A framework for the development of cooperative robotic applications

Patricio Nebot and Enric Cervera
Robotics Intelligence Lab
Jaume-I University
Castelló, Spain
{pnebot, ecervera}@uji.es

Abstract— **The advances in mobile robotics, computing power, and wireless communications have turned feasible the development of communities of autonomous robots. In the last years, there is a greater interest in systems of multiple autonomous robots for the accomplishment of cooperative tasks. This project takes it into account and presents an architecture for the development of collaborative applications for a heterogeneous team of mobile robots based on embedded agents. The implemented architecture has the capacity to allow the team of robots to accomplish such tasks, and also it has tools and features that allow programmers to develop complex applications in a reasonable time. An agent-based development platform has been used for this purpose, due to it integrates the necessary capabilities for developing distributed applications, and manages the communications among all the components. To demonstrate the advantages of the architecture a real application, robot following, has been implemented. It consists of a set of embedded agents, which run in real robot systems, demonstrating that embedding agents in distributed intelligent systems is a powerful method for quick application development in the multirobot domain.**

I. Introduction

This article describes the implementation and development of a distributed architecture for the programming and control of a team of coordinated heterogeneous mobile robots, which are able to collaborate among them and with people in the accomplishment of tasks of services in daily environments.

The presented agent-based architecture, Acromovi, was born from the idea that teamworking is an essential capability for a group of multiple mobile robots [12], [14].

In the last years, there is an increasing interest in the development of systems of multiple autonomous robots, so that they exhibit collective behaviour. This interest is due to the fact that having one single robot with multiple capabilities may waste resources. Different robots, each one with its own configuration, are more flexible, robust and cost-effective. Moreover, the tasks to achieve may be too complex for one single robot, whereas they can be effectively done by multiple robots [1].

Acromovi architecture is a framework for application development by means of embedding agents and interfacing agent code with native low-level code. It addresses the implementation of resources sharing among all the group of robots. Cooperation among the robots is also made easier in order to achieve complex tasks in a coordinated way. Also, Acromovi architecture is a distributed architecture that works as a middleware of another global architecture for programming robots.

Though robot programming has been extensively done in C or C++ languages, a Java- based multiagent development system was chosen to develop the architecture of our team of robots [15]. Any common agent architecture can be used, but it has been implemented by means of the JADE (Java Agent DEvelopment Framework), a tool for the development of multiagent systems, implemented in JAVA, that fulfils with the FIPA specifications.

The embedded agents that constitute the Acromovi architecture work as interfaces between the applications and the physical elements of the robots. Some agents also make easier the handle of these elements and provide higher-level services.

This article is an updated version of previous works that used the Acromovi architecture. In those previous works, Acromovi was implemented by means of the MadKit multiagent platform, but in this new version, it is implemented with the Jade multiagent framework. In the same way, the Robot Following application is explained. With this application we have proved that the application works as well with Jade as with MadKit.

In Section II, the state of the art related with this work is mentioned. In Section III an overview of the Acromovi architecture is given. The different agents that constitute the Acromovi architecture and their functions are then described in Section IV. The Robot Following application is finally described in Section V, before concluding and presenting final remarks.

II. State of art

The amount of research in the field of cooperative mobile robotics has grown progressively in recent years [5], [16]. Some examples of these works are presented below, emphasizing their potentials, and whether they are related (or not) with embedded agents in mobile robotics.

CEBOT (CEllular roBOTics System) is a hierarchical decentralized architecture inspired by the cellular organization of biological entities. It is capable of dynamically reconfiguring itself to adapt to environment variations [9]. It is composed by ”cells”, in the hierarchy there are ”master cells” that coordinate subtasks and communicate among them.

0-7803-9177-2/05/$20.00©2005 IEEE

On the other hand, *SWARM* is a distributed system made up of a great number of autonomous robots. It has been called the term "SWARM intelligence" to define the property of systems of multiple non-intelligent robots to exhibit a collective intelligent behaviour. Generally, it is a homogeneous architecture, where the interaction is limited to nearest neighbours [11].

