Robotica (2007) volume 25, pp. 213–220. © 2007 Cambridge University Press
doi:10.1017/S0263574706003316 Printed in the United Kingdom

An integrated agent-based software architecture for mobile and manipulator systems

Patricio Nebot* and Enric Cervera

Intelligence Robotics Laboratory, Jaume-I University, E-12071 Castelló, Spain
E-mail: pnebot@icc.uji.es, ecervera@icc.uji.es

(Received in Final Form: December 22, 2006)

SUMMARY

Mobile manipulation involves the most important key issue in robotics: integration. While hardware integration seems to be nearly solved due to the increasing dominance of PC-compatible systems, software integration is still a challenge, since a lot of issues arise with the variety of operating systems, device drivers, application libraries, and programming languages which need to be merged in any real-world robotic system. This paper presents a software architecture, which seamlessly integrates robot arms, mobile bases, vision systems and sensing devices, in a distributed, homogeneous agent framework. Based on the Java platform, the agent-based architecture allows great flexibility in the integration of components, and provides a simple yet extensible and powerful software layer to develop further mobile manipulating environments. Detailed software issues, as well as preliminary results are shown, which pave the way towards the development of network-ready applications involving mobile and manipulating artifacts.

KEYWORDS: Control architecture; agents; distributed systems; integration software.

1. Introduction

Since the last few years, there has been an emergent interest in combining the two most important fields of robotics—the control of static manipulators and the navigation of mobile robots. By means of the mobility of the platform, there is an increment of the workspace of the manipulator. Also, having a manipulator on a mobile robot increases its operational capability and flexibility. Such systems allow the most usual missions of robotic systems, which require both locomotion and manipulation abilities. So, they seem particularly suited for field and service robotics.[2]

The presented paper describes an extension added to the Acromovi architecture to manage an arm over a mobile robot. Acromovi architecture is a framework for application development by means of agents. Also, Acromovi is a distributed architecture that works as a middleware for other global architectures for programming robots.[13]

Acromovi has some peculiarities. It allows the reutilization of code, due to its use of native components, by means of agent wrappers. Also, it allows the sharing of the robots' resources and an easy access to the robots' elements by the applications. Other two important characteristics of the Acromovi architecture are the scalability and the facility of use.

For the design of the framework, the characteristics of the two systems that constitute the mobile manipulator have been considered: the mobile base and the manipulator. The framework maintains the characteristics of the two sub-systems independently; while at the same time provides a basis for cooperation by means of the multiagent system. Also, the framework lets the access to the accessories mounted on each sub-system making it possible to coordinate all the elements to perform a certain task.

In Section 2, some works related to control architectures for mobile manipulation are described. Section 3 describes the mobile manipulator. In the following sections, an overview of the Acromovi architecture and the extension are showed. Finally, some tools and applications implemented with such architecture are explained. At last, the future lines and conclusions are described.

2. State of the Art

Many different control systems for mobile manipulators have been developed in the last few years. In ref. [9], a control architecture suitable for mobile manipulation is explained. This architecture combines existing techniques for navigation and mobility with a flexible control system for the manipulator arm.

In ref. [3], the authors implement a behaviour-based controller over a mobile manipulator to make feasible that the robot opens a door. The key issue for realizing such behaviour is cooperation between robots function sub-systems, such as the locomotion control system, the manipulator control system and the sensor systems.

Other approaches to the control of a mobile manipulator take into account neural networks. In ref. [17], a neural network–based methodology is developed for the motion control of mobile manipulators subject to kinematics' constraints.

A task-oriented framework for the dynamic coordination of mobile manipulator systems has been implemented in ref. [12]. The dynamic coordination strategy developed is based on two models concerned with the effector dynamic behaviour, and the robot self-posture description and control. The effector dynamic behaviour model is obtained by a projection of the robot dynamics into the space associated with the effector task, while the posture behaviour is characterised by the complement of this projection.

*Corresponding author. E-mail: pnebot@icc.uji.es

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press

Finally, other possible approach is to use multiagent system as in ref. [10]. This work presents a multiagent system approach to a service mobile manipulator robot that interacts with human during an object delivery and hand-over task in two dimensions. The agents of the system are controlled using fuzzy control. The functions of the fuzzy controllers are tuned by using genetic algorithms.

The agent architecture development environment (ADE)[19] combines and integrates multiagent systems frameworks and agent architecture toolkits for single virtual and robotic agents. It provides a homogeneous, user-friendly environment for the development of architectures for virtual and robotic agents in single and multiagent settings. The functionality of the ADE agent architecture toolkit is built over heterogeneous computational units called "components". An integrated server–client subsystem allows components of the architecture to connect directly to robots or remote agents in a simulated environment in order to control them.

