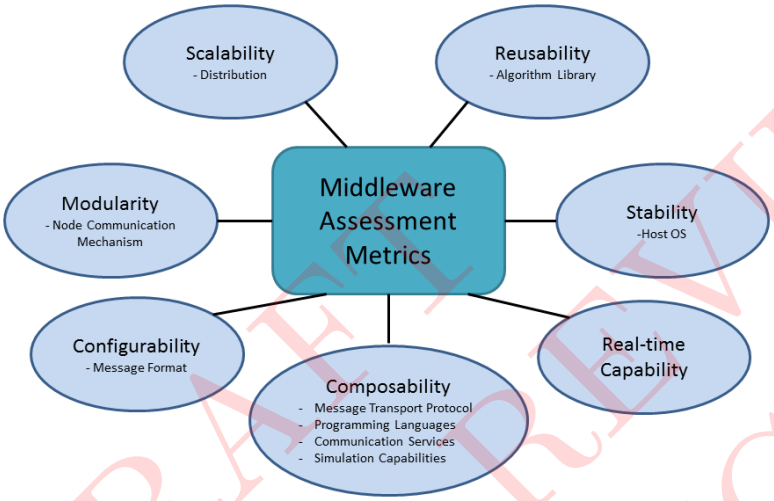

Figure 1: Middleware Assessment Metrics

The assessment shall consider the special requirements on middleware frameworks for industrial applications and embedded systems.

2.1 Modularity

Modularity is the key aspect to enable the application of the divide-and-conquer paradigm. Following the divide-and-conquer paradigm means breaking down a complex problem into two or more sub-problems until the sub-problems become simple enough to be solved. In the robotic domain it means simple tasks are mapped onto nodes which represent the building blocks for more sophisticated functionality. The middleware should facilitate the creation of sophisticated features by providing a feasible communication infrastructure.

2.1.1 Node Communication Mechanisms

The node communication mechanisms can differ in range of simple message passing between nodes to more advanced mechanisms such as ports, topics, events, services and properties. The communication mechanisms will influence the coupling of nodes, performance and ease of use of the middleware framework.

- **Simple message:** This is the simplest mechanism, and is a one-to-one communication where a node sends an asynchronous message and another node receives it. It is the most basic communication mechanism, from which all the other mechanisms are derived. The messages also tend to include metadata about the message, such as the intention, content format, and content ontology.

- **Ports:** These are mechanisms that allow nodes to communicate with a low degree of coupling. The nodes are made up of two types of ports: in-ports and out-ports. These ports can be interconnected with another port through one or several connections (one-to-many). They can be created and destroyed dynamically, which introduces great flexibility. Each node is designed to read and to write in its ports in-dependently of which node is connected in execution. The messages are usually stored in buffers in the in-ports and out-ports. The messages can be read by the receiver in two ways: by checking the state of the buffers and collecting new messages (polling) or by asynchronous method invocation (callback).
- **Topics:** This is an asynchronous communication mechanism that follows the Publish/Subscribe model and allows a many-to-many communication to be made whilst maintaining low coupling. A topic represents a centralized channel where all the nodes connected to it receive any message that is sent by a node. This channel represents logic centralization; that does not imply a bus at the transport level. This logic bus could be implemented by various means in the transport layer, for example, by means of multiple point-to-point connections in an Ethernet network and TCP/IP, or, a physical bus type connection in a CAN network.

  In many aspects topics are similar to ports, nevertheless one of the main differences is that a topic does not pertain to any node, but events. These are one-to-many asynchronous communication mechanisms that allow a low degree of coupling to be maintained between nodes. They are also known as the observable/observer patterns. The emitting node emits messages to all the subscribed nodes, which, although very similar to the ports mechanism, differs mainly in that: the connection concept does not exist; an event is implicitly asynchronous; and the subscribed nodes always deal with events by means of callbacks.
- **Services:** This is a communication mechanism that allows the remote execution of a procedure; the remote procedure call (RPC). Two messages come into play: Request and Response. The message request is sent by the client node and indicates what procedure is desired to be executed and its arguments. The Response message is sent by the server node with the result of the operation. It is a typically synchronous procedure where the client remains blocked while the response is awaited.
- **Data/message centric communication:** The communication of an middleware framework can be splitted into message and data centric transfers. In robotic system with high amount of data transfers (e.g image data) it can be usefull to send data packages seperatly.