Other interesting project, *MICRobES*, is an experiment of collective robotics that tries to study the long time adaptation of a micro-society of autonomous robots in an environment with humans. Robots must survive in this environments as well as cohabit with their people [17].

In an attempt to use traditional IA techniques, the *GOPHER* architecture was thought to resolve problems in a distributed manner by multirobots in internal environments [4]. A central tasks planning system (CTPS) communicates with all the robots and disposes a global vision of the tasks that has been done and of the availability of the robots to perform the rest of tasks.

In order to reuse native software components, agents can be embedded in an interface level or middleware. *ThinkingCap-II* [3] is an architecture developed, in a project of distributed platform based on agents for mobile robots [10]. It includes intelligent hybrid agents, a planning system based on a visual tracking, vision components integration, and various navigation techniques. Furthermore, it has been developed over a real-time virtual machine (RT-Java), implementing a set of reactive behaviours.

Finally, *Miro* is a middleware for create autonomous mobile robot applications from the University of Ulm. It is based on the construction and use of object-oriented robot middleware to make the development of mobile robot applications easier and faster, and to foster portability and maintainability of robot software [7]. This middleware also provides generic abstract services like localization or behaviour engines, which can be applied on different robot platforms with virtually no modifications.

The previous works implement architectures and systems so that teams of mobile robots can make cooperative tasks. Our work consists of the implementation of an architecture of such type, emphasizing the reusability of software, allowing the programmer to seamlessly integrate native software components (vision libraries, navigation and localization modules) in the Java-programmed agent-based architecture. In this way, embedding agents is the key for powerful distributed applications being rapidly developed.

Also, in this work is emphasized other important characteristics, the sharing of resources from a robot among all the team and the easy access to the elements of the robot by the applications.

III. Acromovi architecture overview

To establish mechanisms of cooperation between robots implies to consider a problem of design of cooperative behaviour given a group of robots, an environment and a task, how the cooperation must be carried out. Such problem implies several challenges, emphasizing among them the definition of the architecture of the group. The multiagent systems are the natural environment for such groups of robots, making possible the fast implementation of powerful architectures for the specification and execution of tasks.

A. Architecture design

The robots that constitute the team have already a two-layer programming architecture with native (C/C++) libraries for all their accessories. The lower layer, ARIA, takes care of the requests of the programs to the robot components. The upper layer, Saphira, provides services for range sensor interpretation, map building, and navigation [13].

Acromovi architecture is able to subsume any other extra software, like the ACTS (a colour tracking library) and the Vislib (a framegrabbing library). ACTS is a native (C/C++) module which allows the applications track up to coloured objects which colour the users have been selected previously. ACTS is an external program that works as a server, and the applications work as clients that communicate with ACTS server by means of an ARIA library. On the other hand, Vislib is a library that provides a fast and flexible image processing and single-camera machine vision. The image processing functions that Vislib offers are generic convolution, filtering and morphological tools. Vislib is written in the C language.

Subsuming existing C/C++ libraries is a powerful and quick method for rapidly developing Java code and thus embedding agents with all the functionality of native systems. Furthermore, the new code adds seamless integration with other distributed agents, either running on remote applications, or even as applets in web pages.

However, the existing native code is oriented to both local and remote processing from only one controller, without defining collaboration mechanisms between robots or applications. Acromovi architecture tries to overcome this problem by the introduction of a new level over this architecture that allows an easy collaboration and cooperation.

As can be seen in Fig. 1, Acromovi architecture includes a middleware between the robot architecture and the applications, that allows the collaboration and cooperation of the robots in the team. This middleware is based on an agent oriented focus.