In ref. [7], a multiagent system architecture for a modular mobility enhancement system is presented. The system consists of one or multiple mobile robotic platforms and a set of user-defined application modules, such as chairs, tables, multifunctional chairs etc. All these modules can be inexpensive everyday life items, which become functional when a mobile platform is properly attached to them. This way the system can act as a wheelchair, as a carrier of objects, or even as a walker.

Concerning the vision in ref. [1], a new methodology to achieve vision-based control of a mobile manipulator is presented. This methodology makes it possible that the mobile manipulator reaches the required three-dimensional target position and orientation. The objective is to apply the non-holonomic degrees of freedom represented by two independently driven wheels together with the holonomic dexterity of the onboard arm in order to bring about a desired positioning objective for the arm.

Other work that presents a visual control is ref. [11]. In this project, a high-precision visual control method for mobile manipulators is presented. That method makes efficient use of all the system's degrees of freedom, which reduces the required number of actuators for the manipulator.

Finally, the arm must be able to perform manipulation tasks. In ref. [5], a framework for visually guided manipulation tasks is presented. The complexity of such task determines the precision and freedoms controlled, which also affects the robustness and flexibility of the system. The proposed approach results in a system, which can locate and grasp a certain object.

The previous works implement architectures and techniques so that mobile manipulators can develop their tasks. The work described in this paper consists of the implementation of architecture of such type and some of the techniques shown, but adding the versatility and power of the multiagent systems.

Multiagent systems are an important tool for the development of new robotic applications. They are the natural environment for the development of applications, which consist of more than one robot. And they also make possible the fast implementation of powerful architectures for the specification and execution of those cooperative tasks.

As a mobile manipulator is composed of two robotic systems, the mobile base and the manipulator arm, a multiagent system is an ideal selection to model these two systems and to allow the cooperation between them. Also, it is important to try to have any of these systems independently of the other, in order to facilitate their operation and deputation, but allowing the cooperation between them in the applications level in order to have the whole system working as a mobile manipulator.

3. *Acromovi* Architecture Overview

Acromovi architecture is a distributed architecture for programming and controlling a team of multiple heterogeneous mobile robots that are capable of cooperating among themselves and with people to achieve tasks of service in daily environments.[13]

3.1. Architecture design

The mobile robot already has a programming architecture with native (C/C++) libraries, constituted by two layers. The lower layer, ARIA, takes care of the requests of the programs to the robot elements. The upper layer, Saphira, provides services for range sensor interpretation, map building and navigation.

However, this native code is oriented to local remote processing but only from one controller, without defining collaboration between robots or applications. Acromovi architecture tries to solve this problem by the introduction of a new level over the native code allowing the cooperation and collaboration. Also, the Acromovi architecture is able to subsume any other extra software, like the ACTS (a colour tracking library) and the Vislib (a frame grabbing library), as can be seen in Fig. 1.

So, Acromovi architecture works as a middleware layer between the robot architecture and the applications. This middleware has been implemented following an agent-oriented approach. 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 to handle these elements and provide higher level services.

This middleware level has also been divided into two layers, as can be seen in Fig. 2. In the lower layer, there

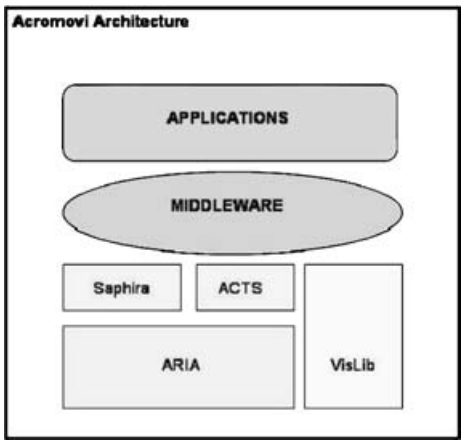


Fig. 1. Acromovi architecture diagram.

