Robotics software frameworks for multi-agent robotic systems development

Pablo Iñigo-Blasco, Fernando Diaz-del-Rio, Mª Carmen Romero-Ternero, Daniel Cagigas-Muñiz, Saturnino Vicente-Diaz

Escuela Técnica Superior de Ingeniería Informática, University of Seville, Avda. Reina Mercedes s/n, 41012, SEVILLE, Spain

A B S T R A C T

Robotics is an area of research in which the paradigm of Multi-Agent Systems (MAS) can prove to be highly useful. Multi-Agent Systems come in the form of cooperative robots in a team, sensor networks based on mobile robots, and robots in Intelligent Environments, to name but a few. However, the development of Multi-Agent Robotic Systems (MARS) still presents major challenges. Over the past decade, a high number of Robotics Software Frameworks (RSFs) have appeared which propose some solutions to the most recurrent problems in robotics. Some of these frameworks, such as ROS, YARP, OROCOS, ORCA, Open-RTM, and Open-RDK, possess certain characteristics and provide the basic infrastructure necessary for the development of MARS. The contribution of this work is the identification of such characteristics as well as the analysis of these frameworks in comparison with the general-purpose Multi-Agent System Frameworks (MASFs), such as JADE and Mobile-C.

Keywords:
Robotics
MAS
Agents
Software frameworks
Middleware
Architecture

1. Introduction

At the end of the last century, industrial robotic systems were oriented toward the mass production of products in factories, where robots of high precision were destined to carry out repetitive individual work in controlled environments. However, the current tendency is toward robotic systems that must be capable of solving problems in environments that are more complex and less controlled. To this end, robotic systems are needed that are more autonomous and intelligent.

Some of these systems are mobile teams of autonomous robots where a set of robots works as a group to attain a common objective [1,2]. These systems require cooperative, social robots, which can move in dynamic, complex uncontrolled environments, where the capacity to understand and interpret the surrounding world is their prime challenge.[1] As a consequence, the complexity of the software architecture increases and the computing needs soar. In these software architectures, the scalability, reusability, efficiency and fault tolerance play a fundamental role. The system software architecture must be designed in a distributed and modular way. These systems must nevertheless continue to take classic problems of industrial robotics into account, which are closer to the sensors and actuators, such as the real-time requirements in their lower levels. The principal characteristics, which the software architecture (and its components) of a complex robotic system must cater for, are then enumerated [3–9]:

- *Concurrent and distributed architecture*. It is necessary to be able to take advantage of all the processing units available in a concurrent way (processors, multi-processors and microcontrollers) in order to cover all the computational needs that a complex robotic system presents. Due to the consequent system complexity, a powerful remote inspection (also known as introspection) mechanism is needed.
- *Modularity*. The software architecture is formed of several components of high cohesion and low coupling. The components interact with each other; however, the dependences must be kept at a minimum in order to obtain a maintainable, scalable, and reusable architecture, which is adaptable to changes and improvements. Although these are desired features in all software architectures, they are especially important in Robotics, where the lack of standards and the closeness with the hardware have made robotics software prone to be non-reusable and non-scalable for succeeding decades. The need for a well-designed common robot hardware interface is another related feature.
- *Robustness and fault tolerance*. The malfunction of a component must not completely block the whole system. On the other hand, the rest of the system must be capable of continuing to work as best it can with the resources available on the condition

Corresponding author. Tel.: +34 954 55 61 44; fax: +34 954 55 28 99.
E-mail addresses: pabloinigo@us.es (P. Iñigo-Blasco), fdiaz@us.es, fdiaz@atc.us.es (F. Diaz-del-Rio), mcromerot@us.es (M.C. Romero-Ternero), dcagigas@us.es (D. Cagigas-Muñiz), satur@us.es (S. Vicente-Diaz).

that it is working toward achieving the objectives. To this end, the rest of the components (still in working order) must be capable of acting on their own initiative and autonomously make decisions to overcome these situations. These decisions must be taken based on cooperation with other agents of the system or on the specific information they possess.
- *Real time* and Efficiency. 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.

Hence, the main reasons why MAS are a good choice for robotics software architecture are that, when using this approach, the resultant software is much more reusable, scalable and flexible whilst the parallelism, robustness and modularity requirements are maintained. This approach is demonstrated through its successful application in multiple areas in Robotics (which will be studied in the Section 2). Indeed, Multi-Agent Robotic Systems (MARS) have been widely studied over recent years and related events exist in the form of competitions such as RoboCup [10], and of workshops such as ICINCO-MARS [11].

Technologies are available that enable the development of general-purpose MAS which will be referred to as MASFs (Multi-Agent System Frameworks) in this work. These technologies include: JADE [12], Grasshopper [13], JACK [14], Cougaar [15], ADE [16], and Mobile-C [17]. JADE is the *de facto* solution for various reasons, most importantly that it was the first to implement the MAS specification defined by FIPA[2] and was an Open-Source solution both accepted and supported by the MAS developers' community [12]. These technologies have been used successfully during the past few years. However, whilst they present a valid approach for robots, their use has not been focused on the development of robotic systems but rather on other areas such as Web services, e-commerce, domotics, sensor networks, social simulation, finances and e-learning.

On the other hand, "Robotics Software Frameworks" (RSFs) [4,18–20] attempt to provide an integral solution through a set of generic tools and off-the-shelf libraries with algorithms and controllers useful to create a general-purpose robotic systems, thereby avoiding to continually reinvent the wheel. On occasions, these frameworks are known as "Robotics Middleware" or "Robotics Software System". In the present study, we refer to these as "Robotics Software Frameworks" (RSFs).

Currently, MASFs and RSFs offer tools and solutions that are similar in many aspects, especially in those focused on distributed communications architecture. This overlapping between RSFs and MASFs is represented in light gray in Fig. 1. The software infrastructure essential for MAS development is already implemented by some RSFs: support for the development of P2P (Peer to Peer) distributed architectures, inter-agent message-passing methods, or Yellow Pages service. For that reason these RSFs already present suitable infrastructure for the development of MARS.

In this study, most widespread technologies that enable the development of software architectures of MARS will be enumerated and analyzed. The most significant common characteristics are explained, with particular attention paid to those characteristics that a robotics framework must include so that it is adaptable to the MAS paradigm. Furthermore, a comparative study will be presented that can serve as a reference for the choice of one of these technologies in a real project. Finally, some interpretations of the most significant characteristics and the shortcomings of these technologies are expounded.

[2] FIPA (Foundation for Intelligent Physical Agents) is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

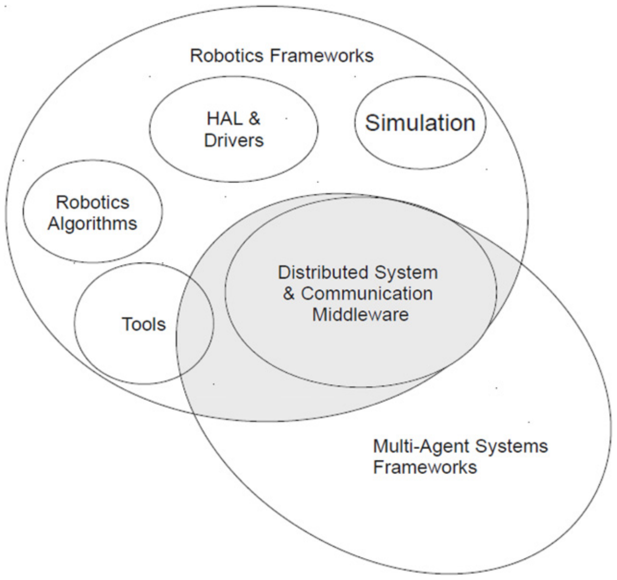

Robotics Frameworks
HAL & Drivers
Simulation
Robotics Algorithms
Distributed System & Communication Middleware
Tools
Multi-Agent Systems Frameworks

Fig. 1. Robotics software frameworks and multi-agent system frameworks.

The structure of the document is as follows: In Section 2, the fields of application for the Multi-Agent Robotic Systems (MARS) are explained. In the Section 3 the definition of a Robotics Software Framework (RSF) is given together with its characteristics, and the MASFs are presented in the context of robotics. Under the heading "Main Aspects of Frameworks", the criteria for comparison between the various RSFs are specified. In the following section, each framework is described and a series of tables that summarizes the comparative results is presented. In the final section, an interpretation of the comparison is carried out with emphasis on the points in common and the deficiencies in the frameworks presented.

2. On the relationship between multi-agent systems and robotic systems

2.1. Multi-agent system definition

In the context of MAS, a general definition of agent could be "an autonomous proactive and social software component" [21]. The agents are autonomous, possess their own thread of control and are independent from other processes. The agents can be reactive but can also be proactive. In addition to responding to other messages or external events, the agents may sometimes take the initiative and change their behavior in order to attain their objectives [21]. Proactivity needs agents of a more intelligent nature. This intelligence may or may not be attained through learning. Less ability to learn usually implies more reactive agents whereas more ability to learn usually implies agents of a more proactive nature. The more uncertainty or complexity within the problem to be resolved, the more intelligence the agents should have, and also the greater the capacity to learn is required. Many authors have defined cognitive models [22–29], or cognitive architectures [30–40,22], that allow automatic learning. It is also important to take into account how the interaction between agents can help toward the learning process and toward the improvement of the problem solution. To this end, social aspects are considered by many authors in the design of the multi-agent system [29,41–46].

Hence, given the aforementioned characteristics and the robotics software architecture constraints, a solution based on the multi-agent paradigm is highly suitable for the design of a complex robotic system. The software architecture of the robotic system would therefore be formed of distributed nodes (Agents), which communicate and cooperate in order to attain the general objectives. This approach makes this architecture much more reusable, scalable and flexible whilst the parallelism, robustness and modularity requirements are maintained.

Multi-agent systems are composed of multiple agents distributed over several agencies (or hosts) on a network. They are social, that is to say, they need to interact with each other to reach an overall objective of the system. These systems form peer-to-peer architectures and communicate by means of messages. Suitable for the development of complex distributed systems, they are heterogeneous where the agents require a low coupling level, and useful in contexts where designing a solution of the problem by a single agent is complex. MAS present a "divide and conquer" approach where the solution of a complex problem can be found through the solving of simpler problems.

2.2. Applications for multi-agent robotic systems

Some areas and examples in robotics where MARS are currently applied and that would benefit from these multi-agent features are described below:

Heterogeneous mobile robot teams.

A team of mobile robots with various capacities cooperate together to solve a problem. The members of the team exchange information and evaluate the decisions to be made. Various pieces of research have been carried out in this respect, for example on a team of aerial anti-incendiary robots where each robot possesses distinct complementary sensory and locomotory capacities. The robots cooperate on order to detect forest fires. When one robot localizes a fire, it warns the rest of the robots and decides through cooperative perception whether it really is a fire or just a false alarm [1,2].

Robots working in ambient intelligence environments.

Mobile robots communicate with each other using elements of environmental intelligence in order to obtain information of a more detailed nature about their surroundings and to be able to make decisions based on compared information. For example, projects exist where robots cooperate with the environment in order to solve everyday housework problems. Furthermore, in the assistance for the disabled, assistive or shared control wheelchairs can cooperate with an intelligent building and decide the optimum route to arrive at some place of interest [47–49,10,50].

Modular robots.

Completely independent robots can be physically united to create a new, larger robot. The new combined robot possesses other kinematic characteristics, a greater number of DOFs (Degree Of Freedom) or special qualities. The new structure enables it to confront problems of a more complex nature such as the overcoming of obstacles, swimming, rolling, crawling, or even the manipulation of objects [51,52].

Collective robot swarms.

A great number of homogeneous robots cooperate so that they can consequently achieve common objectives of a more complex nature. No identification of the robots is necessary since they possess no special characteristics. The robots cooperate with other robots close by and their acts are primarily based on the local information available and on that of the individuals surrounding them. They are scalable and redundant systems; one single robot is dispensable and completely interchangeable. These systems can be capable of solving problems related with exploration, vigilance, path planning, creation of *ad hoc* wide-coverage wireless networks, etc. [53,54,2,55].

Mobile sensor robotics networks

A set of mobile robots with sensors permits the creation of a network of sensors of a flexible structure. Hence they enable the location of the sensors to augment the coverage of the sensors or of another characteristic in order to focus attention on an area, or to support another individual which presents defective behavior [56–58].