2.2 Composability

The composability of an middleware framework in terms of compatible protocols, programming languages or services is an important aspect since it affects its flexibility and versatility. The interfacing capabilities to other processing systems in a robotic environment is essential for most robotic applications. Also the simulation capabilities are an critrion to select a prober middleware framework.

2.2.1 Message Transport Protocol

The message transport protocol is a crucial characteristic of middleware frameworks that is also related to the communication layer. The supported transport protocols like TCP, UDP, EtherCat, CAN-open, HTTP, SSL, CORBA will significantly change the performance of the middleware framework.

2.2.2 Programming languages

A middleware framework can support one or multiple programming languages. On one hand the used programing language can directly influence the performance of the robotic system and should be selected to match the operation platform performance. On the other hand it directly influences the time required to create new software components.

The greater the number of supported languages, the more flexibility is contributed to development, which in turn results in more libraries and projects that can be reused. The language employed brings major consequences on performance, the real-time capabilities, the transport mechanism, and some agent capacities, such as mobility. The most widely accepted language in robotics application frameworks is C/C++ since it offers a good balance between the access to devices, sensors and actuators at low level, while still offering a sufficient level of abstraction to create complex distributed system architectures. This means that it presents an adequate language for both low-level control tasks and high-level program-ming. On the other hand, many algorithms in robotics deal with problems with a high level of abstraction which is why it is also recommended that the system support high-level languages such as Java, Python or MATLAB. Nevertheless, most platforms tend to offer a wider range of languages, to prevent programming language restrictions to be an application barrier when deploying robotic systems.

2.2.3 Communication Services

Services are used to coordinate the communication between nodes. Naming Services, Lookup Services, Discovery Services provide features to set up the data transfer.

Naming service

This is a global service typical in hybrid P2P architectures, otherwise known as the White Pages service, which allows the localization of nodes, and other global system resources such as topics, from a name. Specific node resources, such as ports, properties, services and events, seldom have a global name managed by the naming service. Generally, the middleware framework provide naming-service mechanisms. Some middleware frameworks have characteristics of a more advanced nature such as “Name Pushing” and the relative addressing of agents or resources of the system. These mechanisms are crucial for the prevention of name conflicts when nodes are instantiated in different spheres of the system. In this case although they can both have the same code, they are in different contexts, which is why a relative name references different agents or resources. These complementary mechanisms related to the naming service boost the flexibility of the resources that can be referenced:

- **Renaming (Remapping):** During deployment, this mechanism allows all the references that exist in the logic of a node of the system resources (nodes, topics, services, etc.) to be substituted by others. It is highly useful in obtaining the correct integration of modules. For example, if two modules implemented by different development teams were designed to read and write in a topic, and each team chose a different name for the topic, neither node would manage to communicate. The problem becomes more complicated when more modules are involved. There are two solutions to this problem: change the implementation and change the name of the affected topics; or use the renaming mechanism, which allows the final value of the deployment configuration to be kept in a file which will take the references to the system resources. This last solution is called Renaming when it is a built-in feature of the RSF.
- **Relative and absolute naming (Namespaces):** This is the capacity to represent hierarchies in the names. It allows the code of a node to be referenced by other resources in an absolute manner

(namespace + name) or a relative manner (name). It is an important tool for creating an organized design and clean code. It is mainly useful when it is used in conjunction with Name Pushing.

- **Name pushing:** This is a particular form of renaming. It is a mechanism that allows a group of resources (nodes, topics, etc.) to be instantiated in a specific namespace at the moment of deployment. This mechanism allows conflicts in the resource names to be prevented. When name pushing is carried out, the interconnections of the nodes can change, since all the references to resources with relative names will only be sought in the present namespace, whereas the absolute references to resources will remain unaffected by name pushing and will continue to be referenced to the same resources.

Lookup service

This service is peculiar to Hybrid P2P architectures where a central or master node exists which contain special information about the whole system. It is also known as the Yellow Pages service, where the central agent or node acts as a directory of the existing resources. The system agents can consult this directory and look for other agents that offer certain services or that fulfill certain properties. Not all the middleware frameworks implement the lookup service. This is a fundamental characteristic for the creation of a robust system. For example, where certain nodes cease to function or cease to offer a particular service or functionality, the system must be able to replace these nodes by searching for substitutes that offer similar services or functionality.

Various situations can be solved without a lookup service if every agent node has previous knowledge of all the capacities of the remaining agents. In short, this lookup service is shown to be essential in a dynamic situation where the services available are subject to the runtime context: energy, priority of running tasks of each agent, physical damages in sensors, etc.