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press

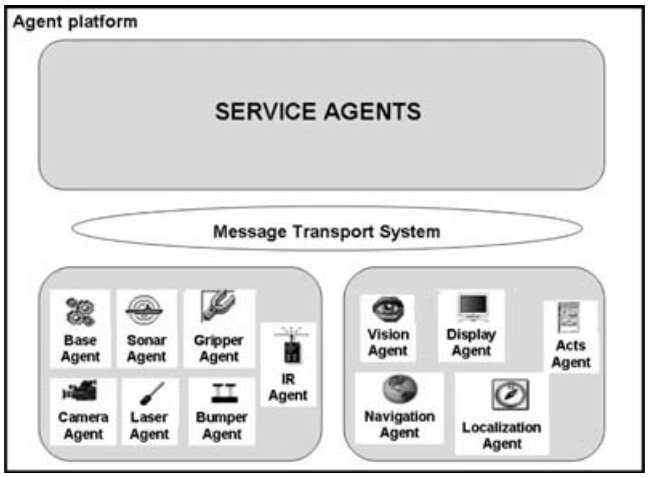


Fig. 2. Middleware layer of the Acromovi architecture.

are a set of components that access the physical elements of the robots, as the sonar, the base or the gripper and other special components that offer services to the upper layer, as the vision, the navigation or the localization. It is important to note that these components have been implemented as agents due to factors of efficiency, implementation facility and integration with the whole structure.

The upper layer of the middleware comprises a great variety of embedded and supervising agents, such as 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 can access the physical elements 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, not considering multiplicity of systems—to an abstraction based on embedded agents—that are available in execution and are natively distributed. One major design guideline was the distribution requirements and the communication between various platforms.

Because of the heterogeneous character of the robots, there are different middleware layers depending on the configuration of each robot. These middleware layers are generated in execution time according to the active elements that each robot has at the moment of its execution. Therefore, each middleware layer will activate those agents that permit the access to the own components of the robot. Some examples of this can be seen in Fig. 3.

Over the middleware layer is the applications layer. These applications, to access the elements of the robots, must communicate with the agents of the middleware, which then access the bottom layer that controls the robot.

Once an application has been tested and found useful for the whole architecture, 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, following a bottom-up design.

3.2. Architecture implementation

Because the middleware is just a set of embedded agents, a multiagent systems programming tool has been selected

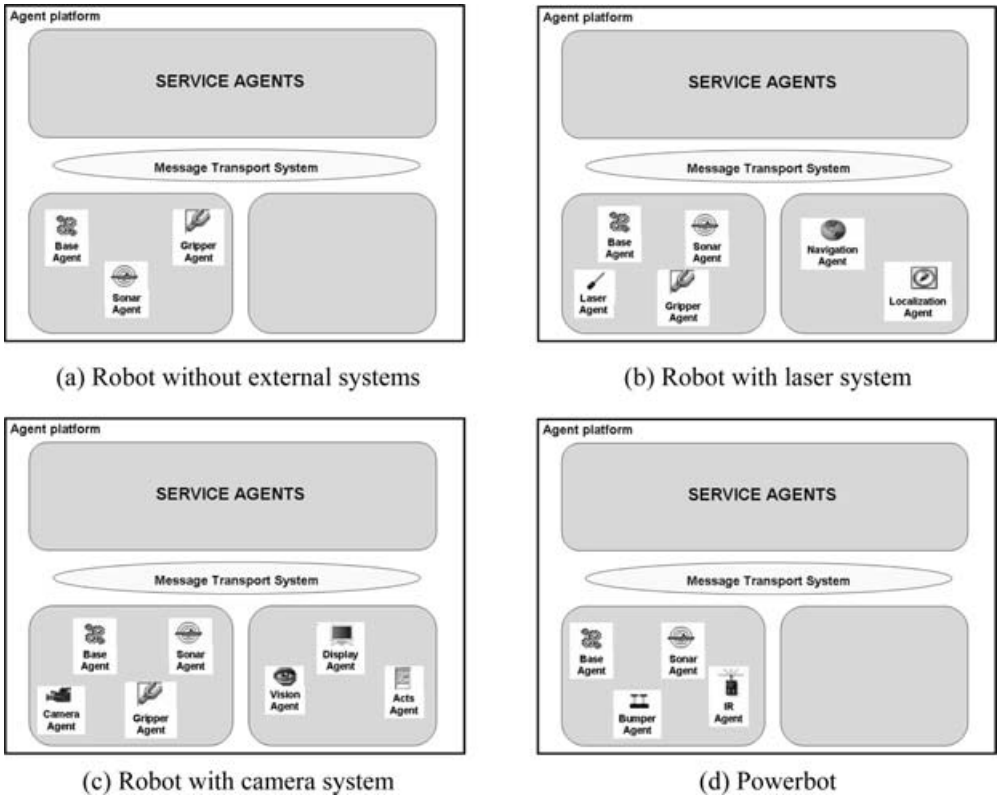


Fig. 3. Differences between middleware layers.

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press



Fig. 4. Structure of the implementation of the Acromovi architecture.