Multi-agent control systems

Multiple agents on a network collaborate to control a robot [59]. Interesting example are Articulated and Humanoid Robots. These are composed of a high number of hardware and software components and require very modular architecture. They possess multiple processing and the software can be organized through agents that control distinct parts of the system and cooperate in order to achieve the overall objectives, but these agents conserve their autonomy, proactivity and sociability. It is crucial that the failure or substitution of one of these components causes no complete system failure. For example, a component of a humanoid robot, where an agent controls an arm, communicates by means of messages with a central brain which makes strategic decisions, however this component can react autonomously in certain situations in order to prevent damage as if it were a reflexive reaction. Another example is an agent that controls an arm and another that controls a leg cooperate to maintain complete balance of the robot by using external information in their control loop when it is available [4,60–62].

2.3. Robotics software architecture for MARS

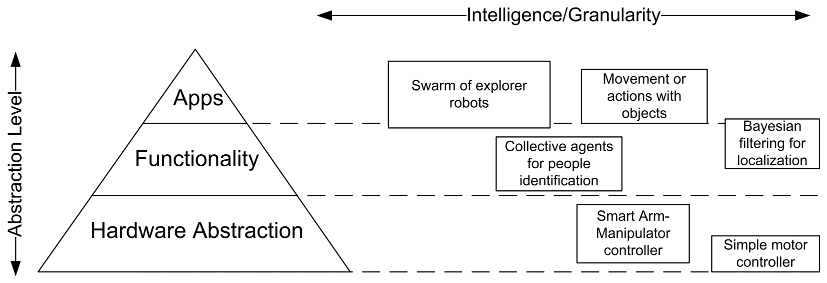
From the point of view of classic layered robotics software architectures (see Fig. 2), MARS are typically located on the top application layer. In layered software architectures each tier exposes an API to the above layer. Components located within the same layer interact with each other horizontally (they usually share the same process).

In terms of MARS, the intelligence, proactivity and social aspects would be located within the top application layer. This means that agents are processes that use the services of lower layers (which handle all the hardware robotic devices and provide functionality for robotics algorithms). Many existing robotic technologies promote this layered architecture for the functional and hardware abstraction layers, thereby providing a set of components (also known as Component-Based Robotic Engineering [7–9]) for each layer. A representative set of technologies is specified in Sections 3 and 4. Moreover, some of these technologies provide a distributed mechanism to further decouple the hardware abstraction layer from the functional layer. Layers communicate to each other through RPC Mechanisms in a Client/Server Architecture, thereby making the system more flexible and modular.

For a proper MARS development, the application layer must have a powerful communication infrastructure that enables agent capacities to be developed. Three alternatives are possible for this application layer:

- Custom communication software built on top of platform communication mechanisms, such as sockets, libraries, field buses, and drivers.
- The use of a general-purpose communication middleware, such as CORBA, and Web services.
- The use of a multi-agent system framework (MASF), such as JADE, Mobile-C, COUGAR.

Most existing cases of MARS applications use one of the first two approaches, both of which end in a continuous "reinvention of the wheel" of the communication software for each project. Moreover, a custom implementation of a P2P architecture could be arduous and usually ends up as a centralized Client/Server Architecture model. The third option is shown to be the most adequate for the



Intelligence/Granularity
Abstraction Level
Apps
Functionality
Hardware Abstraction
Swarm of explorer robots
Movement or actions with objects
Bayesian filtering for localization
Collective agents for people identification
Smart Arm-Manipulator controller
Simple motor controller

Fig. 2. Examples of robotics software organization according to their task abstraction level and their degree of intelligence.

development of MARS applications in a layered architecture, as some experiments have revealed [63].

Nevertheless, although applications are the most evident context for MARS, tendencies show that even the Functionality Layer and the Hardware Abstraction Layer can benefit from a MAS approach. One example is that of the aforementioned multi-agent control systems; further examples are mentioned in Section 4.2. These examples show that even functionality and hardware abstraction components can be considered as agent nodes. Further discussions on the differences between these two architectures are tackled under the heading "Distributed Architecture" of Section 4.2, and under "Robot Hardware Interfaces" of Section 4.3.

Therefore, a robotics software component can be considered as an agent according to the degree of intelligence or complexity, but not according to the abstraction level of the performed task. Hence components of the Functional and the Robot Hardware Abstraction layer may be instantiated as agents. These agents also have the capacities of autonomy, sociality, proactivity and intelligence. Furthermore, they need a powerful software infrastructure that promotes the aspects of distributed communication and remote inspection.

Layered software architectures are consequently considered as being too inflexible to develop MARS properly, since a layered architecture would need a replication of the distributed communication mechanism for each layer and would limit the interaction between layers. Conversely in MARS architectures, each node agent is itself an application, and is independent of the abstraction level of the task. A new Distributed Architecture is therefore needed, which reflects the Service Oriented Architecture (SOA) paradigm in the software engineering context. Certain existing technologies promote this paradigm while others promote a layered architecture. The technologies feasible for MARS development are analyzed in this work (see Sections 4 and 5).

Finally, from the MAS point of view, a proper classification of Software components should be based on the degree of intelligence, or complexity granularity. A robotics software architecture is composed of components of varying granularity; these components can be considered as agents under certain circumstances:

The fine-grained components tend to have a well-defined task and behavior and hence are very cohesive. They are usually modeled by means of mathematic tools such as: functions, rules, and truth tables. These components seldom fall within the definition of an agent. An example could be an algorithm located in a microcontroller that controls the servomotor of a robotic arm, or a component that carries out a specific process on an image from a camera.

Medium-grained components are composed of various subcomponents and can present a more complex behavior, which, on occasions, cannot be modeled. This component of medium granularity can fit the definition of an agent when it implements some degree of autonomy, sociability and intelligence. For instance, a smart agent node that acts as an arm-manipulator controller may interact and communicate with other nodes in an intelligent way. To ensure system robustness, the node asks for external information in order to make a decision, and notifies the consequences to other agent nodes in the case of failure, and sends reports, and so forth. Another example is a humanoid equipped with a set of sensors in the form of cameras and haptic sensors, and a manipulator to grab objects. The sensors could collaborate in the identification of a material, and, at the same time, interact with a hand actuator to quickly drop the object as a reflex movement if damage is being produced (due to temperature, chemical erosion, and so on). In this case, this robot is formed of multiple agents of medium granularity that cooperate, thereby creating a small MARS and defining together the behavior of the main agent which represents the whole humanoid robot.

Coarse-grained components in a complex robotic system tend to fit the definition of an agent. A clear example is an autonomous mobile robot which collaborates with other robots in a team to explore a building. Another example is a set of distributed cameras on robots or in the environment that collaborate to identify and locate people on a map.

3. Robotics software frameworks

3.1. State of technology

In the year 2009, and only in SourceForge, there existed more than 500 free software projects related with robotics [64]. Examples of these projects implement: drivers for robotics devices and sensors; communication middleware; simulators and modeling tools of dynamic systems. A few of these projects attempt to provide an integral solution through a set of generic tools and off-the-shelf libraries with algorithms and controllers useful for the creation of general-purpose robotic systems.

There are RSFs that are sufficiently suitable for the implementation of MARS. This is because, in spite of some deficiencies with respect to general-purpose MASFs, they offer the infrastructure and tools necessary for the creation and deployment of P2P distributed architectures, where they execute components that fulfill the agent definition (autonomy, sociability, and proactivity).

The RSF develop aspects such as scalability, reusability, deployment, and debugging simplicity of hardware and software components. They also provide modeling and simulation tools to facilitate the tasks of design, verification, and testing. In this way, a suitable environment is achieved for the creation of larger, more complex, and more integrated robotics architecture. Due to the great diversity in the characteristics that the current RSFs offer, their comparison is not always straightforward. No outstanding solution for all the aspects exists, nor is there a consensus on the organization and structure of this type of technology, nor on what aspects must be covered. Some of the best-known OpenSource RSFs include: Player [65], OROCOS [66], ROS [67], YARP [68], OpenRave [69], OpenRTM [70], OpenRDK [71], MOOS [72], MIRO [73], JDE+ [74], ORCA [75], MARIE [76], and

CARMEN [77], OPRoS [78], CLARATY [79], MARIE [80], Mobile Robots [81], MRPT [82], MSRS [83], Peis-Ecology [84], Pyro [85], Webots [86]. Moreover, while some non-open-source robot software platforms have appeared, such as Microsoft Robotics Studio, the focus of this study is to be on open source systems. This is because two reasons: first, the diversity of non-commercial platforms is very wide, which permits to compare and discuss lots of implementations, analyzing their drawbacks and advantages. Secondly, internal architecture of non-open-source platforms is (due to commercial reasons) less documented.

3.2. *Aspects focused on by existing RSFs*

Despite the varied foci, many common points are shared; in general they are all geared toward contributing solutions for typical problems in robotics. These RSFs are normally centered on one or several of the following areas:

- *Middleware for distributed robotics*. Some RFSs provide a set of tools that enable the architecture of the robotic system to be organized by taking advantage of characteristics that are intrinsically parallel to the robotic system. At runtime, the architecture is organized into nodes which communicate with each other by means of passing messages. This design of robotic systems may be more complicated than monolithic approximations, but the architecture yielded is a lot more flexible thanks to the high degree of uncoupling of the nodes. All the presented applications in Section 2.2 are highly sensitive to the use of properly distributed middleware during the software design and development stages. The use of this kind of middleware is extremely recommended, almost mandatory (as mentioned in Section 2.3). The middleware for distributed robotics tools promotes scalability, reusability, and the high tolerance of errors, and also renders parallel development and integration tasks easier. Examples include: ROS, YARP, OpenRDK, OpenRTM.
- *Introspection and management tools*. The introspection and management tools permit the monitoring, visualization and analysis of the state of the robotic system to be carried out during execution which enables the tasks of tracking, and of error detection and correction. Tools worth mentioning here include those of logging, management, and system-state monitoring from the graphic interface, the console, or the web. They are especially important in distributed robotics architectures, given the difficulty of debugging and testing of such architectures.
- *Advanced development tools*. These enable time and expense to be saved in the implementation and testing stages; an especially important aspect in a complex robotic system. Development tools include compilers, dependency managers with other modules or system libraries, and Integrated Development Environments (IDEs). Deployment tools permit scripts and configuration files to be designed which define the start-up of the robotic system. The initial values of the parameters, the nodes used, its localization, name, and connections, among other details, are specified in these files.
- *Robot hardware interfaces and drivers*. These encapsulate the code of the controller of a specific device (sensor or actuator) behind a well-known and stable programming interface. This interface has to be defined between robot devices and user modules. The objective is to attain reusability of the devices and of the high-level algorithms that they use. It is common for a specific device to offer various functions and particular configuration parameters. When this occurs, the device controller can implement various interfaces according to its characteristics. For example, if a commercial robotic arm offers a system of integrated vision in the end-effector, then the controller can implement the generic interface of the robotic arm and of the vision. The specific parameters of the device are stated in a configuration file, and thereby the algorithms remain free from coupling with specific devices, and reusability is maintained. In this way, the Open-Close principle of Software Engineering is observed: code is open to extension and closed to modification. The frameworks specialized in this area offer programmers a simple development mechanism for their own drivers. Furthermore, they commonly provide a set of built-in controllers for diverse commercial devices. Player, MOOS, ORCA, and CARMEN are examples of Frameworks that are centered in this area. This aspect will be taken into account in the RSFs surveyed in Sections 4 and 5.
- *Robotics algorithms*. These are often the objective of an RSF: to provide generic and reusable algorithms and functionalities in the field of robotics. They correspond to the functionality and application levels shown in Fig. 2. Those algorithms are encountered 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. 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 RSFs, 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.
- *Simulation and modeling*. 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.

3.3. *Existing RSFs overview*

The earlier technologies tackled a design of a Robotics Framework following an abstraction-level-based architecture (see Fig. 2). However a proper MARS software architecture should have a strong infrastructure focused on distributed systems where nodes communicate with each other independently of the abstraction level of the task that they perform. To this end, in this survey only those frameworks that fulfill the conditions stated under the heading "Middleware for distributed robotics" of Section 3.2 are to be analyzed.

Table 1 shows a set of existing RSFs and their strong characteristics. Those RSFs selected for a more in-depth analysis are shown in bold font. Some of the discarded frameworks may be excellent, popular and active RSFs. However, they fail to represent the best choice for a complex MARS since they provide an insufficiently powerful distributed middleware mechanism. Other aspects significant in the criteria for MARS, such as introspection tools and development tools, are also taken into consideration. In addition, other selection criteria considered in this study include: characterization with an Open-Source and free-commercial license, proper documentation for evaluation, a

Table 1
Characteristics of existing RSFs.

