A CORBA Wrapper for Applications with Multiple Robots

http://dx.doi.org/10.3991/ijoe.v7i4.1724

I. Calvo, I. Cabanes, I. Etxeberria-Agiriano, G. Sánchez and E. Zulueta
University of the Basque Country (UPV/EHU), Spain

***Abstract*—This paper presents a CORBA wrapper which encapsulates a generic anthropomorphic industrial robot. Since this wrapper abstracts the communications, building applications that require remote manipulation or coordination of several devices may be easily achieved by using it. This article describes an implementation of this wrapper over a real-time operating system (RTOS), namely RTAI. This type of OS ensures determinism in the movement operations of the robot. Also, a low resource consuming implementation of the CORBA specification, namely ORBit, has been used to wrap the robot and implement the communications with other devices. Finally, as a matter of example, we present how this wrapper is used to coordinate the operation of several robots in a typical 'pick & place' operation.**

***Index Terms*—CORBA, middleware, Real time systems, Robot applications.**

I. INTRODUCTION

Even though in the latest years there have been important advances in the integration and coordination of industrial control devices, quite often the creation of high scale applications is still a challenging task. This is mainly due to the complexity of connecting several heterogeneous devices provided by different vendors. Several trends try to minimize these difficulties.

Thus, there is an increasing trend in industrial environments to use open middleware technologies since they may be seen as a virtual bus that eases the connection of several software components. Some of the most widely used technologies, such as CORBA, can be executed over different heterogeneous platforms and allow tackling vendor-dependent issues to make communications transparent to the developers.

Also, a growing number of vendors are adopting truly Real Time Operating Systems (RTOS) like QNX or VxWorks, or modified versions of general purpose operating systems like Linux in order to be embedded in their devices. This second alternative is particularly interesting since it allows profiting some existing inexpensive tools while being able to react to device events in a timeliness manner.

This work describes how the combination of both types of technologies eases the development of complex robotic applications composed by several control devices. For this purpose, a CORBA wrapper that encapsulates a generic anthropomorphic industrial robot is presented. This wrapper has been implemented using ORBit [5], which is a very low profile implementation of the CORBA middleware specification [2]. This wrapping allows device hiding and segregating decisions likely to change, thus providing a stable interface.

From the controller point of view, handling the hardware of the robot (i.e. mechanical parts such as arms, grippers, joints or motors) is a time-critical issue. It is therefore important to have an operating system able to guarantee determinism in the robot operations. We have chosen RTAI [11] at the server, which is a real time extension for the Linux kernel developed by the community that lets writing applications with strict timing constraints.

This paper also describes an implementation of the proposed wrapper that has been carried out over a Lynx 5 robotic arm with 4 rotation axes and a gripper. In this case, for the sake of simplicity, the position of the robot has been calculated from the forward kinematics, since it has no position sensors. However, in real industrial robots its real position should be obtained from the readings of the encoders.

The results obtained for a single robot can be extrapolated to other types of robots or different devices like PLCs in a flexible manner.

Related work can be found in the work by Azad et al. [1]. They model and simulate with Simulink a single-link flexible robotic manipulator. Floroian et al. [12] develop a multi-agent mini-robotic system. García et al. [8] utilize micro-servers to access and configure remote robots for the pharmaceutical sector.

The remaining of the paper is structured as follows. Section 2 describes the system architecture with Linux/RTAI and the CORBA communications; in Section 3 we present the generic interface of the wrapper operations; Section 4 is dedicated to the implementation details and in particular to calculate the forward and inverse kinematics of the robot; Section 5 studies a pick and place example of use; the final section draws the conclusions of this work.

II. ARCHITECTURE DESCRIPTION

The basic architecture of our implementation is shown in Figure 1.

This simple architecture can be easily generalized to another architecture in which several robots are controlled by one single client from the Internet as depicted in Figure 2. In the Figure, each robot has been associated with a server which is the robot controller since in the most generic case robots require considerable computing power for the calculation of the inverse kinematics problem which must be solved with real-time constrains.

The logical layers at the client side are schematized in Figure 3. The application makes use of CORBA stubs to access remotely the CORBA wrapper, therefore controlling the movements of the robots.