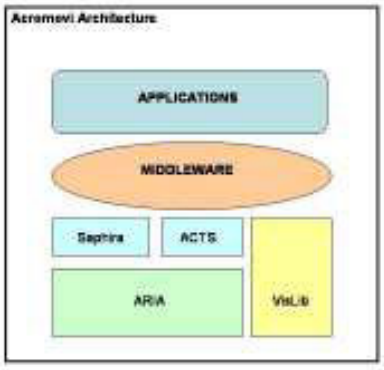


Fig. 1. General architecture diagram

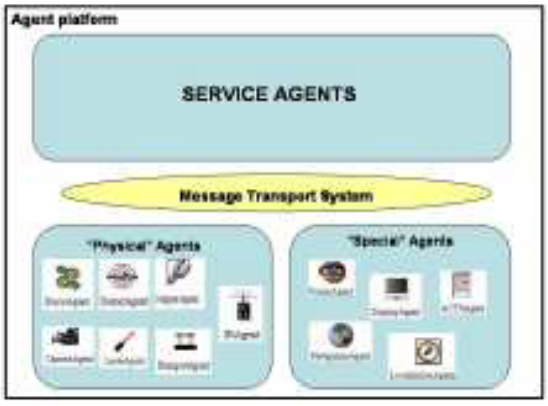


Fig. 2. The middleware layer

The structure of the middleware layer can be seen in the Fig. 2.

This middleware level has also been divided into two layers. In the lower layer, there are a set of components, such as sonar, laser, base, and so on. The others are special components that offer services to the upper layer, as vision, navigation or localization.

These components only perform two kind of functions: requests processing and results sending. Thus, they could be implemented like software components. But because of reasons of efficiency, implementation facility and integration with the whole structure, they have been developed as agents that encapsulate the required functionality

The upper layer of the middleware is still in development. This layer comprises a great variety of embedded and supervising agents, e.g. agents that monitor the state of the agents/components of the lower layer, agents that provide services of subscription to a certain component, agents that arbitrate or block the access to a component, and any other type of agent which might offer a service that can be of interest for the whole architecture or for a particular application. These agents act as links between the applications and the agents that access the components of the robot.

The presented middleware has made feasible the change from an abstraction based on library functions for the handling of the robot -that are available in compilation and not consider multiplicity of systems- to an abstraction based on embedded agents -that are available in execution and natively distributed.

Because of the heterogeneous character of the team, there are different middleware layers for the robots, depending on the configuration of each robot of the team. These middleware layers are generated in execution time according to the elements that each robot has active at the moment of its execution. These different configurations can be seen in Fig. 3.

Over the middleware layer is the applications layer, that also can be implemented as agents. These applications, to access to the components of the robots, must communicate with the agents of the middleware, which then access to the bottom layer that controls the robot.

Once an application has been tested, and if it is useful for the whole architecture, it can be converted in a new agent of the upper layer of the middleware. Thus, each application that we make can increase our system to make applications more difficult and more interesting, following a bottom-up design.

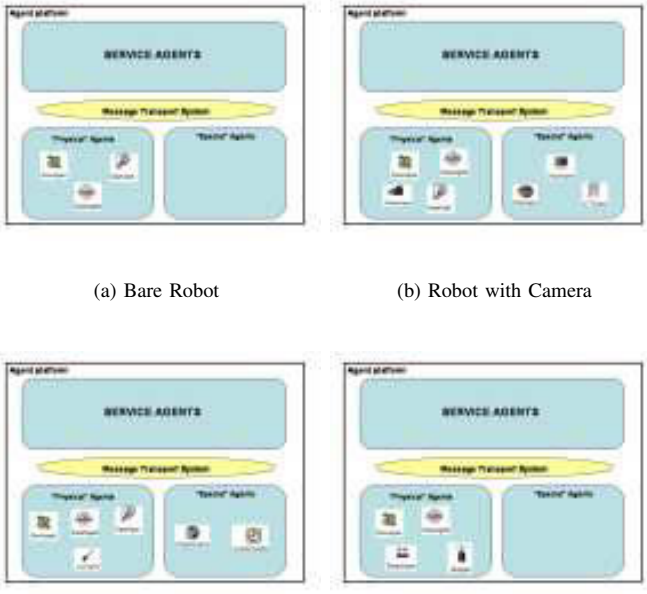
B. Architecture implementation

Due to the fact that the middleware is just a set of embedded agents, a multiagent systems programming tool has been selected for its implementation.