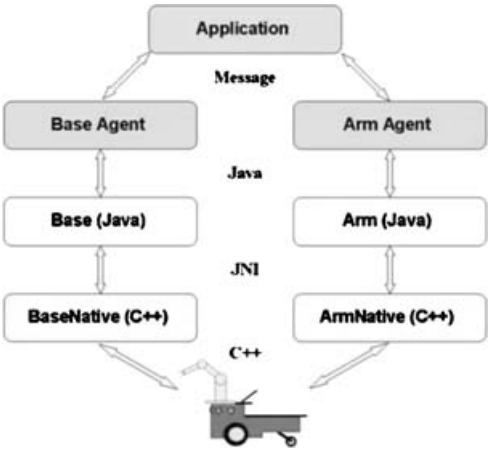


Fig. 5. Different communication languages implemented in the Acromovi architecture.

for its implementation. The chosen multiagent tool was Java Agent DEvelopment Framework (JADE). It is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multiagent systems. It is a software framework that is fully implemented in Java language and simplifies the implementation of multiagent systems through a middleware and through a set of graphical tools that supports the debugging and deployment phases. The JADE middleware implements an agent platform for execution and a development framework.[6]

The Acromovi architecture has been implemented following the JADE specifications. Thus, each robot involved in the architecture is a main container. Each one works as host of a distributed network. In each container, there is a set of agents. These agents are created in execution time depending on the robot configuration, and each one represents one of the elements that has activated the robot at that moment. All this can be seen in 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 reason, Java Native Interface (JNI) has been used, thus allowing managing C++ code in Java programs. In this way, the robot elements are represented and managed by agents and these agents communicate with the native layer of the global architecture by means of JNI.[16] In Fig. 5, an example of communication for one agent is shown, from the applications layer to the call to the function that moves the element, which in this case is the base.

In this example, a generic application would try to move the base by one metre. For that, first it must send a message to the BaseAgent indicating the required movement (*sendMessage*("*move 1.0*")). When BaseAgent receives that message, it calls a Java function from the Base library that makes the robot move (*b.move*(*1.0*)). This function has been implemented in the C++ BaseNative libray, and to make possible the communication between the Java function and the C++ function, it is necessary to use the JNI. The C++ function that makes the robot move simply uses another function defined in the ARIA library that moves the robot physically to the distance desired.

3.3. Architecture performance

Finally, in order to determine how good the architecture is, a time performance is shown. In this performance, the robot must realise the "test of the square", that is, its movement must draw a square of 1 m × 1 m. In the test, the square is repeated five times.

Three different ways of communication have been tested. The first one consists of implementing the architecture using software components. These components communicate among them by means of the Remote Method Invocation (RMI) method. The second way is the implementation of the architecture using the MadKit multiagent system. And the third one is the implementation of the architecture described in this paper, using the Jade system.

The method for getting the times is the following; there is a client, a server and a controller for the base. The client communicates with the server to indicate which movement it would do. When the server receives the order from the client, it communicates with the controller to move the robot physically. The communications, in the case of RMI, are made by means of remote function invocation, and in the case of the MadKit and the actual architecture, by means of messages.

Also, different configurations are related to the place where the interactive entities are have been tested. Thus, in the cases of MadKit and actual implementations, the intra-platform and the inter-platform configurations have been tested. In the case of RMI, only the intra-platform configuration has been tested. In the intra-platform configuration, all the entities are in the same robot or computer. In the inter-platform configuration, there is a robot or computer with the client, and other one with the server and the controller. The information that is of interest for the performance is the transmission time between the client and the server. A graphic comparing these times can be seen in Fig. 6. It is important to note that the times shown in the graphic are the times of sending the message from the client to the server plus the time of the answer from the server to the client.

Also, as can be seen in the figure, in the intra-platform configuration all the systems have similar time. That is because of the fact that Jade and MadKit internally use the RMI method when the messages are sent in the same platform.