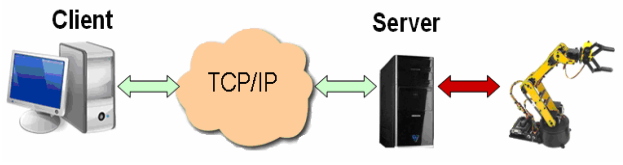

Figure 1. Architecture with a single robot.

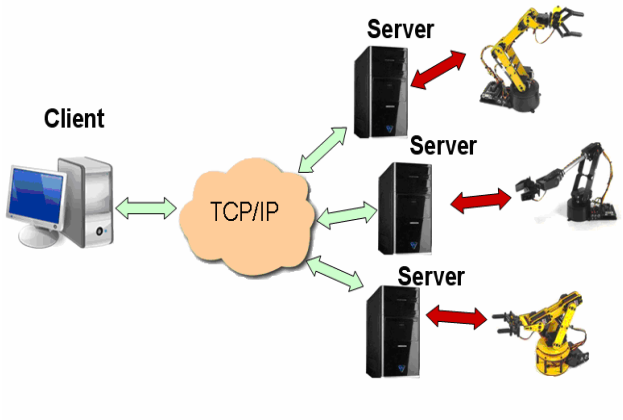

Figure 2. Generic architecture of the multiple robot control.

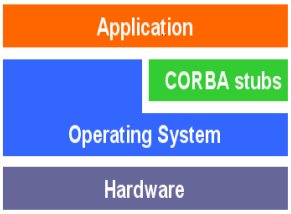

Figure 3. Logical layers at the client side.

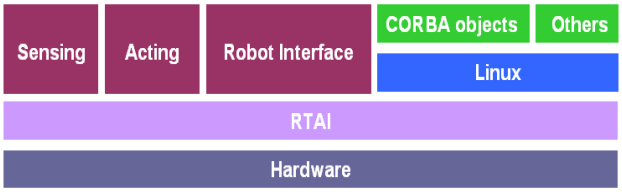

Figure 4. Logical layers at the server side.

Regarding the server side, the logical layers are schematized in Figure 4. This schema shows that only the RTAI kernel has straight access to the hardware. As a consequence, real-time tasks for sensing, acting and controlling the movements of the robot can meet their time constrains. Other activities, including the CORBA objects that encapsulate the robot communication with the client or other robots, are carried out through the Linux layer with not so stringent real-time restrictions.

A. *RTAI*

Some existing general-purpose operating systems (e.g. Linux [4]) can be improved in order to support real-time applications by using an intermediate layer responsible for reacting to critical operations (see Figure 4). The new intermediate layer, which is preemptive, encapsulates all hardware interruptions by means of a hardware abstraction layer (HAL) and provides a programmers' interface to be used by application tasks. Thus, those tasks that require real-time guarantees, such as those interacting with the sensors or actuators, may access directly this new layer without the interferences of a general purpose kernel. But the programmers' interface provided by the new layer is not compatible with legacy applications that are typically executed over the selected general purpose kernel. This problem may be solved by adapting the general purpose kernel (Linux in Figure 4) in order to be executed on top of the new preemptive layer so any application that runs on top of the general purpose kernel may use directly its programmer interface and services without being recompiled, obviously, without real-time guarantees.

There are two main solutions for Linux that follow this approach: RTAI and RTLinux. Both of them have the same roots and their performance has been compared by Ripoll [4] obtaining similar results. For this work RTAI (Real Time Application Interface) has been selected to execute the critical operations related with the control of the robot. RTAI is a real-time microkernel supported by the Polytechnic Institute of Milan in Italy. This GNU/Linux distribution is widely used in the academic environment.

An adapted kernel of Linux is executed on top of the RTAI microkernel in order to execute non-real time tasks and legacy applications.

B. *ORBit*

CORBA is a well established OMG standard to develop distributed applications. It makes use of an ORB (Object Request Broker), a software component to easy object communications hiding object location, implementation (including language, operating system and hardware) and state.

Since manufacturing devices typically require real-time communications, RT-CORBA [3] was one alternative for developing the robot wrapper. Unfortunately, RT-CORBA is a relatively complex technology that requires advanced programming skills. In addition, the RT-CORBA implementation should be adapted to be used with the RTAI microkernel, reducing the number of implementations available. So, the authors adopted a simpler solution based on the use of a low resource consuming implementation of CORBA, that it is executed over the Linux kernel. This choice was justified by the fact that the duration of the robot movements takes much more time (up to several seconds) than the time typically used in the communication tasks.