RSF name	Open source and commercial Free software	Advanced distributed Architecture	Introspection and management tools	Hardware interfaces and drivers	Robotics algorithms	Simulation	Advanced development tools	Control and real-time oriented
CARMEN	Yes	Yes	No	Yes	Yes	Yes	No	No
CLARATY	No	Yes	Yes	Yes	Yes	No	Yes	No
JDE+	Yes	No	No	Yes	Yes	Yes	No	No
MARIE	Yes	Yes	No	Yes	Yes	Yes	No	No
MIRO	Yes	Yes	No	Yes	No	No	No	No
Mobile robots	No	No	No	Yes	Yes	Yes	No	No
MOOS	Yes	No	No	Yes	No	No	No	No
MRPT	Yes	No	Yes	Yes	Yes	No	No	No
MSRS	No	Yes	No	Yes	No	Yes	Yes	No
OpenRave	Yes	No	No	No	Yes	Yes	No	No
OpenRDK	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**No**	**No**	**No**
OpenRTM	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**No**
OpROS	No	Yes	Yes	Yes	Yes	Yes	Yes	No
ORCA	Yes	Yes	No	Yes	No	No	No	No
OROCOS	***Yes***	***Yes***	*bf No*	***Yes***	***Yes***	***No***	***No***	***Yes***
PEIS-ecology	Yes	Yes	Yes	Yes	No	Yes	No	No
Player/stage	Yes	No	No	Yes	Yes	Yes	No	No
Pyro	No	No	No	Yes	Yes	Yes	No	No
ROS	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***No***
Webots	No	No	No	Yes	No	Yes	Yes	No
YARP	***Yes***	***Yes***	***Yes***	***Yes***	***No***	***Yes***	***No***	***No***

minimal level of maturity, and recent project updating (within the last 12 months, that is, since January 2011). Hence, frameworks, such as OpROS, CLARATY, Peis-Ecology, and Microsoft Robotics Studio, have been discarded here (although they may be adequate for the development of MARS).

4. Main aspects for multi-agent robotic systems software development

In this section, a general analysis will be made of the different frameworks (RSFs and MASFs) proposed for the MARS development. Given that the present study is developed in the MARS sphere, special emphasis will be given to the characteristics related to the P2P distributed architecture of the robotic system (since this is the basic infrastructure for MARS development). These architectures are organized by means of nodes (modules), where a group of the nodes of the system fulfill the agent definition. Certain other important general aspects for the development of robotic systems are also briefly analyzed.

There are 23 characteristics compared in the three sections: “General Aspects”, “Aspects of Distributed Systems”, and “Other Aspects of Robotics Frameworks”.

4.1. General development aspects

In the first section of characteristics, general information of the software is shown on the state of the projects of the Frameworks.

Founder organization

All the frameworks studied arose in organizations; typically companies or universities. Although this work only contemplates Open-Source projects and the community that contributes to their development, it is usually the promoting organization that makes the main contributions and indicates the guidelines for the development of the project. This allows the project to evolve in a coherent manner. The implemented techniques and characteristics are usually supported by detailed and formal scientific publications.

Year

The year of creation is a measure of the maturity of the project. Many of the Open-Source RSFs that can be found are left inactive after a couple of years of development. This fact can be justifiable considering the rapidity of the advances in this area. New technologies include new characteristics, leaving the old RSFs obsolete. Therefore, it is another important factor to consider as it represents maturity, good design and capacity to adapt over time.

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 and MAS. The license is also significant in the scope of the investigation 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.

Latest release

This is an indicator of project activity and maturity as well as of current activity and trends. In this comparative study, only those active frameworks that have been updated in the last twelve months have been considered (this study has been made in January 2012).

Supported operating systems

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 multi-tasking 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 RSFs and MASFs 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.

Programming language

The programming language used for development is also an important aspect. 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 (see Section 4.3), the transport mechanism, and some agent capacities, such as mobility (see Section 4.2).

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 MAS architectures. This means that it presents an adequate language for both low-level control tasks and high-level programming. 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 [87], to prevent programming language restrictions to be an application barrier when deploying MARS.

4.2. Aspects of distributed systems

Those aspects related to the communications used in RSFs are analyzed in this section of characteristics. This is the main section where the differences and similarities between the proposed RSF and the MASFs are shown.

Distributed architecture

The distributed architecture (also known as Distributed Topology) for a Multi-Agent System should be Peer to Peer (P2P) from a strictly theoretical point of view (see Section 2.1). In a purely P2P system, the nodes interact among themselves without the need for a coordinating fixed central node or authority. This model displays a high tolerance to failures since no special node is indispensable. With this approach, the communication performance can be also improved since bottlenecks are avoided.

In terms of MARS applications, it is clear that the vast majority can benefit from a pure P2P Architecture. Let us consider, for instance, a swarm of robots that explore huge extensions of terrain, and which are constrained in communications and energy autonomy. The robots communicate to each other wirelessly and form a mesh network logic topology.

In this application, a centralized architecture could be catastrophic for several reasons: If the central node becomes isolated, all the robotics architecture crashes. Moreover, if each message has to pass through the central node, then the latency of communications increases and the central node may become saturated and consequently crash. Both cases show the poor robustness of this architecture.

Nevertheless, a pure P2P architecture has some drawbacks, such as the increase of internal complexity of the RSF. P2P also presents certain difficulties when different nodes are put in contact, and when coordinating these nodes; hence some extra services are needed, such as a distributed naming service, and a discovery service.

Finally, hybrid models also exist that attempt to combine the advantages of these alternatives. A central node server exists (sometimes called the Object Broker) which undertakes special tasks, such as putting different nodes in contact: naming services and lookup services are examples of hybrid models (see below). Nevertheless, once put in contact, the nodes interact among themselves independently by means of point-to-point communication [12].

It should be borne in mind that the many RSFs which use a pure Client/Server model in a distributed layered architecture (see Section 2.3), such as PLAYER, MIRO and MOOS, are not sufficiently flexible, scalable and robust for the development of a MARS. The frameworks presented and analyzed in this survey are suitable for the development of MARS since they provide the necessary infrastructure for the interaction of nodes in a P2P or Hybrid-P2P architecture.

Node communication mechanisms

The nodes interact among themselves by means of the passing of messages. Simple message passing is the basis that allows other types of more advanced mechanisms to be constructed, such as ports, topics, events, services, and properties. These mechanisms allow the attempted communication to stand out and they offer some advantage in the design or implementation: low coupling, performance, ease of use, etc. A common characteristic of these mechanisms with respect to simple message passing is that they are *addressable communication channels*, for example a service or port of a node needs a name in order for it to be located and referenced. In the case of Hybrid P2P systems, all these communication channels must be registered in the main node that hosts the naming service and lookup service.

- *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. It is also the most commonly used in the MASFs. 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). Fig. 3 shows several examples of interconnections between nodes with ports. They can be created and destroyed dynamically, which contributes great flexibility. Each node is designed to read and to write in its ports independently 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 is to say, it 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. Figs. 4 and 5 represent an example (usual on the RSFs) where a topic has two publishers and two subscribers. In many aspects topics are similar to ports, nevertheless one of the main differences is that a topic does not pertain to any node, it is a global system resource, whereas a port pertains to a node.

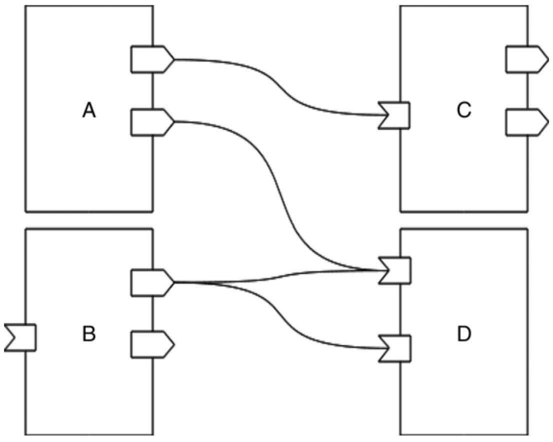


Fig. 3. Port mechanism.

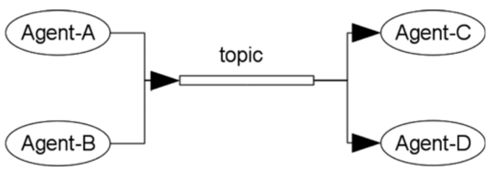


Fig. 4. Node interaction through the topic mechanism (logic view).

- *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. This mechanism is typically used for Robot Hardware Interfaces (see "Robot Hardware Interfaces" in Section 4.3).
- *Properties*. The nodes can show a set of properties that represent part of their status to the rest of the nodes. Each property is usually managed through a pair of services get/set (get property for the listener, and set property for the node that is going to show a property). However, several RSFs treat these two services independently. In Hybrid P2P architectures, the parameters are often stored in the master node. In such a case, it is called the "Parameter Server". It is worth mentioning that the pure P2P approach is more desirable. For instance, ROS uses a parameter server, while OROCOS and YARP have real distributed node properties.

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. This service is present in all of the frameworks analyzed, but most of the discarded frameworks of Section 3 fail to provide it.

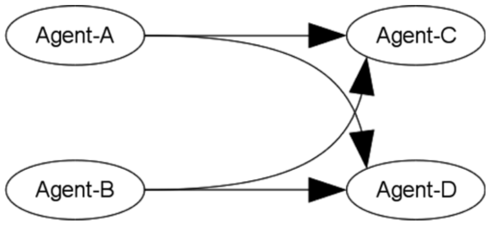


Fig. 5. Topic mechanism implemented over a P2P network protocol (like TCP).

Generally, the RSFs display naming-service mechanisms (also known as White Paper Service). Some RSFs have characteristics of a more advanced nature such as "Name Pushing" and the relative addressing of agents or resources of the MAS. 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. Fig. 6 shows how two groups of resources (Actuator-Agent, Intelligent-Agent and control topic) are instantiated twice and located in two different namespaces. It can also be seen how the absolute references to the monitor topic and Monitor-Agent resources stay in both groups in spite of name pushing.

Lookup service.

This service is peculiar to Hybrid P2P architectures where a central or master node exists which contains 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

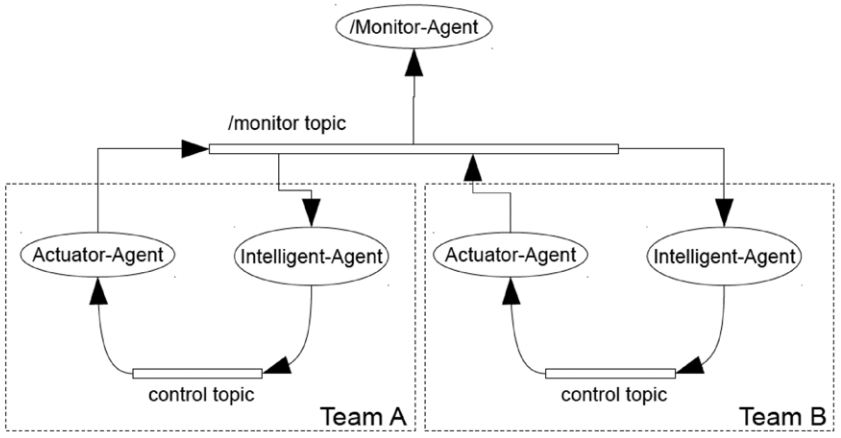


Fig. 6. Absolute and relative names for nodes and topics.

certain properties. It is worth mentioning that the lookup service for multi-agent systems is well defined in the FIPA standard [13].

Not all the RSFs implement the lookup service. This is a fundamental characteristic for the creation of a robust MAS. 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.

There are innumerable situations where this service is essential: heterogeneous robots in teams or swarms can obviously benefit from this service (see the discovery service example below). Another example is a complex autonomous robot controlled by a multi-agent control system in an internal network. Each node works on a specific task: some nodes (agents) in a dedicated CPU work to handle the obstacle avoidance and localization tasks, while other nodes are dedicated to handling sensors and actuators.

The localization and obstacle avoidance agents need a prediction of the future state based on the odometry system. If odometry sensors fail due to the dirtiness of the terrain or to robot damage, then both agents ask the lookup service whether another agent can provide the odometry service. For instance some cameras can start working as a visual odometry system although they have other primary tasks. Although camera agents then become overloaded, the system can continue working until its objectives are achieved or a repair is made.