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press

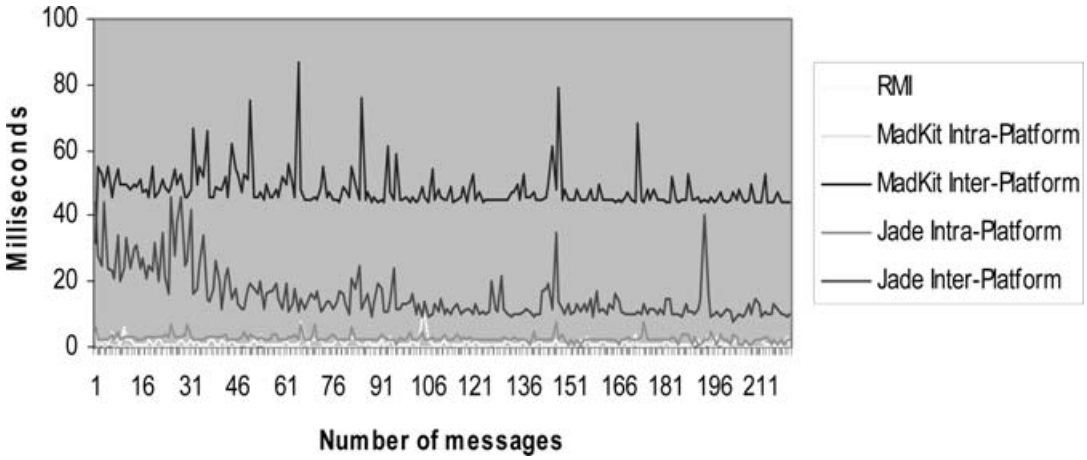


Fig. 6. Timing performance among different implementations of the Acromovi architecture.



Fig. 7. Mobile manipulator: PA10 over Powerbot.

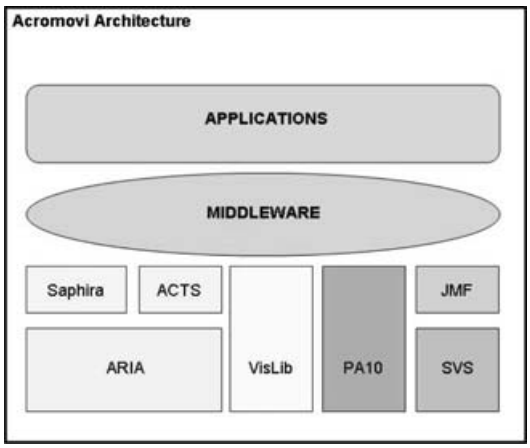


Fig. 8. Extended Acromovi architecture.

The differences come from the inter-platform configuration, in this case, as can be seen, the actual configuration is better than the one implemented with MadKit.

4. Acromovi Extension

The purpose of this work is to add to the architecture previously described, the capability to manage new components, an arm with all its accessories, and make feasible the cooperation between the arm and the mobile base.

The mobile manipulator used in this work is formed by the combination of a Mitsubishi PA10 arm and an ActivMedia Powerbot mobile robot, as seen in Fig. 7.

The arm has seven degrees of freedom and is equipped with a two-finger gripper, a force sensor, some infrared sensors for collision detection and a stereo pair mounted at the end-effector, which consists of two digital IEEE1394 colour cameras.

The mobile robot presents IR sensors for stairs and holes detection in the front, and around it there are bumpers and sonars. Also, it has an onboard computer and a wireless card, which lets it communicate with other robots or Internet.

Moreover, as can be seen in the photograph, the mobile manipulator carries backwards the controller of the arm and batteries rack to provide the necessary power to the arm and to make it autonomous.

As in the case of the Vislib or ACTS, it is necessary to subsume new native libraries in the lower level of the global architecture. These native libraries are the libraries that permit access to the arm and its accessories. This can be seen in Fig. 8.

The PA10 library is a library to develop application programs for the operation control of the robot, but moreover, is an interface library to ease the operation of all the functions for the motion control of the robot. It is written in C language.

To manage the stereo, we use the C++ library–Small Vision System (SVS).[8] The SVS consists of a set of library functions for performing stereo correlation. Programs may call these functions to compute stereo results on any image that is available. It is written in C++.

These two libraries need to transfer their functions and information that returns to Java. For this reason, the Java Native Interface (JNI) is used. In the case of the PA10 library, there is a wrapper that transforms the functions of this library to Java functions. For the vision system, there is a wrapper for the SVS, and it is able to deliver stereo 320 × 240 colour images at 20 frames per second.

The stereo provides information that can be used to control the arm with a Java program. The acquisition of the images need the libraries provided with the cameras. Then these

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press

images are used to create the video stream using the Java Media Framework (JMF).[15]

One major advantage of using JMF is that it is able to seamlessly transmit video over the network, using the RTP protocol.[4] With JPEG compression (setting the quality factor to 0.5), the system is able to deliver 20 fps of stereo images through a wireless network (802.11b) to the university network, without noticeable loss of quality and response.

Image processing is done entirely in Java, keeping pace with the incoming frame rate. Several processing techniques are available:

- Colour blob segmentation: up to 32 channels, with multiple blobs on each channel, are available. The centroid, size and bounding box of each blob is computed
- Edge and corner detection based on the SUSAN principle[18]
- Line detection based on the Hough transform[14]
- Motion detection thorough a simple luminance change detector
- Regions of interest are used thoroughly to avoid processing of the complete image

Other techniques are currently being implemented, or planned for the near future, are stereo disparity computation, face recognition, people tracking and multisensor fusion techniques.