The CORBA implementation used, ORBit2 [5], is known for its outstanding performance. As a matter of example, ORBit2 was used for communication tasks in the GNOME project. This implementation was considered sufficiently stable and well documented.

In this architecture, the communications at the server side are carried out as CORBA objects that execute at the Linux level, i.e. with no Real Time guarantees as shown in Figure 4. However, as commented above, this will not jeopardize the operation as communications are relatively fast when compared to the movement of the axes.

It is also interesting to point out that several authors and vendors have ported some CORBA products to be used within RTAI and RT-Linux. One example may be found in Pérez et al. [10] where a distributed architecture for embedded systems is proposed based on the use of RTAI and ORBit.

III. WRAPPER INTERFACE

This section presents a generic wrapper for anthropomorphic industrial robots of up to 6 Degrees of Freedom (DoF). This wrapper, which is provided as a distributed object, may be easily included in distributed applications that require the use of robots. The interface implements most typical operations of this type of robots consisting of:

- Initialize the robot to the initial position
- Move the robot from one point to another
- Move the robot in a linear trajectory
- Move the robot an arc of a circular trajectory
- Open and close the gripper

The robot interface also has attributes to:

- Know the state of the robot and the gripper
- Set the approaching and moving speed

More specifically, Figure 5 shows the attributes and methods proposed for the generic Robot class. These will be described in the next paragraphs.

Some of the attributes are private, represented in Figure 5 with their names preceded by a minus (-), such as ArtiCoord, CartCoord, GripperStatus and Status, which keep internal information. In particular, ArtiCoord and CartCoord keep the location of the robot using its joint and Cartesian coordinates respectively. ArtiCoord is a vector with as many components as Degrees of Freedom that define each articulation in degrees in the case of a joint rotation or meters if it is a translational joint (θ_0, θ_1, θ_2, θ_3, θ_4, θ_5). CartCoord is a 6 element vector (x, y, z, a, b, c) that defines the position and orientation of the gripper with respect to a coordinate system located in the base, being a, b, c, the Euler angles for representing the orientation of the gripper. All robot movements and approaches change these values. GripperStatus defines the status of the gripper which may be Open or Close according to the homonymous methods described below. Finally, Status is a more general attribute that supplies the status of the robot. The values considered so far are Error, Waiting and Standby.

Other attributes such as RobotType, DoF, MovRange and Reach are public. These attributes, are represented in Figure 5 with their name preceded by a plus (+).

In particular, RobotType allows the configuration of the robot type to be used (Anthropomorphic, Cartesian, Cylindrical, Polar, Scara or Other). This attribute is related to the DoF attribute which describes the Degrees of Freedom of the robot. MovRange is an array that defines the maximum movement per articulation. Finally, Reach defines the maximum reach of the robot as a whole.

The proposed wrapper provides some methods that allow consulting the value of these attributes (not represented in Figure 5). The wrapper also proposes eleven additional methods that allow using the robot in distributed applications. These methods are distinguished in different categories: Movement operations and configuration operations.

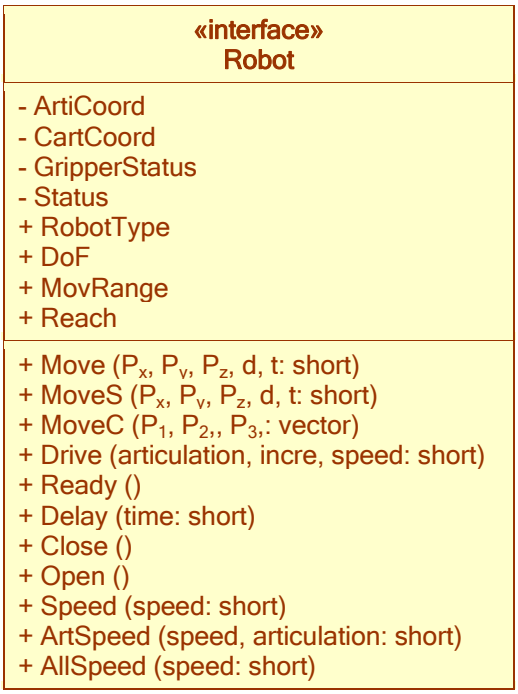

Figure 5. Interface for the robot.