All these 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. Hence, a discovery service makes the MARS more robust. Let us consider a team of firefighters as a team of heterogeneous robots [1] that is composed of members with different capacities: fire hose, first-aid, tracking, etc. When a fire is detected in a forest, the tracking robots could explore to find people in danger or injured. In this case, one tracking robot should look for the nearest and unoccupied first-aid robot and waterspout robot in order to heal and protect such people. All this negotiation and interaction between robots starts with the use of the discovery service asking for the first-aid and fire hose agent robots.

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.

Agent mobility

This characteristic defines the capacity by which a software agent can move between network nodes, for example, sensors and robots. Some definitions such as the MASIF standard [88] consider an inherent characteristic of the agents, whilst other references [17] distinguish between stationary-agent and mobile-agent. Nevertheless, it is usually implemented in the general-purpose MASFs. On the other hand, this feature has not been implemented by any RSF since robotics architectures are nowadays much more static than other multi-agent systems architectures. Moreover, many of robotics software agents are usually coupled with the hardware and this causes a much higher platform dependency (which is a big problem for an Agent Mobility implementation). In any case, this feature would be desirable in future RSFs especially for higher-level agents of a complex robotic system.

This characteristic could be highly useful in certain Robotic systems, for example, high-level nodes which processes certain algorithms with little dependency on the hardware. In these cases, the agent could travel to idle processing units, whereas the busiest units could concentrate on a specific task of great computational cost. It can also be highlighted that one of the reasons why RSFs do not have mobile agents, is because they are subject to platforms and languages that need to be recompiled when they change platform, such as C++. Nevertheless, there are original alternatives, such as Mobile-C, that use C++ interpretation mechanisms, thereby maintaining access at a low level. Furthermore, to implement this mobility it is possible to take advantage of the high-level interpreted languages, such as Python or Java, which are supported by some RSFs.

An example of application that can benefit from this feature can be found in robots interacting with intelligent environments where high-resolution image-processing tasks are required. High-resolution images bring higher communication latencies between nodes and high computational resources and are therefore tasks of high energy consumption. Let us consider a "navigator agent" originally located in a mobile robot, which requires the recognition of an unfamiliar face. This navigator agent could travel temporarily to the environment computers to perform such a task there, in order to access a more complete face database stored in the building.

Therefore, if an image-processing task is computationally expensive, an agent could use the powerful computational infrastructure of the intelligent building to save both time and energy consumption.

Multi-agent or robotics standard

Standards are crucial for the development of robotics. There are very few standards related with robotics architectures and multi-agent systems, some of which are FIPA [12], OMG-MASIF [88] and OMG-RTC [89]. FIPA and OMG-MASIF are standards that define interaction agent patterns, their communication language, and the necessary infrastructure for the system. The use of standards promotes interoperability between various technologies, which is positive in areas of technology with a great diversity of implementations. Over the last decade, several general-purpose MASFs have implemented these standards, thereby enabling their compatibility to each other. In this way, their interoperability is improved, and therefore their acceptance and diffusion can increase considerably. On the other hand, the RSFs studied here fail to implement inter-agent communication standards, such as FIPA and MASIF. The RSFs are more oriented toward communications middleware. However, P2P-architecture-developed RSFs are becoming more extensive and the lack of standards of interaction between nodes may well be a problem in the near future. Hence, certain guidelines for future RSFs could include the implementation of MAS standard, such as FIPA.

OMG-RTC is a standard for robotics architectures which defines the concept a Robotic Technology Component (RTC) which is a logical representation of a hardware and/or software entity that provides well-known functionality and services. Again this standard potentially promotes the interoperability between various robotics technologies.

Introspection and distributed management tools

Monitoring, introspection, debugging and administration are fundamental problems in a distributed system. Some tools that can make life easier for the development team include:

- Distributed Log System: allows the creation of an event log.
- Monitoring of the agents or node state in execution. These tools can be console or graphic applications. For example, ROS allows the graphical monitoring of some standard messages, which contains robot pose, robot vision, and robot trajectory through the rviz 3D tool.
- Monitoring of the communication channels, or incoming and outgoing messages of an agent, port or topic.
- Administration of the nodes in execution: eliminate, start, clone, migrate (if they are agents), etc.

The studied RSFs and MASFs provide a great number of debugging, monitoring and management tools, and are found in a similar state of maturity.

Transport mechanisms

The node communication mechanisms described above are based on messages. These messages are first serialized, and then sent through a real physical network. These transport mechanisms are also known in certain RSFs (such as YARP and OPRoS) as Carriers or Connectors. The capacity of an RSF to choose between various mechanisms is essential to maintain the scalability, the communication performance, and even the real-time guarantees. This means that MARS applications with heavy communications or real-time constraints must seriously take this aspect into account. This is the case of Multi-Agent control systems and MARS where numerous image and point cloud messages are sent over the network.

Each node communication mechanism may be implemented in a different way. For instance the topic communication mechanism may be more properly implemented with a physical bus network and using a multicast protocol. However there are many situations in which this straightforward solution is not possible. Hence, the more transport mechanisms a RSF supports, the better the service is. For instance, Fig. 5 shows a typical situation where the topic communication mechanism is implemented over a point to point protocol like TCP.

It is worth mentioning that robotic systems are often designed over an Ethernet. Field Buses, such as CANBus, I2C, EtherCAT, Serial lines, FireWire, PROFIBUS, and even PCI are often used. Unfortunately, most RSFs and MASFs use only the IP protocol, which means that certain capabilities, such as real-time are missed. Therefore, most of these mechanisms are based on TCP/IP or UDP/IP. However, other alternatives exist. Each method of transport presents its own advantages and disadvantages. The most notable include: TCP protocol displays a trustworthy connection between two nodes; and UDP is useful when faster connections are required where the loss of packets is not critical.

Some frameworks also make use of network-multicast or shared memory mechanisms for processes and threads in the same machine. These transport mechanisms can greatly boost the communications performance. Moreover, many RSFs use transport mechanisms built on top of other general purpose communication middleware or libraries. Examples include CORBA, Java RMI, ACE, and ICE. When general-purpose communication middleware is used as a transport mechanism, all its features can be inherited by the RSF. Some of these features are very interesting: configurable sub-transport (TCP, UDP), security SSL, authentication, quality of service QoS. Other features can be inherited; for instance, the use of CORBA makes a naming service in a Remote RPC API isolation mechanism available (see "Robot Hardware Interfaces" in Section 4.3).

In general, RSFs and MASFs should improve the transport layer in order to support a wider set of choices. A well-designed MASF or RSF should maintain these aspects from the logic-level communication between two agents through the use of isolation mechanisms, such as configuration files.

Perhaps the best RSF in this aspect is YARP. It provides an acceptable set of implemented carriers and also provides a mechanism to implement custom carriers. This enables the application of YARP with most field buses. OROCOS is also special in this aspect since it provides support for the Ethercat and CanOPEN transport mechanisms. ROS also provides basic support for serial wire communications.

Message format and type marshaling

The message format constitutes a secondary aspect but is worth evaluating since it brings certain consequences. A suitable message format can promote interoperability between different platforms and languages. For example, the text-type message formats that are easily interpretable by parsers are very useful, especially XML, XDR, CSV, JSON, and YAML. Another advantage of the text-type format is that the content of the messages can be comprehensibly analyzed even by tools executed from a command console. Extended binary formats such YAML [90] are also recommended since they outperform text formats with regards to communication, memory, and computational requirements. Binary formats are less frequently used in MASFs than in RSFs.

Performance, energy consumption, and introspection facilities are affected according to the message format selected. Therefore, this aspect should be also considered in those applications whose constraints are related with communication performance or energetic autonomy. Applications based on mobile robots which work autonomously should especially consider this aspect: mobile sensor networks, swarm robots, and modular robots usually have significant energy autonomy constraints. The use of a binary data format facilitates better performance and energy saving. Furthermore, binary formats focused on performance are custom made and have no standardization. Therefore, introspection tools have to be specifically designed to handle this kind of format. On the other hand, text-based formats are more adequate for platform-independent introspection tools.

Concurrency model

This characteristic indicates if the framework organizes the agents by means of processes or threads. Processes are more flexible and can be distributed between different physical nodes of a network, whereas threads pertain to the same process and are sometimes useful for agents with closely related tasks where the velocity of communication is critical. There are frameworks in robotics that support mixed models. Frameworks that allow the modules to be organized as multi-thread usually internally implement the necessary memory protection and concurrency mechanisms so that the programmer uses this system transparently or semi-transparently.

4.3. *Other aspects of robotics frameworks*

The general characteristics of the RSFs are briefly described in Section 2 of the present study. In addition to the middleware characteristics of the communications, the RSFs present other aspects significant for Robotic systems development, some of which are analyzed in this section.

Development tools

Some frameworks provide a set of tools that facilitate robotics software development. Commands for the construction of the system tend to be of a regular type, and can solve problems of dependences on other modules of the framework. On the other hand, some frameworks allow the graphical creation and interconnection of system modules.

Deployment tools

Frameworks usually offer infrastructure to facilitate deployment tasks. This infrastructure typically appears in the form of tools and configuration files. Deployment tools are crucial in maintaining the scalability of the distributed and complex robotics architectures. The architecture of the system is defined and the participating nodes are specified in the configuration files. The configuration files can also define boot and connection parameters and mechanisms of communication between the nodes. The tools can be visual or command-lines and allow the start, stop and administration of system nodes, as well as the establishment of the initial parameters through configuration files, or the conducting of operations related to the naming service, such as renaming or name pushing.

Simulation capabilities

The simulators offer a visual representation of the problem and execution in virtual worlds of rigid solids. The frameworks studied are not focused on simulation capacities; nevertheless some frameworks have implemented modules that allow a rapid integration with simulators such as Stage, Gazebo, OpenRave, UsarSIM or Webots. Simulators play a major role by allowing the system to be modeled in the design stage, by finding errors in the early stages of development, and by saving time and money. They are also useful for testing ideas.

Robot hardware interfaces

RSFs usually support a set of hardware devices and robots. This is one of the lower-level aspects of the Robotics software Development (see Section 3.2). Most existing RSFs mentioned in Table 1 tackle this aspect, and commonly use one of the two following approaches for the implementation of the interface:

- *The API isolation approach*: This approach follows the layered architecture discussed in Section 2.3. An application programming interface (API) is designed in a language, such as C++, and Java. The specific code then implements such interface, usually through inheritance. Programmers are encouraged to make high-level algorithms over these interfaces instead of specific hardware code. The functionality is normally isolated in one of two ways:
  - Dynamic Link Library Isolation: The driver code is isolated in a dynamic library and loaded by the node at runtime. This is the simplest way of abstraction. It is adequate for rapid software development and higher driver performance. However problems arise when multi-language support or introspection compatibility is required. Clear examples of RSFs with this approach are MRPT [82] and Pyro [85].
  - RPC-API Server Isolation: The driver code is isolated in a server that is invoked by a RPC. The API internally calls a remote server which executes the requested function by means of a Client/Server Architecture between layers (see Section 2.3). This mechanism is transparent for the client (upper layer) since the code is coupled over a programming interface. This approach is more flexible, and enables the use of distributed processing, various programming languages in each layer, etc. The main drawback of this approach is that the latency of operations may increase and real-time capabilities may be lost, if the correct transport layer and operative system is not used. The most representative example of this approach is the Player/Stage RSF. This approach is typical in RSFs which use CORBA as a transport layer, given that CORBA provides itself with a distributed OOP mechanism. The multiple-language support is achieved through code generation from any Interface Definition Language, such as CORBA-IDL. Examples of RSFs using this approach are MIRO and MOOS.
- *The node isolation approach*: This approach is peculiar to P2P architectures (see Section 2.3). The device driver code is isolated in a component that can be instantiated as a node at runtime. This node uses the communication mechanisms explained in Section 4.2 (ports, topics, services, etc). The programmer incorporates these mechanisms explicitly in the node code, and hence the coupling point is located in the message data structure. Programmers are therefore encouraged to use standardized message types for reusability purposes. Messages types include: motor commands, images, clouds of points, and IMU sensor readings. Again the message data type may be problematic in multi-language RSFs. As in the RPC-API isolation approach, the main drawback of this approach is that the use of distributed communication mechanisms may increase the latency of operations and therefore real-time capabilities may be lost.