To control these new libraries, it is necessary to include new agents/components to the middleware layer, in more detail, in the lower level, as can be seen in Fig. 9. These new components are in charge of serving the petitions of the other agents of the system to the element that they manage.

Thus, the first agent developed is the agent that moves the arm. This agent makes possible the communication between the upper agents and the native library of the arm to move it.

In order to integrate the images of the stereo as per the agent architecture, a new agent has been implemented, the SVSAgent. This agent serves the images that are acquired by the stereo to the system and offers all the facilities that present the SVS library.

Finally, the JMF agent is capable to create a video stream through the images that serves the SVSAgent. This video frame can be transmitted over the network without an elevated cost due to the jpeg compression. Also, this agent is able to execute some functions of image processing.

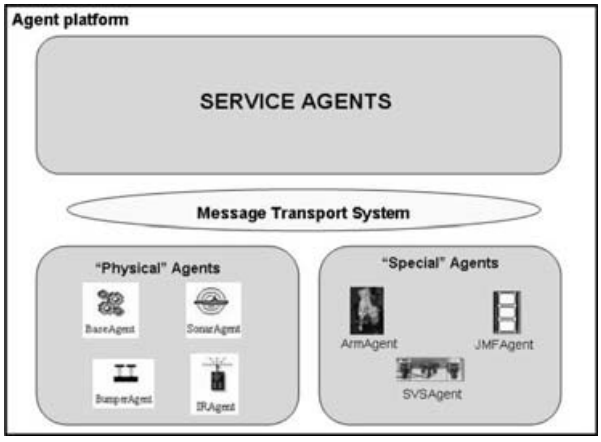


Fig. 9. Mobile manipulator middleware.

As can be deduced, each of the systems that form the mobile manipulator, the mobile base and the manipulator arm, are considered independently, but at the same time allowing their cooperation on the applications layer of the architecture to solve possible problems of global redundancy or control sharing.

It is the responsibility of the programmer to provide adequate tools for the architecture to make possible the mobile manipulation of the system. Also, it is possible, because of the scalability of the architecture, to implement a new application that makes possible the manipulation of the whole mobile manipulator, and then to implement that application as a new agent of the upper layer, allowing the agents of the architecture and other applications to access the facilities of the mobile manipulator.

5. Experimental Applications

In the following subsections, two applications implemented over the extension of the architecture are described. The first, the arm controller, lets us move the arm in the two spaces of coordinates, joint and Cartesian. The second, the visual servoing, permits on one hand to move the arm to situate it in a certain distance and position of an object, and on the other hand permits to detect coloured objects to control the orientation of the arm.

5.1. Arm controller

This application controls the arm through the network by means of agents. This control can be made by means of messages or by means of a user interface. The Control agent is in charge of instantiating the user interface, which is composed of two tabs setting Cartesian in tip or base coordinate and joint motions. This user interface is completely implemented with Java Swing components.

Figure 10 represents the interaction between different parts that make possible the communication between a user and the arm.

When the user interacts with the interface, it generates events that are captured by the ControlAgent. This agent, depending on the events, sends different orders through the

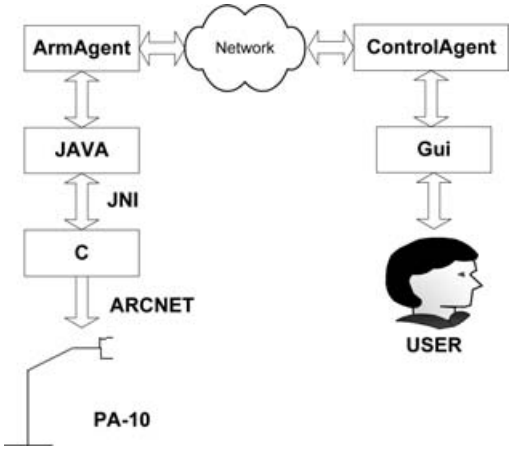


Fig. 10. Communication between the agents in the GUI arm control application.

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press



Fig. 11. Joint control tab of the PA10 arm.