The movement operations Move, MoveS and MoveC produce a change in the ArtiCoord and CartCoord attributes. The Move method is used to specify a free motion of the end effector, i.e. a movement from the current position to a new position given in Cartesian coordinates. MoveS is similar to the previous method but combines the movement of all joints in order to achieve a linear trajectory of the end effector. Both methods receive as a parameter the coordinates of the destination point (P_x, P_y, P_z), a distance (d) which provides the vertical position in Z axe of point P as well as a time interval (t) to reach the destination from which the movement speed will be calculated. Finally, MoveC produces a circular movement by interpolation of the three points whose Cartesian coordinates are provided in the method.

There are other movement operations provided by the wrapper such as Drive, which allows moving a single articulation a given increment or decrement of degrees at a given speed and Ready, which sets the robot to its initial position.

Other operations include, Delay which makes the robot wait during a certain time interval. It is typically before carrying out a movement in order to synchronize the robot movements. Also, some actions such as Open and Close are aimed at manipulating the gripper. Both of them affect the GripStatus private attribute. And the Speed method which defines the speed limit as a percentile of the real maximum speed of the end effector, to prevent from precision loose or damage. ArtSpeed is a similar operation but aimed at one articulation. Finally, AllSpeed configures the maximum speed for all articulations.

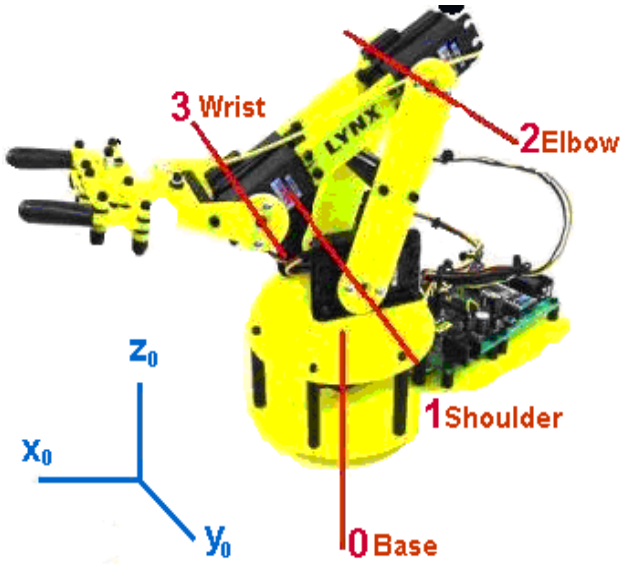

Figure 6. Robot Lynx axes.

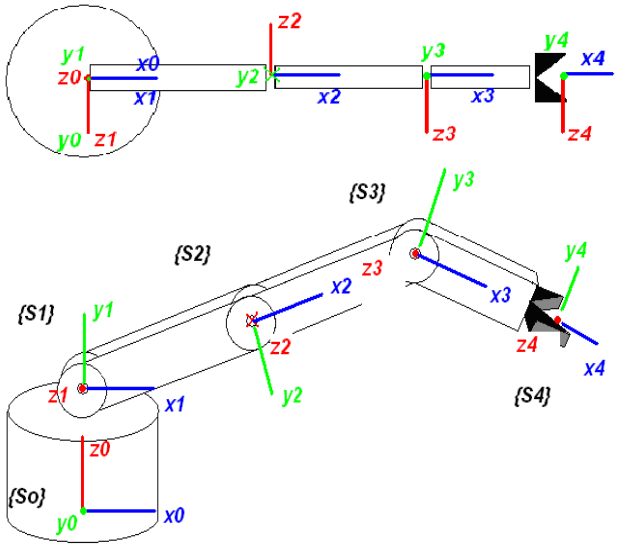

Figure 7. D-H Systems for the robot.

TABLE I.
DESCRIPTION OF THE MODEL'S AXES

	Name	Joint type	Axis	Turn	Start
0	Base	*Rotate*	^0Z	*Right*	0
1	Shoulder	*Rotate*	^0Y	*Right*	10
2	Elbow	*Rotate*	^0Y	*Left*	0
3	Wrist	*Rotate*	^0Y	*Right*	-45

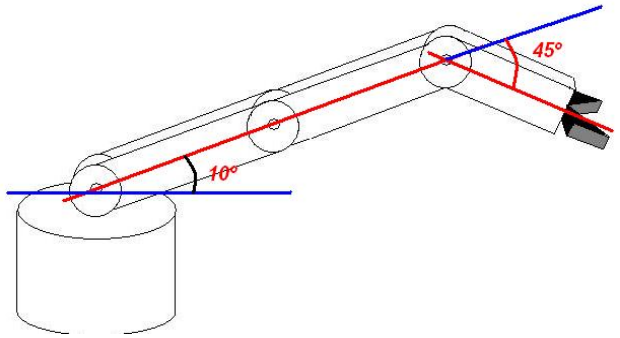

Figure 8. Initial position of the robot.