The difference between the node isolation approach (using a service communication mechanism) and the RPC-API approach are sometimes unclear, but the key differences between them are:

- The RPC-API approach imposes a strict client–server layered architecture for Robot hardware interfaces. In this architecture, the server cannot take the initiative since it only provides a set of services to the upper layer. In the node isolation approach, communication is balanced and both nodes can synchronously communicate to each other.
- The programming style in the node isolation approach with services is much more explicit and the coupling point is in the message data structure while the coupling point in RPC-API is the provided list of methods. Programming code can be more comfortable over RPC-API than over the service communication mechanism.

Robotics algorithms

Many RSFs offer reusable generic algorithms (see Section 3.2) in the sphere of robotics: kinematics, robot perception, Bayesian estimation, SLAM, motion planning and machine learning, to name but a few. These algorithms present one of the most important aspects for evaluation, since to some extent it measures the good design of an RSF and means that it has successfully overcome the chronic obstacles to the reuse and generalization of robotics technologies.

Real-time capabilities

One of the most important features in an RSF is Real-Time constraint support. In order for the whole system to be considered as having real-time capacities, all the hardware and software components used (operating system, communication protocol, and libraries) must fulfill these restrictions, which is why it is a very complex objective in a distributed robotic system. Support for Hard Real-Time constraints is unusual in the RSFs; nevertheless many are compatible with Real-Time Operating Systems (such as RTAI and RTLinux) or transport mechanisms which guarantee the real-time constraint systems such as ACE, (see under the heading of "Transport Mechanisms" in Section 4.2). These RSFs can support Soft Real-Time constraints, and offer better temporal behavior by being more determinist and efficient.

Real time is an important characteristic of many robotics systems and is the least taken into account in these frameworks [91]. In the majority of these frameworks, the nodes must carry out non-critical tasks without real-time restrictions (task planning, for example), which are not suitable for control laws.

The reason for this is probably that most of the existing RSFs have focused on the specific field of autonomous mobile robots, where real time is not so crucial. It is worth mentioning that OROCOS is the only RSF analyzed which tackles the real-time problem in an adequate manner. Nevertheless, most RSFs should improve this aspect, which is inherent to the majority of robotic systems. In order to improve this situation, suitable communication protocols such as CAN (Controller Area Network) should be supported to handle these restrictions. These types of protocols are not supported in the technologies studied here, which tend to employ IP networks. It is also necessary to provide basic libraries to deploy nodes in microcontrollers or simple processors. This heterogeneity, in the types of nodes and communications technology that these frameworks use, can complicate other aspects of the distributed architecture, for example, the naming service or lookup service.

5. Individual analysis and comparative summary

It is important to analyze and compare the RSFs and MASFs, in order to determine the similarities and differences, and to select the most suitable alternative for a specific robotics project where the application of a MAS-based solution is required. Following this criterion, the RSFs chosen for the comparative study are: ROS, YARP, OpenRDK, OpenRTM, OROCOS, and ORCA. As a reference, they will be compared with the MASFs: JADE and Mobile-C (in total this study includes 2 MASFs and 6 RSFs). These two MASFs have been selected from a wide assortment because they are probably the most extended nowadays and since they enclose most of the characteristics and features of MASFs (see Introduction).

Other RSFs are not included in this study since they present objectives that are very different from the creation of distributed architectures and are focused on aspects such as the reuse of drivers and algorithms, and simulation support. Some of these frameworks are: Player, MOOS, MIRO, JDE+, OpenRave and CARMEN. Even so, using these RSFs together with other general-purpose MASFs, such as JADE or Mobile-C, give possible mixed solutions [92], where the tools of each are used in a complementary manner. This is another reason why the analysis of JADE or Mobile-C is relevant for this comparison.

There follows a general description of the frameworks that are the objectives of the study, and finally a comparative study is summarized.

5.1. JADE (Java Agent Development Framework)

JADE [21,93,94,12] is a general-purpose MASF developed by Italian Telecom and the University of Parma. It is the *de facto* solution for multi-agent systems development. JADE implements the FIPA Standard, which promotes interoperability with other platforms. It currently has a high level of maturity, and thereby serves a wide community, and is well documented.

With JADE, hybrid P2P distributed architectures can be developed where the agents are hosted in containers of agents called agencies. Each computer can have one or several agencies, although it is usual to have only one. The agents identify themselves by means of a unique name that can be changed in execution and is managed by a naming-service system (White-Pages service). This system does not allow pushing mechanisms (or namespaces), nor absolute nor relative names. It offers a powerful lookup service (Yellow Pages service) that enables it to find agents that provide services with specific characteristics. JADE uses a concurrency model by way of processes for agents located in different agencies and threads for agents located in the same agency.

The agents communicate by means of the simple passing of messages (agent–agent). The messages used in JADE include various fields such as content, intention, ontology and content format. It also allows the mechanism of communication by topic and is useful when a communication channel needs to be disconnected between different agents. JADE is especially powerful in that it allows different presentations of the message format, including RDF and XML. At the transport level, the use of shared memory mechanisms for agents in the same agency also allows high-level transport mechanisms such as HTTP, with which HTTP off-the-shelf security mechanisms can be used in the communications. It also supports other methods of transport such as RMI, and CORBA (IIOP and JCIP). When two agents in the same agency communicate, these methods of transport are not used, but the object is cloned and its new reference is passed between threads making use of some available mechanisms in the virtual machine.

The main development language of JADE is Java, although it also allows other programming languages that are executed in the virtual Java machine: J2EE, J2ME and J2SE. The execution in a virtual Java machine entails certain disadvantages in robotics work since they are somewhat less suitable for interactive tasks with the hardware. In addition, a high level of indeterminism is introduced which complicates its use in systems with Real-Time constraints. In contrast, the use of a virtual machine provides agents with great portability and mobility. Different types of virtual JAVA machines are supported by many kinds of computers: servers, PCs, embedded systems etc. Some virtual machines need no operating system, which makes them sufficiently light for the less powerful computers. Another advantage of the Java platform is the mobility of agents via the nodes of the network; this is possible since the code of an agent is in a language independent from the hardware platform.

JADE also offers a set of libraries that facilitate typical interactions between agents by means of the use of patterns. It offers a large number of graphical tools to manage and monitor the status of agents and to monitor the message traffic of the system. Being of general-purpose MAS technology, neither drivers nor generic algorithms for robotics are implemented. Furthermore, JADE displays no compatibility with other RSFs in robotics or simulators.

5.2. Mobile-C

This is a framework for the development of specialized MASs in embedded systems [17,95]. Mobile-C originated in the University of California and the Michigan Technological University, and implements the OMG MASIF standard for multi-agent systems and also, albeit only partially,[3] the FIPA standard, which is why it is similar to JADE in many respects. The most special characteristic of Mobile-C is that it allows the creation of mobile agents developed in C/C++. It is a lower level language, which allows greater control over the hardware of the systems in which it is hosted (agencies).

It is important to emphasize that C/C++ is a language that, during its process of compilation, remains strongly coupled to a specific hardware architecture and operating system. Nevertheless, Mobile-C is able to implement the mobility of agents. To obtain the independence of the platform, C++ agents do not travel via the already compiled network but in a form of source code accompanied with their status variables.

Mobile-C is suitable for strict temporal requirements (Real-Time) since the C++ interpreter is temporally deterministic and is also compatible with real-time operating systems such as QNX. These characteristics can make it suitable for the programming of robotic systems. Mobile-C supports various operating systems in addition to QNX, such as Windows, Linux, OSX, and other UNIX operating systems such as Solaris.

The communication mechanism of Mobile-C is simple asynchronous message passing and the transport method used is HTTP. Mobile-C has a naming service that allows it to locate the different agents; however it does not allow pushing mechanisms (or namespaces) or absolute and relative names. It also has a powerful lookup service, as defined by the FIPA specification, which allows it to find agents that provide services with specific characteristics.

Mobile-C lacks drivers or generic algorithms for robotics since its sphere of application is wider. It does not display compatibility with other RSFs in robotics or simulators and lacks a good set of simulation, development, deployment and monitoring tools, which is why these tasks must be made manually or with off-the-shelf tools.

Mobile-C is used in various situations, such as: multi-sensor data fusion, multi-camera surveillance for intelligent homes, and urban traffic-signal control based on MAS.

5.3. ROS (Robot Operating System)

This is an RSF that is notable for being generalist and able to integrate with other existing technologies [67]. It arose as a spin-off from Willow Garage in collaboration with the University of Stanford. The latest release is ROS Electric Emys (30/7/2011) and a new release ROS Fuerte Turtle is due in March 2012.

This RSF has a numerous, active and growing community, which probably constitutes the most important factor toward making ROS one of the most complete RSFs today. Several organizations, such as companies and universities, are using or collaborating with ROS in their research, thereby forming a pool of federated repositories not controlled by Willow Garage. It is noteworthy the quality and quantity of its documentation and its fast growth. This has been promoted by the use of global wiki (http://www.ros.org) support for federated repositories and certain mechanisms for the automatic generation of documentation.

This RSF possesses a Hybrid P2P distributed architecture where a master node must be known.[4] This master node handles the naming service and the lookup service. The agent nodes communicate by means of message passing through topic and service mechanisms. Over these services another complex communication mechanism between nodes is provided for tasks of longer duration called *actions*. All these mechanisms are built over a TCP transport layer using a custom-built binary format of messages. Nonetheless, interactions with the master node are made through XML-RPC/TCP and experimental support for UDP transport is featured. Serial transport implementation (called Rosserial) is also provided especially for use with microcontrollers such as Arduino.

The name service let a hierarchical organization of the network resources like topics, services and nodes. ROS also has the so-called "pushing feature" to organize and identify properly these resources. These resources can be referenced in a relative or absolute manner thereby promoting reusability and avoiding problems of name conflicts. The master node provides a service lookup system to search any registered service that fulfills requirements for certain characteristics. This allows microcontrollers to belong to the P2P network. These transport capabilities are weak in comparison with other technologies, such as YARP.

ROS allows the use of multiple languages, although the main languages are C++ and Python. Other secondary languages include Java, LISP, Octave, and LabView. Regarding interoperability between languages, the connection points are the interfaces of messages and services. The key feature that makes ROS available in different languages is the set of tools for message code generation (genmsg and gensrv). These tools take an interface description language for messages (for topics) or services and generate specific language code.

The system is focused on working in Unix-based systems (Linux, OSX, etc). I. A small ROS library has been also implemented to be used on AVR microcontrollers using the so-called "Rosserial" transport mechanism, and support for developing nodes using an EtherCAT network is provided. However, ROS is inadequate for real-time applications since neither the underlying Operative System nor the main transport mechanism (TCP/IP) are real-time. Other RSFs, such as OROCOS, are best suited for real-time control tasks.

This RSF provides a good number of graphical tools and command lines for various purposes: development, deployment, management, monitoring and debugging of the distributed system, as well as 2D and 3D visualization that allows a virtual view of how the system interprets the real environment.

ROS implements the multi-process and the multi-thread concurrent model. The multi-thread concurrent model is useful when the transport mechanism becomes problematic due to intensive use of heavy data messages, such as images and clouds of points. Nodes in threads (called nodelets) provide the major advantage of making possible a zero-copy mechanism of messages through shared memory. However, they present two important drawbacks: nodelets are only available for the C++ language, and shared memory communication has to be explicit in the nodelet code. Other systems such as YARP enable the transport layer to be set at runtime.

Regarding certain aspects common to MAS, it is worth mentioning that neither agent mobility nor any agent communication standards are considered in ROS.

The application field of ROS is mainly focused on Service Robotics. Many successful experiments should be mentioned: robots that fold towels, several algorithms for robot navigation, robots that fetch you a beer from the fridge, those that open doors at home, and so on. In spite of the widespread growth of ROS in this application field, it is also used for underwater vehicles, Unmanned Aerial Vehicles, and industrial manipulators.

One of the strongest points of ROS is the high number of robotics software packages with drivers and robotics algorithms. Many

[3] It is nevertheless sufficient to be interoperable with other technologies, such as JADE, at the level of ACL-message passing, although not at the level of agent mobility.

[4] Nevertheless a pure P2P architecture with multiple master nodes is planned for the *Fuerte Turtle* Release.

areas are tackled. Numerous examples of low-level robotics algorithm packages include functionality for: Kinematics, Motion Planning, Bayesian Filtering, machine learning, image processing, and 3D Perception. Other higher level robotics algorithms or robotics functions include: navigation, object manipulation, grasping, and object recognition. All this variety is made possible by the philosophy of integration with other robotics libraries (OpenCV, OpenSLAM, etc). ROS offers a good level of compatibility with other robotics-related frameworks, such as YARP, OpenRTM, OROCOS, and OpenRave. It also provides a good support of simulators, such as Stage, Gazebo, and Webots.