At beginning MadKit(Multi-Agent Development Kit) was the tool chosen for such purpose. MadKit is a Java multi-agent platform built upon an organizational model called Aalaadin. It provides general agent facilities (lifecycle management, message passing, distribution, ...), and allows high heterogeneity in agent architectures and communication languages, and various customizations. It can run in various modes: a graphical runtime environment in console-mode, embedded in an applet [8].

After a time with MadKit, we decided to change it to prove new multiagent tools. The new multiagent tool chosen was JADE (Java Agent DEvelopment Framework). It is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. Also, nowadays, JADE is considered the most used platform. This two things made that we change to JADE.

JADE is a software framework fully implemented in Java language. It simplifies the implementation of multiagent systems through a middleware and through a set of graphical



(a) Bare Robot (b) Robot with Camera

(c) Robot with Laser (d) Powerbot Base

Fig. 3. Different middleware layers.

tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which not even need to share the same OS). The JADE middleware implements an agent platform for execution and a development framework. It provides some agent facilities as lifecycle management, naming service and yellow-page service, message transport and parsing service, and a library of FIPA interaction protocols ready to be used [2].

The agent platform can be distributed on several hosts. Only one Java Virtual Machine (JVM) is executed on each host. Each JVM is basically a container of agents that provides a complete runtime environment for agent execution and allows several agents to concurrently execute on the same host.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent. The transport mechanism is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol.

JADE supports also scheduling of cooperative behaviours, where JADE schedules these tasks in a light and effective way. The runtime includes also some ready to use behaviours for the most common tasks in agent programming, such as FIPA interaction protocols, waking under a certain condition, and structuring complex tasks as aggregations of simpler ones

Having all in mind, we implemented Acromovi once again. In this case, each robot of the team is a main container, thus, we have seven main containers, one in each robot that works as host of the distributed network. In each container, we have a group of agents. These agents are created in execution time depending on the robot configuration. Each of these agents represents one of the elements which in that moment has active the robot. So, in that way, we implement the heterogeneity that we mentioned in the generation of the middleware layer of Acromovi. All this, can be seen in the Fig. 4.

Because of the fact that ARIA and Saphira are implemented in C++ and JADE in Java, it has been necessary to use a mechanism to integrate such different programming languages. For this, it has been used JNI (Java Native Interface), that allows to manage C++ code in Java programs.

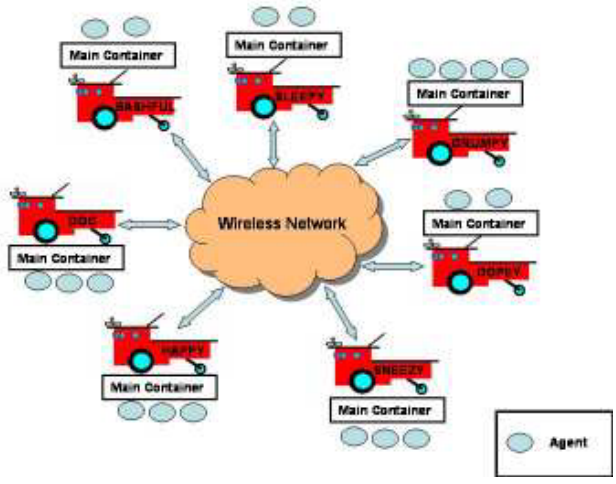


Fig. 4. Structure of the jade implementation

The robot elements are represented and managed by agents. These agents communicate with the native layer of the global architecture by means of JNI. A brief explanation of the agents is given in the next section.

IV. Acromovi agents

The agents that constitute the Acromovi architecture are described in the following paragraphs. These agents are conceptually divided in three groups depending on the part of the robot in which the agent carries out its task: body agents, laser agents and vision agents.