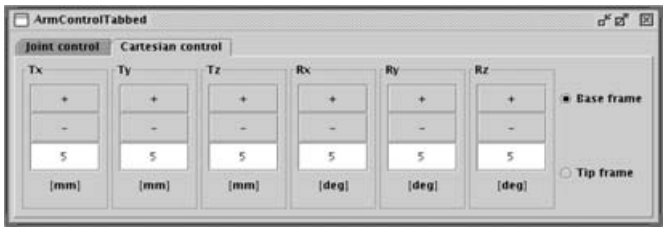


Fig. 12. Cartesian control tab of the PA10 arm.

network to the ArmAgent. Then, the ArmAgent invokes the requested Java function implemented from the native C function provided by the PA10 library via the JNI technology. This function is sent to the arm via ARCnet Attached Resource Computer Network (ARCnet), which is one of the oldest, simplest and least expensive types of local-area network.

The PA10 library communicates with the embedded computer that manages the arm. Communication between the embedded computer and PA10 controller enables either to launch an arm motion or to acquire useful arm information that following the inverse way is showed in the user interface.

Also, the rest of the agents in the system can communicate directly with the ArmAgent in order to ask it to move the arm to get something of interest for the certain application. This communication is made by message passing. When the ArmAgent receives a message, it invokes the appropriate Java function implemented from the native PA10 library.

In the following figures, the two tabs that manage the application can be seen. In Fig. 11, the joint control tab is shown. By means of this tab, the user can move the arm indicating the relative movement of each joint.

Using the tab in Fig. 12, a user is able to move the arm in Cartesian coordinates. There are two possible origins of coordinates for the arm movements, the base and the tip. Also, it is possible to specify two different movements of the arm, translational movements or rotational movements.

5.2. Visual servoing

The purpose is move the arm to a certain position respective to an object. The position is which the object is centered in the image and the arm is a certain distance above the object. To monitor the process, two images (left and right) taken from the stereo are displayed in the window.

Figure 13 shows the initial and final positions of the process. It is possible to appreciate as in the final position, the object is centered in the image and closer than in the initial position.

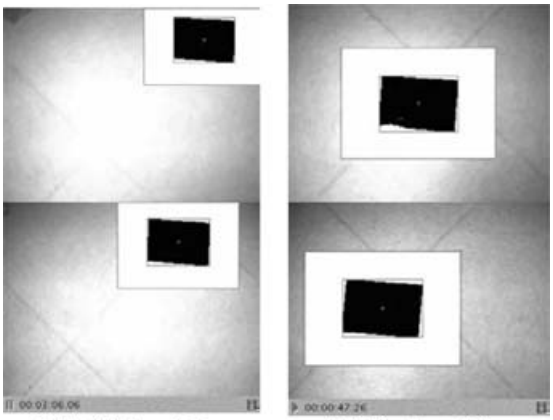


Fig. 13. Visual servoing processing.

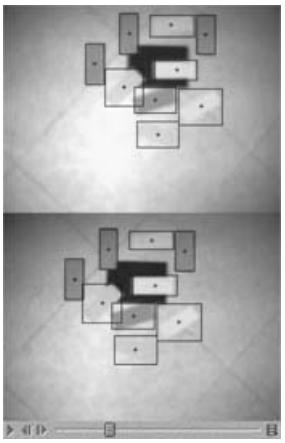


Fig. 14. Objects colour tracking.

The program looks for coloured objects in a clear background. Then it computes the coordinates of their centres of gravity. The distance between the object and the camera is computed with the disparity between the two images.

With these three pieces of information (coordinates and distance of the object) the arm is controlled to situate it to the needed position. The application communicates with the ArmAgent to move the arm with the three translational movements that this agent offers in the Cartesian space.

In order to control the seven degrees of freedom of the arm, many colour objects can be detected, as in Fig. 14. These objects bring the information necessary to control the orientation of the arm that can be modified with the ArmAgent by means of the rotational movement in the Cartesian space.

6. Conclusions and Future Extensions

New agents have been implemented based on the Acromovi architecture to make feasible the integration of a manipulator arm over a mobile base to create a mobile manipulator. These agents facilitate the use of the arm and all its accessories. Thanks to the Acromovi architecture, it is possible to establish the necessary cooperation between the arm and the mobile base in order to accomplish tasks for mobile manipulators.

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press

By means of a GUI, it is possible to control the arm and to know how it works before using it in autonomous programs. Also, the operation of the stereo cameras has been tested by means of an application of visual servoing. These applications have demonstrated that the agents generated to integrate the arm in the Acromovi architecture are adequate.

It has been possible to see how the agents cooperate in the visual servoing application. In this application, the arm agent moves the arm depending on the visual information obtained from the agent that controls the camera. Following this point of view, it is the responsibility of the programmer to establish the correct cooperation and coordination between the arm and the mobile base in order to accomplish the tasks for mobile manipulators. Acromovi architecture simply offers the necessary tools to accomplish this objective in an easy way.