With respect to Robot hardware interface, ROS uses the node-isolation approach (see Section 4.3). It provides a wide collection of implemented drivers that allow the management of hardware devices, such as laser rangers, sonars, joysticks, and cameras, and dozens of mobile and articulated robots (PR2, HERB, HRP2, Erratic, Qbo, etc).

5.4. *YARP (Yet Another Robot Platform)*

This is a very mature, light and flexible RSF that originated [4] in the University of Genoa and The Massachusetts Institute of Technology (MIT). It is oriented toward humanoid robotic systems, although it presents an excellent alternative for MAS development. YARP displays a P2P architecture with a multi-process concurrency model through a topics communication mechanism.[5]

It is highly interoperable, and works in multiple operating systems including Linux, Windows, OSX and QNX. Nowadays, YARP is probably the RSF that shows the best capacities for distributed robotics. It enables the design of Pure P2P architectures with a partially implemented discovery service. YARP offers a good naming-service mechanism that allows the definition of namespaces (pushing), and provides good programming support, allowing C++, Java, Python, and Matlab.

It is also especially suitable for applications with high communications requirements since it supports many efficient transport mechanisms. It is worth mentioning its multicast support, since it is an option that no other alternative implements and can be very efficient for sending heavy messages, (e.g. images and clouds of points) from one node to multiple nodes. The shared memory transport method is also very powerful and allows zero-copy messaging between nodes in the same machine.

The message format is proprietary (Bottle Format) but has a very simple text syntax that can easily be implemented from other basic platforms by means of sockets. This system offers the possibility of using many languages such as Python, Octave and Java since the YARP library is implemented through Simplified Wrapper and Interface Generator (SWIG).

It offers a large number of implemented drivers and makes use of a very clear Robot hardware interface based on both the Node isolation approach and the DLL-API approach. It also contains various high-level algorithms and has seamless compatibility with other RSFs such as Player/Stage/Gazebo, ROS, and OROCOS. No mobile agents are allowed nor are any agent communication standards implemented.

YARP is recommended for all of the applications mentioned in Section 2.2. However, it was originally designed to work on humanoid robots with multiple CPUs and other multi-agent control systems [4]. It has also been used successfully on heterogeneous robot teams with high communication bandwidth requirements due to the intensive use of images [1].

[5] YARP is termed a port, although according to the criteria of this article it is a topic, since it displays a Publish/Subscribe model. Furthermore, it is a system resource (not exclusively of one agent) and allows Many-to-Many asynchronous communication to be maintained by means of callbacks.

5.5. *OpenRDK*

This is an RSF developed by the Università Sapienza Di Roma [71]. Its main objective is to serve as a research platform for robotic systems for the RoCoCo research group. OpenRDK is a light RSF and its target is to find a fast way to develop robotic systems for little research projects.

It displays a hybrid P2P distributed architecture a dual concurrency model of threads and processes distinguishing between the concepts of agents (processes) and components (threads). The components are executed on the same computer and pertain to a single agent. The components can be communicated efficiently by means of shared memory mechanisms.

Agents and components can communicate by means of passing messages through the Ports and Properties mechanisms. For the distributed processes, the messages can be sent through TCP or UDP in XML or binary format. Nevertheless, the implementation of how an object in C++ is translated to and from these formats must be implemented manually for each type of message. This is usually a disadvantage, although in certain cases it can be useful to have greater control of the communications operation, for example, good manual serialization could optimize the communications performance.

OpenRDK includes a set of console commands for the monitoring and logging of the system in execution and a visual management and monitoring tool (rconsole). The development in this platform is by means of tools that already exist in Linux such as CMake and does not include any additional tool.

OpenRDK has compatibility with some simulators such as Stage, Gazebo, USARSim and Webots. It provides some implemented drivers for robot management such as Aldebaran's NaoQi, and a little set of drivers for scanners, lasers, cameras, etc. No mobile agents are allowed and no agent communication standard is implemented.

5.6. *OpenRTM (Open Robot Technology Middleware)*

This is an RSF developed by the Japanese National Institute of "Advanced Industrial Science and Technology (AIST)". OpenRTM, also known as OpenRTM-AIST [70], implements the RTC (Robot Technology Components) standard defined by the OMG [89]. OpenRTM is a mature project with a ten-year history. However, the current release is the OpenRTM 1.0 (January 2010), although the candidate OpenRTM 1.1 was released in June 2011. This RSF is especially accepted in Japan and has a large community and documentation. One of its drawbacks is that a large proportion of the documentation is only available in Japanese and has yet to be fully translated into English.

This is a hybrid P2P architecture constructed on CORBA which is why it inherits its characteristics of transport, serialization and interoperability between languages and platforms. OpenRTM is independent of the version of CORBA used, and supports several implementations such as OmniORB, ACE/TAO, MICO. At the transport level, inherently through CORBA, OpenRTM allows transports such as TCP, UDP, SSL and UNIX Sockets. OpenRTM uses the naming service of CORBA, where the agents can register. Nevertheless, it has no lookup nor discovery service to look for a specific agent with particular characteristics or services.

It provides the port, property and service communication mechanisms. It supports multiple languages such as C++, Python and Java thanks to the agnostic interface definition IDL (Intermediate Description Language) of CORBA. OpenRTM is available in Windows, Linux, and OSX.

OpenRTM offers a good set of development, deployment and monitoring tools. For example, OpenRTM has a command-line tool (rtc-template) which facilitates the development by being able to generate the proxy code from its IDL interface, for each of the supported languages. Furthermore, it has a visual tool (RTLink) integrated in the Eclipse development environment, which allows block diagram design of the interconnections between the different nodes and also specifies their initial configuration.

Furthermore, OpenRTM has a good number of drivers implemented for different types of robots, sensors and actuators. It is also compatible with simulators such as Stage, Gazebo and OpenHPR. OpenRTM does not implement mobile agent capacities or any agent communication standard.

5.7. OROCOS (Open Robot Control Software)

This RSF arose from a project promoted by the European Robotics Research Network (EURON), and was implemented mainly by the K.U. Leuven in Belgium. OROCOS [66] offers a very specialized framework for the development of industrial robotic systems. The framework focuses on the following aspects: Robot hardware interfaces and drivers of components for real-time applications. To this end, this RSF allows the use of various operating systems, including several with real-time capacities [59]: RTAI, Xenomai, and there are also positive experiences with other systems such as QNX and WxWorks. It also allows generic O.S. such as Linux, OSX, Windows, and Windows CE. This framework provides the infrastructure and the functionalities to build robotics applications in C++ as the main language, and other languages such as Python and Simulink. OROCOS also provides a custom script language to orchestrate the communications and interactions between nodes and state machines of each agent.

OROCOS proposes the use of hybrid P2P architecture of nodes that can communicate by means of port, service, event and property mechanisms. The distributed system is based on CORBA as a transport layer. CORBA components are typically presented by using various implementations of ORB, ACE or TAO, which presume real-time capabilities. Other transport mechanisms are supported such as EtherCAT and CANOpen with which real-time communications can be achieved.

OROCOS can use either the RPC API Isolation or the node-isolation mechanism as Robot hardware interface. The communication mechanisms between nodes are: ports, events and services. It is worth mentioning that even with this kind of Robot hardware interface, OROCOS can achieve real-time capabilities when the proper operative system (such as Xenomai) and transport layer (such as EtherCAT, and CANOpen) are used. It uses the naming service of CORBA and also provides its own lookup system. It may be the most industrially oriented RSF studied in this survey and makes emphasis in real-time which is one of the most forgotten aspects in the existing RSFs.

This RSF also provides a good set of robotics algorithms and libraries which focusing on aspects such as kinematics, motion control, and Bayesian filtering. Many drivers are also provided for robots of KUKA and some perception sensors such as cameras and laser rangers. It is partially integrated with ROS and YARP.

OROCOS have been applied intensively in real-time multi-agent control systems for manipulator robots and has also been used as the software infrastructure in the Team Berlin for the DARPA's Urban Challenge.

In the context of MAS development neither agent mobility nor agent communication standard is implemented.

5.8. ORCA (Open Robot Control Software)

This is a mature RSF developed by the Kungliga Tekniska Högskolan and currently mainly maintained by the Australian Centre for Field Robotics. It was originally a part of the OROCOS project, although these two branches never managed to merge. It allows the development of P2P distributed architectures where the nodes are processes that communicate by means of services. These services implement the functionality defined in stable interfaces that are well known for promoting reuse. ORCA uses the ICE naming service and has no mechanism of lookup or discovery service.

The supported operating systems are Linux, Windows, OSX and other UNIX systems. It also supports QNX; nevertheless ORCA does not offer its own mechanisms to confront problems with real-time restrictions.

ORCA uses ICE technology in its transport layer, which is why it uses the binary serialization mechanism that this technology provides. It also allows it to make TCP, UDP and SSL communications. The main development language is C++, although it has experimental and limited support for other languages such as Java, Python and C #.

ORCA provides visual and command-line development tools, which allow a convenient definition of the architecture. ORCA offers a good number of off-the-shelf components for the use of extended devices such as laser rangers, sonars, cameras, and joysticks. It supports no mobile agents nor does it implement any agent communication standard.

5.9. Comparative study

The comparative study is at this point summarized in three tables. For each group of aspects selected in the previous section, Section 4, a table of results is shown, where rows represent the RSFs and the columns give the described characteristics. Taking into account all these aspects and the most significant differences between the frameworks analyzed, an interpretation of this comparison in the fields of RSF and MASF is carried out in the following section.

Table 2 reviews the general aspects discussed in Section 4.1 for the proposed frameworks.

In accordance with the individual analysis, a comparative summary of those aspects related to the communications used in RSFs is presented in Tables 3 and 4. Note that implementation of the various RSFs and the MASFs have lead to significant differences and similarities between these aspects.

The last group of significant framework aspects discussed previously is summarized in Table 5 for the selected RSFs and MASFs.

As verified in this survey, there is a variety of technologies suitable for MARS development. For each framework studied, certain aspects can be highlighted. Mobile-C allows the mobility of agents programmed in C++. JADE is noted for its compliance with the FIPA standard and its acceptance by the researcher and development community, in addition to having the most powerful lookup service. YARP offers very flexible and optimal transport methods and shows the best distributed aspects, such as availability of a pure P2P Architecture and a very powerful message-passing mechanism (topic-port). ROS stands out for its transverse and integrating approach to the various RSFs and provides the wider set of robotics software packages for drivers and robotics algorithms. It offers great tools, possesses well-organized quality documentation, and boasts the most powerful naming service. OpenRDK offers an interesting dual approach of process (agent)/thread (driver) which is common in robotics but is seldom implemented in other RSFs. OpenRTM displays a mature platform with a large number of integrated development tools with environments, such as Eclipse, and offers a visual language that greatly facilitates the development tasks.

6. Conclusions

For many of the complex robotic systems application contexts, a solution based on Multi-Agent Systems is, in our opinion, the

Table 2
General aspects of the compared frameworks.

Name	Organization	License	Year	Latest release	OS	Programming language
JADE	Italian Telecom and University of Parma	LGPL	1998	JADE 4.1.1 November 2011	JVM Platforms: J2EE, J2SE, J2ME	Java JVM languages .NET languages
Mobile-C	University of California and Michigan Technological University	BSD	2006	Mobile-C 2.1.4 June 2011	Windows, Linux, OSX,QNX, other UNIX Systems	C/C++
OpenRDK	Università Sapienza di Rome	GPL2	2009	OpenRDK 2.1.5 June 2010	Linux, OSX	C++
OpenRTM	AIST (Japan)	LGPL	2002	OpenRTM 1.1 (RC3) June 2011	Linux, Windows	C++, Java, Python
OROCOS	K.U.Leuven in Belgium	LGPL	2000	OROCOS 2.5 October 2011	Linux/RTAI/Xenomay, OSX, Windows, WindowsCE	C++, Python, Simulink Orocos Scripting Language
ORCA	Kungliga Tekniska Högskolan Australian Centre for Field Robotics	LGPL	2004	ORCA 9.11 Nov. 2009	Linux, Windows, OSX, QNX	C++, experimental (Java, Python, PHP)
ROS	Willow Garage	BSD	2008	ROS 1.0.1 July 2011	Linux, OSX and Windows (limited and experimental)	C++, Python, Java, Lisp, Octave
YARP	University of Genoa and Michigan Institute of Technology	GPL2	2005	YARP 2.3.9 August 2011	Windows, Linux, OSX, QNX4	C/C++, SWIG languages (Python, Java, Octave)

Table 3
Aspects of distributed systems of the compared frameworks (1).