IV. ROBOT DESCRIPTION

The previous interface has been validated over a robotic arm. Namely, it has been implemented over a Lynx 5, which is a robotic arm with 4 rotation axes created by Lynxmotion. Even though this is an economical and compact robot no generality is lost since it reproduces most of the characteristics of more advanced and expensive industrial robots. In fact, this robot has become a very interesting alternative for educational environments. Figure 6 shows a picture where its 4 axis have been depicted. This section describes the main issues related to the description of the robot including the resolution of the Forward Kinematics and Inverse Kinematics problem which are used by the real-time tasks to produce the movements described in the previous section.

Table I describes the characteristics of these axes, which are shown in Figure 7. Note that except for the first axis of rotation (Base), which is on the 'Z' absolute axis, the other axes are over the 'Y' axis (see Figure 7 below). Also note that all axis turn rightwards except the elbow that turns leftwards. The robot can move in a fast, accurate and repetitive way on behalf to a set of embedded servo drives.

Since an origin of coordinates must be chosen for the movements of the robot, Figure 8 shows the initial position of the robot from which the movements will be executed.

A. *Forward Kinematics (FK)*

The solution of the Forward Kinematics problem (FK) of the robot is given by the Transformation Matrix T (1). This matrix T relates the Cartesian position (P_x, P_y, P_z) of the gripper or end effector respect to the reference coordinate system at the base of the robot: {X_0, Y_0, Z_0} with the input values of every joint (θ_0, θ_1, θ_2, θ_3).

$$T = {}^0_4A = \begin{bmatrix} nx_i & ox_i & ax_i & Px_i \\ ny_i & oy_i & ay_i & Py_i \\ nz_i & oz_i & az_i & Pz_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

In order to obtain this matrix, the Denavit and Hartenberg (D-H) convention is followed. This convention involves the use of an algorithm that allows building the resulting matrix, T, from the multiplication of a sequence of matrixes like (2) that represent the transformations of coordinates for every axis of the robot.

$$ {}^{i-1}_{i}A = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i S\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i C\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2) $$

Thus, matrix (2) describes the coordinate transformation from the previous coordinate system (*i-1*) to the next coordinate system (*i*). This matrix requires the calculation of four parameters, known as Denavit-Hartenberg parameters which are represent by θ_i, d_i, a_i and α_i. These parameters depend on the geometry of the robot. Table II shows the values of these parameters for each of transformations of coordinates for every axis of the Lynx 5 robot used in the case of study.

TABLE II.
PHYSICAL PARAMETERS USED IN THE D-H MODEL

	θ_i	d_i	a_i	α_i
1	θ_0	L_0	0	90º
2	θ_1+10º	0	L_1	180º
3	θ_2	0	L_2	180º
4	θ_3-45º	0	L_3	0º

Figure 9 shows the length of each robot component taking into account the geometry of the Lynx 5 robot.

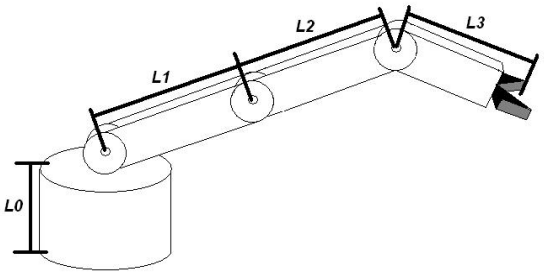

Figure 9. Length of the robot components.