Regarding future work, a first extension of the system is to give it the capability to manage the force sensor of the PA10 arm. This sensor permits to know how much force is applied to the arm at a certain moment. This force is produced by the strength by which the gripper is pushing any object.

Other extension, concerning the physical elements that the robot offers, is to create a correspondent agent to manage the infrared sensors that wrap the arm in order to avoid collisions. More complicated extensions could be creating agents to calculate the path that must be followed by the arm to catch an object localised by the stereo.

The most important application for our mobile manipulator is the librarian robot. Due to the arm installed on the mobile robot, this is capable of manipulating books. This robot will be in charge of a library to look for and to extract from the book shelves the requested books, and will be able to carry them to the users.

Other possible applications can be a robot painting a wall or a robot navigating by its environment and picking up all the cans that it detects.

Acknowledgments

This work has been partially funded by the EU-VI Framework Programme under grant IST-045269 in the EC Cognitive Systems initiative, and by the Spanish Ministry of Science and Education (MEC) under grant DPI2005-08203-C02-01.

References

1. A. Cardenas B. Goodwine, S. Skaar and M. Seelinger, "Vision-based control of a mobile base and on-board arm," *Int. J. Robot. Res.* **22**(7), 677–698 (2003).
2. B. Bayle J.Y. Fourquet and M. Renaud, "Manipulability of wheeled mobile manipulators: application to motion generation," *Int. J. Robot. Res.* **22**(7), 565–581 (2003).
3. B. J. W. Waarsing, M. Nuttin and H. Van Brussel, "Behaviour-Based Mobile Manipulation: The Opening of a Door," *Proceedings of ASER'03* (2003) pp. 168–175.
4. C. Perkins, *RTP: Audio and Video for the Internet* (Addison-Wesley, Boston, MA, (2003).
5. D. Kragic L. Petersson and H. I. Christensen, "Visually guided manipulation tasks," *Robot. Autonom. Syst.* **40**(2–3), 193–203 (2002).
6. F. Bellifemine, G. Caire, A. Poggi and G. Rimassa. "JADE a white paper," *EXP in search of innovation* (special issue on JADE) **3**(3), 6–19 (2003).
7. G. Lidoris and M. Buss, "A Multi-Agent System Architecture for Modular Robotic Mobility Aids," *Proceedings of the European Robotics Symposium* (2006) **22**, pp. 15–26.
8. K. Konolige, "The SRI small vision system," http://www.ai.sri.com/~konolige/svs/
9. L. Petersson, M. Egerstedt and H. I. Christensen, "A Hybrid Control Architecture for Mobile Manipulation," *Proceedings of the IROS'99* (1999) pp. 1285–1291.
10. M. S. Erden, K. Leblebicioglu and U. Halici, "Multi-agent system-based fuzzy controller design with genetic tuning for a mobile manipulator robot in the hand over task," *J. Intell. Robot. Syst.* **39**(3), 287–306 (2004).
11. M. Seelinger, J. D. Yoder, E.T. Baumgartner and S.B. Skaar, "High-precision visual control of mobile manipulators," *IEEE Trans. Robot. Autom.* **18**(6), 957–965 (2002).
12. O. Khatib, "Mobile manipulation: the robotic assistant," *Robot. Autonom. Syst.* **26**(2–3), 175–183 (1999).
13. P. Nebot and E. Cervera, "Agent-based Application Framework for Multiple Mobile Robots Cooperation," *Proceedings of the ICRA* (2005) pp. 1509–1514.
14. P. V. C. Hough, "Methods and means for recognizing complex patterns," U.S. Patent 3, 069, 654 (1962).
15. R. Gordon and S. Talley, *Essential JMF: Java Media Framework* (Prentice Hall, Boston, MA, 1998).
16. S. Liang, *Java(TM) Native Interface: Programmer's Guide and Specification* (Addison-Wesley, Boston, MA, 1999).
17. S. Lin and A. A. Goldenberg, "Neural-network control of mobile manipulators," *IEEE Trans. Neural Netw.* **12**(5), 1121–1133 (2001).
18. S. M. Smith and J. M. Brady. "SUSAN—a new approach to low level image processing," *Int. J. Comput. Vis.* **23**(1), 45–78 (1997).
19. V. Andronache and M. Scheutz, "Ade – an architecture development environment for virtual and robotic agents," *Int. J. Artif. Intell. Tools* **15**(2), 251–286 (2006).

https://doi.org/10.1017/S0263574706003316 Published online by Cambridge University Press