Discovery Service

This is a service that is intrinsically associated with pure P2P distributed architectures which enables a node that offers a certain service to be found. In pure P2P architectures, a central node where the services are registered (Yellow Pages) does not exist, which is why this information must be distributed across the existing nodes. In other words, the discovery service is the pure distributed version of the lookup service: it finds agents that match a specific set of characteristics. As each node can be aware of the existence of different services offered by other nodes, a service search protocol is necessary. This protocol is called the discovery service. It could be useful in Collective Robot Swarms, in certain Robots working in Ambient Intelligence Environments, or in a team of heterogeneous robots. All these applications are subject to network connectivity problems. On a hybrid P2P architecture, losing the master node could be critical for the system.

The discovery service is a very dynamic solution when new robots must be dynamically incorporated into a team, when connectivity problems can occur, or when the availability of robots is not ensured, due to robot damage, energy autonomy, etc.

2.2.4 Simulation Capabilities

These permit modeling, prototyping, and simulation of the final system to be generated, thereby saving both time and expense. They also serve as an early test of viability of the solutions, which may prevent situations of malfunction, conflict, etc. These tools stand out for their capacity to express mathematical concepts and for the possibility of carrying out simulations on these models, thereby obtaining results which can be interpreted for an improvement in the system design. These tools usually offer the possibility of generating graphs and also permit simulations of the robotic system in virtual worlds with rigid

solid dynamics. Some free tools which deserve a mention include: OpenRave, Stage, UsarSIM, Gazebo, and Breve.

2.3 Configurability

A major concern of a middleware is to support as much use-cases as possible. To facilitate this essential features of the middleware need to be configurable in order to meet the requirements of the use-case.

2.3.1 Message format

The message format of a middleware framework determines the interoperability between different platforms and languages. New formats should be easily definable, to support new components with their specific message content without too much programming efforts. Text formats, such as YAML, JSON or XML, can be interpreted by parsers and comprehensibly analysed, while still readable to humans. Binary formats outperform text-formats with regards to communication bandwidth, memory and computational requirements.

Hence the selected message format affects performance, energy consumption and introspection facilities.

2.4 Real-time capability

The majority of robotic systems have some type of real-time constraints. These restrictions are problematic in distributed software architecture. Efficiency is also a common requirement, especially when a robotic system has limited communication and computation capacities. Hence the design of the architecture must consider the use of software, hardware, communication mechanisms and protocols that guarantee compliance with these restrictions.

2.5 Stability

Robotic systems are very complex systems which are developed over years. To maintain compatibility over years stable APIs are necessary.

2.5.1 Host OS

The operating systems offer a hardware abstraction layer (HAL) that simplifies the development of applications enormously and encourages the reuse of hardware and software. Classically, the software architectures in robotic systems have undergone ad hoc development and the use of an operating system has not always been necessary. Lightweight operating systems have usually been employed (RTKernel, FreeRTOS, QNX, etc.) that are specifically adapted to the hardware and to the devices and peripherals used. Some of the most important characteristics supporting these operating systems are multitasking and real-time restrictions; highly interesting aspects in robotics. They also possess controllers for a group of specific devices that allow them to access typical peripherals such as Ethernet and CAN. A disadvantage of these systems is that they have a smaller range of off-the-shelf libraries and drivers for devices and peripherals.

On the other hand, the middleware frameworks allow very diverse, complex robotic systems to be developed. In order to maintain reusability they need to rely on the support of a high level of abstraction

in the Operating Systems and on a large quantity and diversity of off-the-shelf libraries and drivers for devices and peripherals. The general-purpose operating systems which are the most highly developed in these aspects are found in the world of PCs: Linux, Windows and OSX are some examples. As a disadvantage, these systems can be considered as heavyweight since they require greater hardware resources. Nevertheless, in many cases, this does not prevent them from being able to support tasks with real-time restrictions (RTLinux, RTAI). Sometimes a framework is supported on a Virtual Java Machine, which allows the framework to isolate itself from the operating system since the majority of operating systems have support for some of these VMs (Virtual Machines). A disadvantage is the loss of control of the hardware and the physical platform; a fundamental aspect in robotics.

2.6 Reusability

2.6.1 Algorithm library

The robotic algorithms are often the objective of an middleware framework to provide generic and reusable algorithms and functionalities in the field of robotics. Those algorithms are designed for different levels of abstraction: from a low level, such as those related to kinematics, control, robot perception, Bayesian estimation up to others of a high level such as planning, human interaction, robot learning, navigation algorithms, motion planning, Bayesian Filtering, and SLAM. These abstract algorithms are usually built over the Robotic Hardware Interfaces or other low-level robotics algorithms. This software is usually provided in the form of libraries or components that can be instantiated as nodes of the system.