Name	Distributed architecture	Node communication mechanisms	Naming service	Lookup service	Discovery service
JADE	Hybrid P2P	Simple-messages, topics, complex interactions	Yes	Yes	No
Mobile-C	Hybrid P2P	Simple-messages	Yes	Yes	No
ROS	Hybrid P2P/Pure P2P (planned)	Topics, properties, services	Yes, pushing, remapping, rel./abs. addresses	Yes	No
YARP	Hybrid/Pure P2P	Ports, topics	Yes, pushing, rel./abs. addresses	No	No
OpenRDK	Hybrid P2P	Ports, properties	Yes, remapping	Yes	No
OpenRTM	Hybrid P2P	Ports, services, Properties	Yes	No	No
OROCOS	Hybrid P2P	Ports, services, events, properties	Yes	Yes	No
ORCA	Hybrid P2P	Services	Yes	No	No

Table 4
Aspects of distributed systems of the compared frameworks (2).

Name	Multi-agent or robotics standard	Message format and type marshaling	Concurrency model	Message transport	Agent mobility	Distributed management tools	Introspection tools
JADE	FIPA	XML, RDF, Java serialization	Processes, threads	RMI, CORBA, HTTP, JICP	Yes	Yes GUI and command-line	Yes GUI monitoring and management tools
Mobile-C	FIPA MASIF	XML	Processes	HTTP	Yes	No	No
ROS	No	Custom binary format	Processes, threads	TCP, serial (own protocol), UDP (experimental)	No	Yes command-line	Console monitoring, Graphical (rviz)
YARP	No	Custom format (Bottle)—binary and text	Processes	ACE, TCP, UDP, shared memory (locally), multicast (LAN), TCP/XML-RPC	No	Yes command-line	Console monitoring tools
OpenRDK	No	Binary (manual), XML (manual)	Processes, threads	TCP, UDP, shared memory (locally)	No	Yes command-line	Yes
OpenRTM	RTC	Corba serialization	Processes	CORBA (OmniORB, ACE/TAO, MICO) (TCP, UDP, SSL, UNIX)	No	Yes command-line	Yes
OROCOS	No	Corba serialization	Processes, threads	CORBA (OmniORB, ACE/TAO), (TCP, UDP, SSL, UNIX), MQueue, EtherCAT, CanOPEN	No	No	Yes
ORCA	No	ICE serialization	Processes	ICE (TCP, UDP, SSL)	No	No	Yes

most suitable. These contexts include heterogeneous mobile robot teams, robots working in ambient intelligence environments, and mobile sensor networks based on robots. MASFs and RSFs provide technological solutions to tackle the MARS development.

It is noteworthy that, as shown in this work, RSFs and MASFs provide distinct features for the development of MARS, due to the differences in the areas to which they have been applied so far. On one hand, MASFs provide a general framework to build MAS by focusing on features such as agent mobility, interaction among agent patterns, and ontologies. On the other hand, RSFs are focused on developing robotics systems: robot hardware interfaces, algorithms reusability, and introspection mechanisms.

Table 5
Other aspects of robotics frameworks.

Name	Development tools	Deployment tools	Simulation capabilities	Robot hardware interfaces and drivers	Robotics algorithms	Real-time capabilities
JADE	No	Deploy configuration.	No	Node isolation but no drivers	No support	No
Mobile-C	No	No	No	Node isolation but no drivers	No support	Yes
ROS	Command-line	Command-line, deploy configuration	Stage, Gazebo	Node isolation	High support	No
YARP	No	Command-line, deploy configuration	Stage	DLL-API isolation and node isolation	Medium support	Yes
OpenRDK	No	Command-line, deploy configuration	Webots, USARSim, Stage, Gazebo	Node isolation	Low support	No
OpenRTM	Command-line and visual tools	Command-line, deploy configuration	Stage, Gazebo, OpenHPR	RPC-API isolation and node isolation	Medium support	No
OROCOS	No	Deploy configuration	No	RPC-API isolation and node isolation	Medium support	Yes
ORCA	Visual tools	Deploy configuration	Stage, Gazebo	RPC-API isolation	Medium support	No

It is also well-known that although MASFs have been applied historically to wireless sensor networks (WSN), currently RSFs are threatening to replace MASFs in this field. This is due to the fact that RSFs present better support for this kind of application due to their provision of a wider support for sensors and actuators (which are typical in WSN applications).

Nevertheless, RSFs should acquire features, such as agent mobility and the use of communication standards, from MASFs. Moreover the complex interaction mechanisms between agent nodes (beyond services, topics and ports) defined in FIPA and implemented in MASFs, such as JADE, clearly supercede the elemental mechanisms provided by RSFs. Certain RSFs, such as ROS, already implement some more advanced interaction protocols, for example those called "actions".

According to the discussion in Section 2.3, at first, using RSFs and MASFs together in a layered architecture could be conceived as a feasible approach. However, a strict division of the architecture in these two layers can be redundant, since the functionality is duplicated for each layer. It may become counterproductive by limiting scalability, flexibility and efficiency of the system. Despite this drawback, RSFs could greatly benefit by incorporating certain aspects from the MASFs and from the agent paradigm, which can provide a unified and non-layered architecture to the robotic system. The analyzed RSFs offer the necessary infrastructure to develop a MARS.

It has been discussed and demonstrated that even the lower software layers need the benefits that a P2P distributed and decoupled architecture provides: naming services, lookup services, communication mechanisms, etc. Conversely, it is clear that, at runtime, agent nodes may need to interact with other basic nodes regardless of the abstraction level of the performed task (drivers, pure reactive nodes without initiative, etc.), which do require a more hierarchical architecture.

Robotics software evolution reflects general-purpose software engineering (GPSE). Some ideas [7–9], such as the component-based robot software, the Event Driven Architecture, the Model Driven Architecture, and many of the techniques for robot distributed communications were tackled for GPSE some years ago and are being taken into account in current RSFs. Therefore, it is reasonable imagine that many current ideas of GPSE will be incorporated into robotics systems in the future.

In the context of MARS, the proposed P2P architecture clearly reflects Service Oriented Architecture (SOA) 2.0 concepts. Although SOA is intrinsically tied to Internet and to non-real-time applications, it would be interesting to investigate into which features of SOA can be incorporated into current RSFs for the development of MARS. It is highly probable that other concepts, such as orchestration of services and business processes for robotics, will become the focus of research in the near future.

Acknowledgments

The authors would like to acknowledge the contributions of Alessandro Farinelli from the University of Verona (Italy).

This work has been partly supported by Spanish government grants VULCANO (TEC2009-10639-C04-02) and under Andalusian Government Excellence Research projects P06-TIC-02298 and P08-TIC-03862 (CARISMA Project).

References

[1] Luis Merino, Fernando Caballero, J. Ramiro Martinez de Dios, Joaquin Ferruz, Aníbal Ollero, A cooperative perception system for multiple UAVs: application to automatic detection of forest fires, Journal of Field Robotics 23 (3–4) (2006) 165–184.
[2] Lynne E. Parker, Multiple mobile robot teams, path planning and motion coordination, in: Encyclopedia of Complexity and Systems Science, 2009, pp. 5783–5800.
[3] Anders Orebäck, Henrik I. Christensen, Evaluation of architectures for mobile robotics, Autonomous Robots 14 (1) (2003) 33–49.
[4] Paul Fitzpatrick, Giorgio Metta, Lorenzo Natale, Paul M. Fitzpatrick, Towards long-lived robot genes, Robotics and Autonomous Systems 56 (1) (2008) 29–45.
[5] David Kortenkamp, Reid G. Simmons, Robotic systems architectures and programming, in: Bruno Siciliano, Oussama Khatib (Eds.), Springer Handbook of Robotics, Springer, 2008, pp. 187–206.
[6] A. Farinelli, G. Grisetti, L. Iocchi, Design and implementation of modular software for programming mobile robots, International Journal of Advanced Robotic 3 (2006) 037–042.
[7] Blocks, Reusable Building, Component-based robotic engineering (part I), 2009.
[8] Alex Brooks, et al. Towards component-based robotics, in: Intelligent Robots and Systems, IROS 2005, 2005 IEEE/RSJ International Conference on IEEE, 2005, p.p. 163–168.
[9] B.Y. Davide, Brugali, Azamat Shakhimardanov, Component-based robotic engineering (part II), 2010.
[10] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, Hitoshi Matsubara, Robocup: a challenge problem for AI, AI Magazine 18 (1) (1997) 73–85.
[11] Janan Zaytoon, Jean-Louis Ferrier, Juan Andrade-Cetto, Joaquim Filipe (Eds.), ICINCO 2007, Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics, Robotics and Automation 2, Angers, France, May 9–12 2007, INSTICC Press, 2007.
[12] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, Giovanni Rimassa, JADE: a software framework for developing multi-agent applications: lessons learned, Information & Software Technology 50 (1–2) (2008) 10–21.
[13] C. Bumer, M. Breugst, S. Choy, T. Magedanz, Grasshopper—a universal agent platform based on OMG MASIF and FIPA standards, in: Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications, MATA'99, Ottawa, Canada, October 1999, pp. 1–18.