By multiplying the individual matrixes of each system in a noncommutative way the solution of the Forward Kinematic problem is reached, i.e. the Cartesian position of the end-effector with (θ_0, θ_1, θ_2, θ_3) known.

$$ P_x = C_0 \left[L_3 C_{(1-2+3)} + L_2 C_{(1-2)} + L_1 C_1 \right] \quad (3) $$

$$ P_y = S_0 \left[L_3 C_{(1-2+3)} + L_2 C_{(1-2)} + L_1 C_1 \right] \quad (4) $$

$$ P_z = L_3 S_{(1-2+3)} + L_2 S_{(1-2)} + L_1 S_1 + L_0 \quad (5) $$

Where C_0 and S_0 are $\cos(\theta_0)$ and $\sin(\theta_0)$ respectively, $S_{(1\text{-}2+3)}$ means $\sin(\theta_1\text{-}\theta_2\text{+}\theta_3\text{-}35º)$, $C_{(1\text{-}2)}$ means $\cos(\theta_1\text{-}\theta_2\text{+}10º)$ and C_1 means $\cos(\theta_1\text{+}10º)$. L_1, L_1, L_1, represent the length of the arms as shown in Figure 9. The same notation has been used in the calculation of the other expressions.

B. Inverse Kinematics (IK)

The Inverse Kinematics (IK) problem allows to calculate the values of each joint (θ_0, θ_1, θ_2, θ_3) in order to locate the end effector of the robot at a specific point given in Cartesian coordinates (P_x, P_y, P_z), respect to the reference coordinate system at the base of the robot {X_0, Y_0, Z_0}.

In order to solve the IK problem an algorithm must be provided. Unfortunately, solving the IK problem for a robot is usually a more complex task than solving its FK problem since there are no general algorithms to solve it. In addition, a robot with 4 DoF like the Lynx 5 has several solutions to reach a given point (for example, *elbow up* or *elbow bellow*).

Frequently, iterative numerical methods or algebraic methods are used to solve the IK problem. However, in this case it is possible to find the solution in a closed form using geometric methods. This is a simple way to solve the IK by relating the coordinates of the end effector and the joint angles. This has been the approach followed in this case study due to its lower processing load when compared with other types of algorithms.

For the sake of simplicity due to the 'pick and place' nature of the robot operation described in the next section, it has been considered that the last articulation will always move in parallel to the base plain. Thus, the inclination of the last joint is 0 degrees, because of its parallel position with the base plain. This fact simplifies largely the consecution of a simple algorithm to solve the IK.

The closed form for each joint (with P_x, P_y, P_z known) is provided by the following equations (6-9):

$$ \theta_0 = \arctan\left(\frac{P_y}{P_x}\right) \quad (6) $$

$$ \theta_1 = \arcsin\left(\frac{P_z - L_0}{P}\right) + \arccos\left(\frac{P^2 + L_1^2 - L_2^2}{2 \cdot L_1 \cdot P}\right) \quad (7) $$

$$ \theta_2 = \arccos\left(\frac{-(P^2 - L_1^2 - L_2^2)}{2 \cdot L_1 \cdot L_2}\right) \quad (8) $$

$$ \theta_3 = \theta_2 - \theta_1 + 35º \quad (9) $$

Where P is obtained from expression (10):

$$ P = \sqrt{(P_z - L_0)^2 + P_x^2 + P_y^2 + L_3^2 - 2L_3\sqrt{P_x^2 + P_y^2}} \quad (10) $$

V. CASE STUDY

In this section a simple pick and place operation is described in order to illustrate how to use the wrapper proposed in section 3. This case study aims at demonstrating a scenario in which several robots need to cooperate in order to execute an operation.

In this case study, two Lynx 5 robots identical to the robot described in Section **Fehler! Verweisquelle konnte nicht gefunden werden.** were involved. Note that one of the advantages of using the proposed wrapper is that different robots can be used as long as they support the generic wrapper described in Section 3. Different robot configurations could even be exchanged without too much trouble.

Operation in both arms is similar and it is based on the operations of the wrapper described above. This approach

requires either using a central coordinator that implements applications with multiple robots in an easy way or that the operations are triggered by the nodes that recognize the images acquired by the cameras.

More specifically, pick and place operations at every robot require the use of the commands Move, MoveS or MoveC to execute the rough and approaching movements of the robots, as well as the Open and Close methods in order to get and leave the pieces. Also the Delay operation is used to synchronize sequences of movements among the robots.