Some middleware frameworks, such as Player, do not show a clear line between these algorithms and device drivers since both are usually wrapped behind a stable and known programming interface to promote reutilization.

2.6.2 License

The license is a crucial factor for the success of development frameworks. There have been multitudes of proprietors of unsuccessful frameworks in the sphere of robotics. The license is also significant in the scope of the assessment on one hand since Open-Source projects allow an understanding of how the framework functions, and facilitate the creation of a more powerful community, which together form a source of new ideas. On the other hand possible license fees have an impact on the financial calculation in the exploitation phase.

2.7 Scalability

2.7.1 Distribution

The modular nature of middleware architectures enable scalability features in terms of distribution over multiple machines in the middleware network. One of the major scale factor of middleware frameworks is the number of processing modules or nodes.

The node configuration of an robotic system using ROS is shown in Figure 2.

Hence, the active processing nodes and as a result the processing load of a single machine can be reduced by extending the middleware network.

The system itself can be scaled by using multiple processing cores or processing threads.

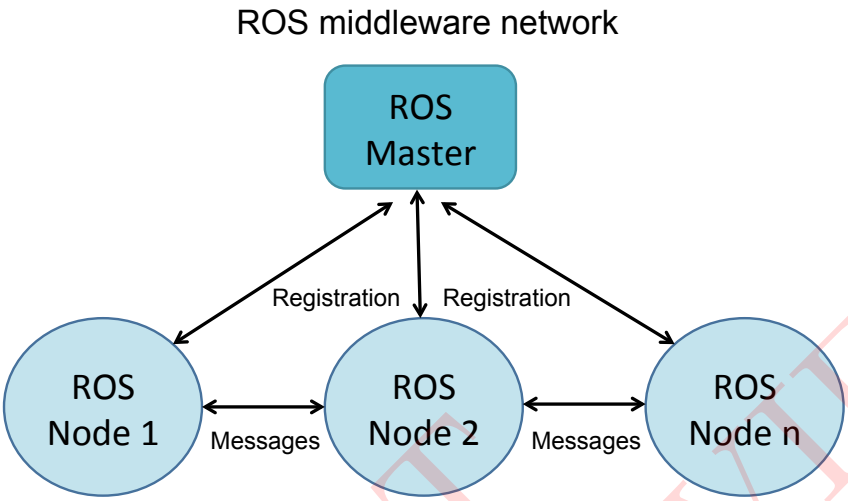

Figure 2: ROS node configuration

On the other hand the overall size of the middleware can be reduced by using a customized set of active nodes that are required for the specific robotic system.

3 Summary of Middleware Assessment Metrics

This chapter summarizes the metrics to assess middleware frameworks described in the previous chapters in order to give a brief overview. The following tables include an unique metric identifier and a short description:

- The *Range* of a metric corresponds to a subset that can be used as assessment criterion.
- The *Target aspect* refers to targeted assessment metric aspect displayed in Figure 1.
- The *Importance* level gives a measure of the general importance of the corresponding metric with respect to overall assessment. It must be noted, however, that these are based on a generic design flow. Specific design decisions might affect the level, for instance, in the case of certain licenses to be omitted or certain programming languages to be used.