[14] Nick Howden, Ralph Rönnquist, Andrew Hodgson, Andrew Lucas, Intelligent agents—summary of an agent infrastructure, in: 5th International Conference on Autonomous Agents, 2001.
[15] Aaron Helsinger, Michael Thome, Todd Wright, Cougaar: a scalable, distributed multi-agent architecture, in: SMC (2), IEEE, 2004, pp. 1910–1917.
[16] V. Andronache, M. Scheutz, APOC: A Framework for Complex Agents, in: Proc. of AAAI Spring Symp, AAAI Press, 2003.
[17] Bo Chen, Harry H. Cheng, Joe Palen, Mobile-*c*: a mobile agent platform for mobile c/c++ agents, Software: Practice and Experience 36 (15) (2006) 1711–1733.
[18] A. Makarenko, A. Brooks, T. Kaupp, On the benefits of making robotic software frameworks thin, in: Workshop for Measures and Procedures for the Evaluation of Robot Architectures and Middleware at IROS'07, 2007.
[19] James Kramer, Matthias Scheutz, Development environments for autonomous mobile robots: a survey, Autonomous Robots 22 (2) (2007) 101–132.
[20] Nader Mohamed, Jameela Al-Jaroodi, Imad Jawhar, Middleware for robotics: a survey, in: RAM, IEEE, 2008, pp. 736–742.
[21] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa, Developing multi-agent systems with JADE, in: Agent Theories, Architectures, and Languages, 2000, pp. 89–103.
[22] M. Drummond, J. Bresina, S. Kedar, The entropy reduction engine: integrating planning, scheduling, and control, SIGART Bulletin 2 (1991) 61–65.
[23] Ron Sun (Ed.), Cognition and Multi-Agent Interaction: from Cognitive Modeling to Social Simulation, Cambridge University Press, Cambridge, ISBN: 0521839645, 2006.
[24] F. Heylighen, Cognitive levels of evolution: from pre-rational to meta-rational, in: F. Geyer (Ed.), The Cybernetics of Complex Systems: Self-Organisation, Evolution and Social Change, Intersystems, Salinas, CA, 1991.
[25] Sébastien Picault, Anne Collinot, Designing social cognition models for multi-agent systems through simulating primate societies, in: Proceedings of ICMAS'98, 3rd International Conference on Multi-Agent Systems, 1998.
[26] N. Bredeche, J.-D. Zucker, From distributed robot perception to human topology: a learning model, in: Proceedings of DARS'2000, 2000.
[27] K. Kawamura, T.E. Rogers, X. Ao, Development of a cognitive model of humans in a multi-agent framework for human–robot interaction, in: AAMAS'02, Bologna, Italy, July 15–19, 2002.
[28] Xiaocong Fana, Po-Chun Chenb, John Yenc, Learning HMM-based cognitive load models for supporting human-agent teamwork, Cognitive Systems Research 11 (1) (2010) 108–119.
[29] Sarvapali D. Ramchurn, Dong Huynh, Nicholas R. Jennings, Trust in multi-agent systems, The Knowledge Engineering Review 19 (1) (2004) 1–25.
[30] R.A. Brooks, A robust layered control system for a mobile robot, IEEE Journal of Robotics and Automation RA-2 (1986) 14–23.
[31] E. Gat, Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots, SIGART Bulletin 2 (1991) 17–74.
[32] T.M. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, J.C. Schlimmer, Theo: a framework for self-improving systems, in: K. VanLehn (Ed.), Architectures for Intelligence, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, pp. 323–355.
[33] J.G. Carbonell, C.A. Knoblock, S. Minton, PRODIGY: an integrated architecture for prodigy, in: K. VanLehn (Ed.), Architectures for Intelligence, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, pp. 241–278.
[34] P. Langely, K.B. McKusick, J.A. Allen, W.F. Iba, K. Thompson, A design for the ICARUS Architecture, SIGART Bulletin 2 (1991) 104–109.
[35] B. Hayes-Roth, An integrated architecture for intelligent agents, SIGART Bulletin 2 (1991) 79–81.
[36] D.R. Kuokka, MAX: a meta-reasoning architecture for 'X', SIGART Bulletin 2 (1991) 93–97.
[37] S. Vere, T. Bickmore, A basic agent, Computational Intelligence 6 (1990) 41–60.
[38] J. Laird, A. Newell, P. Rosenbloom, Soar: an architecture for general intelligence, Artificial Intelligence 33 (1987) 1–64.
[39] K. VanLehn, W. Ball, Goal reconstruction: how teton blends situated action and planned action, in: K. VanLehn (Ed.), Architectures for Intelligence, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, pp. 147–188.
[40] S.J. Russell, An architecture for bounded rationality, SIGART Bulletin 2 (1991) 146–150.
[41] T. Lux, The socio-economic dynamics of speculative markets: interacting agents, chaos, and the fat tails of return distributions, Journal of Economic Behavior and Organization 33 (1998) 143–165.
[42] Kerstin Dautenhahn, Embodiment and interaction in socially intelligent life-like agents, in: C. Nehaniv (Ed.), Computation for Metaphors, Analogy, and Agents, in: LNCS, vol. 1562, 1999, pp. 102–141.
[43] D. Oviedo, M.C. Romero-Ternero, M.D. Hernández, A. Carrasco, F. Sivianes, J.I. Escudero, Model of knowledge spreading for multiagent systems, in: 12th International Conference on Enterprise Information Systems, Madeira, Portugal, 8–12 June, 2010.
[44] J. Martínez-Miranda, J. Pavón, Modeling trust into an agent-based simulation tool to support the formation and configuration of work teams, in: 7th International Conference on Practical Applications of Agents and Multiagent Systems, 2009.
[45] A. Carrasco, M.C. Romero-Ternero, F. Sivianes, M.D. Hernandez, J.I. Escudero, Multi-agent and embedded system technologies applied to improve the management of power systems, JDCTA: International Journal of Digital Content Technology and its Applications 4 (1) (2010) 79–85.
[46] P. Smart, T. Huynh, D. Braines, K. Sycara, N. Shadbolt, Collective cognition: exploring the dynamics of belief propagation and collective problem solving in multi-agent systems, in: Network-Enabled Cognition: The Contribution of Social and Technological Networks to Human Cognition, Lulu Press, Raleigh, North Carolina, USA, 2010.
[47] Alessandro Saffiotti, Mathias Broxvall, Marco Gritti, Kevin LeBlanc, Robert Lundh, Md. Jayedur Rashid, B.S. Seo, Young-Jo Cho, The peis-ecology project: vision and results, in: IROS, IEEE, 2008, pp. 2329–2335.
[48] José Luis Sevillano, Jorge L. Falcó, Julio Abascal, Antón Civit Balcells, Gabriel Jiménez, Roberto Casas, Saturnino Vicente-Diaz, On the design of ambient intelligent systems in the context of assistive technologies, in: ICCHP, 2004, pp. 914–921.
[49] A. Saffiotti, S. Coradeschi, Symbiotic robotic systems: humans, robots, and smart environments, IEEE Intelligent Systems 21 (3) (2006) 82–84.
[50] A. Cesta, G. Cortellessa, R. Rasconi, F. Pecora, M. Scopelliti, L. Tiberio, Monitoring older people with the robocare domestic environment: interaction synthesis and user evaluation, Computational Intelligence 27 (1) (2011) 60–82.
[51] Zack J. Butler, Alfred A. Rizzi, Distributed and cellular robots, in: Springer Handbook of Robotics, 2008, pp. 911–920.
[52] Mark Yim, David Duff, Kimon Roufas, Polybot: a modular reconfigurable robot, in: ICRA, IEEE, 2000, pp. 514–520.
[53] Francesco Mondada, Giovanni C. Pettinaro, André Guignard, Ivo W. Kwee, Dario Floreano, Marco Dorigo, Luca Maria Gambardella, Stefano Nolfi, Jean-Louis Deneubourg, Swarm-bot: a new distributed robotic concept, Autonomous Robots 17 (2–3) (2004) 193–221.
[54] B. Anthony Kadrovach, Gary B. Lamont, A particle swarm model for swarm-based networked sensor systems, in: SAC, ACM, 2002, pp. 918–924.
[55] C. Cianci, X. Raemy, J. Pugh, A. Martinoli, "Communication in a swarm of miniature robots: the e-puck as an educational tool for swarm robotics" on swarm robotics, 2006, pp. 103–115.
[56] Andrew Howard, Maja J. Mataric, Gaurav S. Sukhatme, An incremental self-deployment algorithm for mobile sensor networks, Autonomous Robots 13 (2) (2002) 113–126.
[57] Andrew Howard, Maja J. Mataric, Gaurav S. Sukhatme, Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem, 2002, pp. 299–308.
[58] A. Makarenko, H. Durrant-Whyte, Decentralized data fusion and control in active sensor networks, in: The 7th International Conference on Information Fusion, Fusion'04, Stockholm, Sweden, 2004, pp. 479–486.
[59] Bas. Burgers, Automated I/O access with CoE in Orocos, Control Engineering M.Sc. Thesis, University of Twente, 2010.
[60] Berthold Baeuml, Towards the evaluation of software concepts for complex mechatronic systems, in: Erwin Prassler, Klas Nilsson, and Azamat Shakhimardanov (Eds.), IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'07, Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware, November 2007.
[61] Hiroshi Kimura, Yasuhiro Fukuoka, Avis H. Cohen, Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts, International Journal of Robotics Research 26 (5) (2007) 475–490.
[62] Angel Jiménez-Fernandez, Rafael Paz-Vicente, Manuel Rivas, Alejandro Linares-Barranco, Gabriel Jiménez, Antón Civit, Aer-based robotic closed-loop control system, in: ISCAS, 2008, pp. 1044–1047.
[63] Patricio Nebot, Joaquín Torres-Sospedra, Rafael J. Martínez, A new HLA-based distributed control architecture for agricultural teams of robots in hybrid applications with real and simulated devices or environments, in: Sensors in Agriculture and Forestry, Sensors 11 (4) (2011) 4385–4400. (special issue).
[64] Zhongquan Xie, Kumiko Miyazaki, Openization, standardization and diversification in the case of robotics software sector in Japan, 10, 2009.
[65] Toby H.J. Collett, Bruce A. MacDonald, Brian P. Gerkey, Player 2.0: toward a practical robot programming framework, in: Proc. of the Australasian Conf. on Robotics and Automation, ACRA, Sydney, Australia, 2005.
[66] Herman Bruyninckx, Open robot control software: the OROCOS project, in: ICRA, IEEE, 2001, pp. 2523–2528.
[67] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, Andrew Y. Ng, ROS: an open-source robot operating system, in: ICRA Workshop on Open Source Software, 2009.
[68] G. Metta, P. Fitzpatrick, L. Natale, Yarp: yet another robot platform, in: Software Development and Integration in Robotics, International Journal of Advanced Robotic Systems 3 (1) (2006) (special issue).
[69] Rosen Diankov, James Kuffner, Openrave: a planning architecture for autonomous robotics, Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008.
[70] Noriaki Ando, Takashi Suehiro, Tetsuo Kotoku, A software platform for component based rt-system development: OpenRTM-aist, in: SIMPAR'08: Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 87–98.
[71] Daniele Calisi, Andrea Censi, Luca Iocchi, Daniele Nardi, OpenRDK: a modular framework for robotic software development, in: IROS, IEEE, 2008, pp. 1872–1877.

[72] Michael R. Benjamin, White paper software architecture and strategic plans for undersea cooperative cueing and intervention, 2007.
[73] Stefan Enderle, Hans Utz, Stefan Sablatnög, Steffen Simon, Gerhard Kraetzschmar, Günther Palm, Miro: middleware for autonomous mobile robots, in: Telematics Applications in Automation and Robotics, 2001.
[74] J.M. Canas, D. Lobato, P. Barrera, Jde: an open-source schema-based framework for robotic applications, in: Davide Brugali, Christian Schlegel, Issa A. Nesnas, William D. Smart, Alexander Braendle (Eds.), IEEE ICRA 2007 Workshop on Software Development and Integration in Robotics, SDIR-II, IEEE Robotics and Automation Society, 2007.
[75] Alex Brooks, Tobias Kaupp, Alexei Makarenko, Stefan Williams, Anders Orebäck, Orca: a component model and repository, in: Davide Brugali (Ed.), Software Engineering for Experimental Robotics, in: Springer Tracts in Advanced Robotics, vol. 30, Springer-Verlag, Berlin, Heidelberg, 2007.
[76] Carle Côté, Yannick Brosseau, Dominic Létourneau, Clément Raïevsky, Francois Michaud, Robotic software integration using MARIE, International Journal of Advanced Robotic Systems 3 (1) (2006) 055–060.
[77] Michael Montemerlo, Nicholas Roy, Sebastian Thrun, Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) toolkit, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS, 2003, pp. 2436–2441.
[78] Byoungyoul Song, et al., An introduction to robot component model for OPRoS (open platform for robotic services), Autonomous Robots (2008) 592–603.
[79] I.A. Nesnas, Claraty: a collaborative software for advancing robotic technologies, in: Proc. of NASA Science and Technology Conference, 2007. http://esto.nasa.gov/conferences/nstc2007/papers/Nesnas_Issa_C10P2_NSTC-07-0088.pdf (accessed 25. 01. 12).
[80] Carle Côté, et al., Using MARIE for mobile robot component development and integration, in: Software Engineering for Experimental Robotics, Springer Verlag, 2007, pp. 211–230.
[81] Kurt Konolige, Saphira robot control architecture, SRI Int. (22), 2002.
[82] J.L. Blanco Claraco, Development of scientific applications with the mobile robot programming toolkit: the MRPT reference book, University of Malaga, 2010.
[83] Microsoft, Microsoft robotics developer studio, Writing, 2011. http://www.microsoft.com/robotics/.
[84] Alessandro Saffiotti, Mathias Broxvall, Ecologies: ambient intelligence meets autonomous robotics, Cognition (2005) (October).
[85] Douglas Blank, Lisa Meeden, Holly Yanco, The Pyro toolkit for AI and robotics, Science 27 (1) (2005).
[86] Webots-user guide, Cyberbotics. http://www.cyberbotics.com.
[87] D. Vallejo, J. Albusac, J.A. Mateos, C. Glez-Morcillo, L. Jimenez, A modern approach to multiagent development, Journal of Systems and Software 83 (3) (2010) 467–484.
[88] Dejan S. Milojicic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny B. Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, Jim White, Masif: the OMG mobile agent system interoperability facility, Personal and Ubiquitous Computing 2 (2) (1998).
[89] Object Management Group, Robotic technology component specification version 1.0, formal/2008-04-04.
[90] Oren Ben-Kiki, Clark Evans, Brian Ingerson, YAML ain't markup language (YAML) (tm) 1.0, Working draft, July 2002. YAML.org.
[91] B. Bauml, G. Hirzinger, When hard real-time matters: Software for complex mechatronic systems, Robotics and Autonomous Systems 56 (1) (2008) 5–13.
[92] Patricio Nebot, Enric Cervera, Agent-based application framework for multiple mobile robots cooperation, in: ICRA, IEEE International Conference on Robotics and Automation, IEEE, 2005, pp. 1509–1514.
[93] Fabio Bellifemine, G. Caire, A. Pogg, G. Rimassa, JADE—a white paper, Technical Report 3, Telecom Italia Lab, EXP Online, 2003.
[94] Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood, Developing Multi-Agent Systems with JADE, Wiley, 2007.
[95] Yu-Cheng Chou, David Ko, Harry H. Cheng, An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems, Information and Software Technology 52 (2) (2010) 185–196.