A. Body agents

These agents are the basic agents for the operation of the robot and they correspond to the main elements of the robots. There are five body agents: Base Agent, Gripper Agent, Sonar Agent, Bumper Agent and IR Agent.

Base Agent: agent responsible of the physical operation of the robot, all about the motion of the motors and internal state of the robot. This agent is the most important because it is the link between all the rest of the agents and the physical robot.

Gripper Agent: agent that makes that the gripper works in a proper way and permits to other agents the manipulation of the gripper. The gripper is a two degree of freedom manipulator attached to the front of the robot and contains in its paddles a grip-sensor and infrared breakbeam switches in front and rear to sense objects and their positions.

Sonar Agent: it makes the sonar work correctly and returns the appropriate sonar values when these are required by other agents or applications. The sonars consist of transducers that provide range information for collision avoidance, localization, and navigation. Measurements range from ten centimetres to over four meters.

Bumper Agent: agent that manages the bumpers of a robot. It permits to know if a robot disposes of bumpers, how many of them, and in which position. Also, it allows to recognize which bumper has been triggered.

IR Agent: agent that is in charge of the management of the 4 IR sensors that are disposed in the front of the Powerbot. These sensors has the function to prevent the robot of ledges such as stairways or holes. The agent indicates the state of each sensor, indicating whether it is triggered or not.

It is important to emphasize that bumper and IR sensors are sensors for the safe navigation of the robot, and they respond at two levels. First, the activation of one of this sensors causes the robot's controller to take a certain action, by the moment it stops the motor and sets the brakes, but this behaviour can be changed. At the next level, the controller informs about the condition that has produced the previous action.

B. Laser agents

These agents communicate with the laser rangefinder, using the ARIA interface, and permit to use two modules of Saphira:

Localization and Navigation. These two modules are powerful native libraries integrated into Saphira, using a Montecarlo-Markov method for robot's localization and a Gradient system for the navigation All their functionality is readily subsumed by the embedded agent.

Laser Agent: agent that allows multiple operations with the laser such logging its lectures, determining the distance to an object with better precision than the sonar, etc. With the tools and primitives provided by the Laser Agent, we are able to do tasks like mapping, environment scout and so on.

Localization Agent: agent that provides the robot a very good localization in a known environment, i.e. a map loaded previously in the robot. This agent also can be used in combination with the sonar sensors. This agent provides the functions needed to load a map in the robot.

Navigation Agent: with the previous two agents -Laser and Localization- the Navigation Agent calculates the path between two points. It also includes a collision-avoidance system and real-time path calculation, that allows the robot to select the best path, avoiding dynamic obstacles in the environment. This agent uses the map loaded in the Localization Agent.

C. Vision agents

Acromovi architecture is increased with the vision agents, who capture images for the robot camera. The vision agents divide themselves in four parts: Camera Agent, Acts Agent, Vision Agent and Display Agent.

Camera Agent: agent which moves physically the camera (pan, tilt and zoom) and informs about its state. Thus, although the camera is an external device, it can be integrated how any other element (sonar, laser navigation, gripper, etc). The camera is a Pan-Tilt-Zoom camera Canon VC-C4.

ACTS Agent: agent used for colour tracking of objects by means of the ACTS system. It provides information about these objects as can be area, bounding box or centroid. This agent makes use of the ARIA class that permits the communication with the ACTS server.

Vision Agent: agent that acquires images by the camera, to visualize and modify them. It makes use of the Vislib.

Display Agent: agent that modifies and to process operations on the captured image by the Vision Agent, with the Vislib library. These operations are implemented with Java AWT (Abstract Window Toolkit) y Java 2D.

V. Robot Following

The application described here is an old application that we developed with the old version of Acromovi. Now, we have implemented it with the new version to proved that this new version works as well as the old.

In this application one robot equipped with local vision pursuits another one. The leader robot navigates through an indoor environment, controlled either by a user or by its own navigation algorithms. The follower robot uses its PTZ vision system to determine the position and orientation of the leader robot, by means of a colour pattern. Such pattern, together with the local visual parameters, allows to easily compute the distance and relative orientation of the leader robot with regard to the follower-observer one.