3.1 Assessment Metrics

Field Name	**Value**
Metric ID	WP31.1.1
Title	Node communication mechanism
Description	The node communication mechanisms can differ in range of simple message passing between nodes to more advanced mechanisms such as ports, topics, events, services and properties. The communication mechanisms will influence the coupling of nodes, performance and ease of use of the middleware framework. The node communication mechanisms include Simple messages, Topics, Ports, Services, Events and Properties.
Range	Simple messages, Topics, Ports, Services, Events, Data/Message Centric
Importance	High
Assigned Tasks/WP	Task 31.1
Target aspect	Modularity
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.2
Title	Message Transport Protocol
Description	The message transport protocol is a crucial characteristic of middleware frameworks that is also related to the communication layer. The supported transport protocols like TCP, UDP, EtherCat, CAN-open, HTTP, SSL, CORBA will significantly change the performance of the middleware framework.
Range	TCP, UDP, EtherCat, CAN-open, HTTP, SSL, CORBA
Importance	High
Assigned Tasks/WP	Task 31.1
Target aspect	Composability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.3
Title	Programming Languages
Description	A middleware framework can support one or multiple programming languages. On one hand the used programing language can directly influence the performance of the robotic system and should be selected to match the operation platform performance. On the other hand it directly influences the time required to create new software components.
Range	C++, C, Python, Java, Lisp, Octave, PHP, Simulink
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Composability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.4
Title	Services
Description	Services are used to coordinate the communication between nodes. Naming Services, Lookup Services and Discovery Services provide features to set up the data transfer. Middleware Services can be divided into Naming Service, Lookup Service and Discovery Service.
Range	Naming Service, Lookup Service, Discovery Service
Importance	High
Assigned Tasks/WP	Task 31.1
Target aspect	Composability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.5
Title	Simulation Capabilities
Description	These permit modeling, prototyping, and simulation of the final system to be generated, thereby saving both time and expenses. They also serve as an early test of viability of the solutions, which may prevent situations of malfunction, conflict, etc. Some free tools which deserve a mention include: OpenRave, Stage, UsarSIM, Gazebo, and Breve.
Range	Yes/No
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Composability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.6
Title	Message format
Description	The message format of a middleware framework determines the interoperability between different platforms and languages. New formats should be easily definable, to support new components with their specific message content without too much programming efforts.
Range	XML, RDF, Java serialization, custom format
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Configurability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.7
Title	Real-time capabilities
Description	The majority of robotic systems have some type of real-time constraints. These restrictions are problematic in distributed software architecture. Efficiency is also a common requirement, especially when a robotic system has limited communication and computation capacities. Hence the design of the architecture must consider the use of software, hardware, communication mechanisms and protocols that guarantee compliance with these restrictions.
Range	Yes/No
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Real-time capability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.8
Title	Host OS
Description	The operating systems offer a hardware abstraction layer (HAL) that simplifies the development of applications enormously and encourages the reuse of hardware and software. Classically, software architectures in robotic systems have undergone ad hoc development and the use of an operating system has not always been necessary. General-purpose operating systems which are the most highly developed in these aspects are found in the world of PCs: Linux, Windows and OSX are some examples. As a disadvantage, these systems can be considered as heavy-weight since they require greater hardware resources. Nevertheless, in many cases, this does not prevent them from being able to support tasks with real-time restrictions (RTLinux, RTAI).
Range	Windows, Linux, OS X, RTAI, Xenomai,RTEMS, μC/OSII
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Stability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.9
Title	Algorithm library
Description	Robotic algorithms are often the objective of an middleware framework to provide generic and reusable algorithms and functionalities in the field of robotics. Those algorithms are designed for at different levels of abstraction: from a low level, such as those related to kinematics, control, robot perception, Bayesian estimation up to others of a high level such as planning, human interaction, robot learning, navigation algorithms, motion planning, Bayesian Filtering, and SLAM.
Range	Small/Medium/Large
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Reusability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.10
Title	License
Description	The license is a crucial factor for the success of development frameworks. There have been multitudes of proprietors of unsuccessful frameworks in the sphere of robotics. The license is also significant in the scope of the assessment on one hand since Open-Source projects allow an understanding of how the framework functions, and facilitate the creation of a more powerful community, which together form a source of new ideas. On the other hand possible license fees have an impact on the financial calculation in the exploitation phase.
Range	LGPL, GPL2, BSD, proprietary
Importance	Low
Assigned Tasks/WP	Task 31.1
Target aspect	Reusability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

Field Name	Value
Metric ID	WP31.1.11
Title	Distribution
Description	The modular nature of middleware architectures enable scalability features in terms of distribution over multiple machines in the middleware network. One of the major scale factor of middleware frameworks is the number of processing modules or nodes.
Range	Yes, No, max. number of nodes
Importance	Medium
Assigned Tasks/WP	Task 31.1
Target aspect	Scalability
Responsibility and Reference Person	Viatcheslav Tretyakov (SYN), Sönke Michalik, Jan Wagner (TUBS)

References

[1] Simone Ceriani and Martino Migliavacca. Middleware in robotics. Advanced Methods of Information Technology for Autonomous Robotics, Internal Report on. http://home.deib.polimi.it/gini/AdvancedRobotics/docs/CerianiMigliavacca.pdf.

[2] Pablo Iñigo Blasco, Fernando Diaz-del Rio, Ma Carmen Romero-Ternero, Daniel Cagigas-Muñiz, and Saturnino Vicente-Diaz. Robotics Software Frameworks for Multi-agent Robotic Systems Development. Robot. Auton. Syst., 60(6):803–821, June 2012.