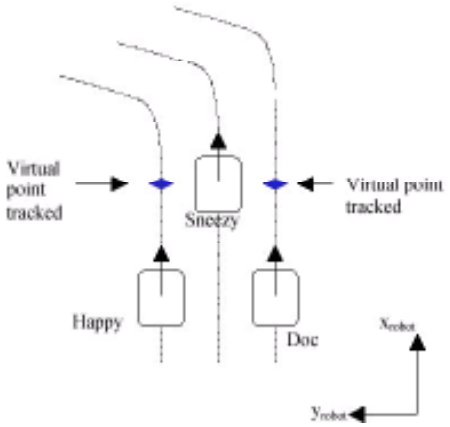


Fig. 5. Triangular formation using virtual points

The method may be extended to more robots in a line formation, provided that each robot follows its predecessor, and it is equipped with a vision system.

Also, it is possible that the robot could internally add a fixed displacement to its leader position, thus allowing robot formations other than lines to be easily deployed. As depicted in Fig. 5, each robot computes a virtual point instead of the leader position. These virtual points can be calculated simply displacing the leader position in the correct form. Now, the followers do not follow the leader, but these virtual points.

Colour tracking is provided by the ACTS server, which needs to be trained with the colours of the target. The Acts Agent works as link between the ACTS server and the Robot Following Application.

A number of agents are directly (in one or another way) involved in this application. A diagram of their interactions is depicted in Fig. 6.

The first agent implied in the process is the BlobCenter, whose role is to coordinate the ACTS, Base, and Camera Agents. Its goal is to keep the target visible. To do so, it determines the blob centroid, by means of a message received from ACTS Agent, computes the motion needed by the camera and compensates the camera and robot motions, sending messages to Base and Camera Agents.

The Position Agent calculates the robot orientation and position. When the target is centered on the image (it sends

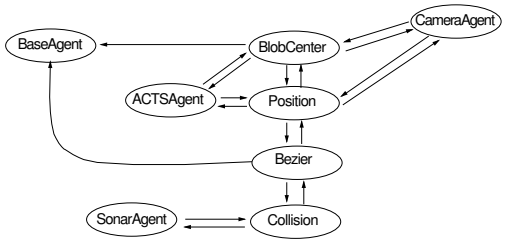


Fig. 6. Agents communication in the Robot Following Application

a message to the BlobCenter to know it), it sends messages to the Camera and ACTS Agents to determine the camera position (pan and tilt) and the different blob areas. With this information, it calculates the distance to the target and its orientation, as well as the camera orientation. Those data is forwarded to the Bézier Agent.

The Bézier agent determines a path to the target, defined by a Bézier curve. It determines two intermediate Bézier's curve control points and the initial and final points (the positions of the follower and leader robots) [6]. Motion commands are then sent to the Base Agent to perform the real motion of the robot. The system is robust enough to respond to sudden changes in the position or orientation of the leader robot, thanks to the real-time tracking system, and the small processing overhead.

Last but not least, in the background, the Collision Agent supervises the robot's path, communicating with the Bézier agent and Sonar Agent. By communicating with the Sonar Agent, it is able to detect the presence of obstacles in the robot's path. When an obstacle is getting closer the robot, the agent sends a message to the Bézier agent, which moderates the robot's speed, stopping the robot if necessary, sending the corresponding message to the Base Agent.

As a future extension, another agent could calculate alternative paths to avoid obstacles maintaining the previous curve made by the Bézier agent, by e.g. adding new intermediate control points.

VI. Conclusion

In this article, we have explained the new Acromovi architecture. In this new version, its implementation has changed from MadKit to Jade. It's important to remark that this change has been made without any problem.

Some of the conclusions of the old implementation are also true with the new implementation. Some of those conclusions are:

- Reutilization. The framework enables the use of native components, by means of agent wrappers, providing the programmer with a huge set of preprogrammed, tested and efficient tools for mobile robots.
- Integration. Once an application has been tested, it can be converted in a new agent of the upper layer of the middleware. Thus, each application can increase our system to make applications more difficult and more interesting.
- Quick development of multirobot applications. Joining the advantages of the distributed computation and multiagent systems, this architecture is very appropriate for the implementation and execution of tasks that require collaboration or coordination by part of the robots of the team.
- Sharing resources of a robot among all the team and easy access to the elements of the robot by the applications.
- Time needed by users to develop an application. In just a few weeks, a student with very low Java skills, is able to program agents and develop the necessary testing procedures.

Future extensions of this work include the study of efficiency, particularly in image applications and communications.

Acknowledgements

This paper describes research carried out at the Robotic Intelligence Laboratory of Universitat Jaume-I. Support for this research is provided in part by the Ministerio de Ciencia y Tecnologia under projects DPI2001-3801, HF2001-0112, and FIT-020100-2003-592 (PROFIT), by the Generalitat Valenciana under projects inf01-27, GV01-244, CTIDIA/2002/195, and by the Fundacio Caixa-Castello under project P1-1B2001-28. The authors gratefully acknowledge this support.

References

[1] T. Arai, E. Pagello, and L. E. Parker, "Guest editorial: Advances in multirobot systems," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 655–661, 2002.

[2] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "Jade a white paper," *EXP in search of innovation (Special Issue on JADE)*, vol. 3, no. 3, pp. 6–19, 2003.

[3] D. Cáceres, H. Martínez, M. Zamora, and L. Balibrea, "A real-time framework for robotics software," in *Int. Conf. on Computer Integrated Manufacturing (CIM-03)*, 2003.

[4] P. Caloud and et al., "Indoor automation with many mobile robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems(IROS)*, 1990.

[5] Y. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, pp. 1–23, 1997.

[6] S. Y. Chiem. and E. Cervera, "Vision-based robot formations with Bézier trajectories," in *Eighth Conference on Intelligent Autonomous System*, 2004.

[7] S. Enderle, H. Utz, S. Sablatng, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for autonomous mobile robots," in *IFAC Conference on Telematics Applications in Automation and Robotics*, 2001.

[8] J. Ferber and O. Gutknecht, "A meta-model for the analisys and design of organizations in multi-agent systems," in *Third International Conference on Multi-agent Systems(ICMAS' 98) Proceedings*, 1998.

[9] T. Fukuda and G. Iritani, "Construction mechanism of group behavior with cooperation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems(IROS)*, 1995.

[10] F. Jiménez, H. Martínez, and R. Rizo, "An agent-based distributed platform for learning control and tele-operation of mobile robots by internet. (tic2001-0245-c02)," in *Jornadas de Seguimiento de Proyectos en Tecnologías Informáticas, CAEPIA-TTIA*, 2003.

[11] J. Johnson and M. Sugisaka, "Complexity science for the design of swarm robot control systems," in *26th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2000.

[12] D. Jung and A. Zelinsky, "An architecture for distributed cooperative planning in a behaviour-based multi-robot system," *Journal of Robots and Autonomous Systems*, vol. 26, no. 2–3, pp. 149–174, 1999.

[13] K. Konolige and K. Myers, *The Saphira Architecture for Autonomous Mobile Robots*. MIT Press, 1998, ch. 9, pp. 211–242.

[14] M. J. Mataric, "New directions: Robotics: Coordination and learning in multirobot systems," *IEEE Intelligent Systems*, vol. 13, no. 2, pp. 6–8, 1998.

[15] P. Nebot, D. Gomez, and E. Cervera, "Agents for cooperative heterogeneous mobile robotics: a case study," in *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, 2003.

[16] L. Parker, "Distributed autonomous robotic systems 4," in *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS2000)*, 2000, pp. 3–12.

[17] S. Picault and A. Drogoul, "The microbes project, an experimental approach towards open collective robotics," in *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS'2000)*, 2000.