A diagram for this operation is provided in Figure 10. Basically, a node with a camera starts the Pick & Place operation in Robot 1 by calculating the coordinates of the piece at the origin point of the 'Pick' operation in the Pallet 1 and then coordinates at the destination of the 'Place' operation in the Store 1. These coordinates are used to trigger the movement of the robots with the central coordinator (or directly by the camera node). In parallel, Robot 2 picks a piece from Store 1 and places it on Pallet 2 provided that at least one piece is present in a producer-consumer fashion.

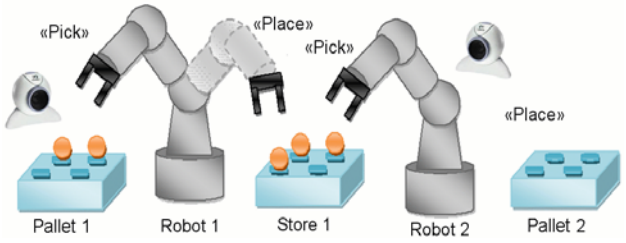

Figure 10. Pick and place case study.

VI. CONCLUSIONS AND FUTURE WORK

This article presents a generic CORBA wrapper to encapsulate industrial robots. This wrapper can be used to create distributed applications with multiple robots in an easy way either by coordinating nodes of the robotic applications or by triggering the operations from other nodes of the distributed application. The proposed wrapper is generic enough to be used with different types of industrial robots and provides operations to execute rough and approaching movements of the robot as well as to operate the gripper.

This wrapper has been implemented to demonstrate its viability for an economical and compact robot suitable for educational environments. It has been implemented over a Real-Time Operating System, namely Linux-RTAI which ensures determinism in the local operations of the robot, and a low resource consuming implementation of the CORBA specification, namely ORBit. Video demonstrations of the movement of one robot by using this wrapper may be found in [9].

One of the main advantages of this wrapper is its flexibility since it allows creating applications with multiple robots in an easy way. As a matter of example, a simple 'Pick&Place' operation that involves two robots is presented. This approach allows that students could create applications with multiple robots by using the proposed wrapper.

Currently, the authors are working on a graphical user interface from where the robot could execute different operations in an easy way.

REFERENCES

[1] A. K. M. Azad, M.O. Tokhi and M.H. Shaheed (2009). "A Virtual Environment for Studying Flexible Robot Manipulators", iJOE, 4 (4), http://dx.doi.org/10.3991/ijoe.v5i4.918

[2] OMG, Object Management Group, (2004) "*Common Object Request Broker Architecture: Core Specification*", Version, 3.0.3, March 2004.

[3] OMG, Object Management Group, "Real Time-CORBA Specification", Version, 2.0, November 2003.

[4] I. Ripoll, "A comparative analysis of RTLinux and RTAI", Linux Devices, Sep. 2002, at http://www.linuxfordevices.com/files/misc/ripoll-rtl-v-rtai.html

[5] The ORBit2 Project, at http://orbit-resource.sourceforge.net/

[6] R. Sanz, A. Hernando, C. Martínez and I. López, "Wrapping a Mobile Robot with RT-CORBA", Proceedings of the 8th International IFAC Symposium on Robot Control (SYROCO06), Bologna, Italy, 2006

[7] I. Calvo, I. Cabanes, A. Noguero, A. Zubizarreta, L. Almeida and M. Marcos, "Using a CORBA Synchronous Scheduling Service in Pick&Place Operations". Proc. of the 13th IEEE Int'l Conf. on Emerging Technologies and Factory Automation (ETFA). pp: 464-467. Hamburg, Germany, Sept, 2008.

[8] J. Garcia-Zubia, I. Trueba and D. Lopez-de-Ipina, "Web 2.0 Pharmacy Robots", iJOE, 6 (REV2010), http://dx.doi.org/10.3991/ijoe.v6s1.1389

[9] Robot demo at http://lsi.vc.ehu.es/wdocs/pub/robots/avi.html

[10] S. Pérez, J. Vila, J.A. Alegre and J.V. Sala, "A CORBA Based Architecture for Distributed Embedded Systems Using the RTLinux-GPL Platform", Proc. 7th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'04), 2004.

[11] RTAI – *The RealTime Application interface for Linux from DIAPM* http://www.rtai.org/

[12] D. Floroian, D. Ursutiu, F. Moldoveanu and L. Floroian (2010). "RoboSmith: Wireless Networked Architecture for Multiagent Robotic System", iJOE, 6 (4), http://dx.doi.org/10.3991/ijoe.v6i4.1468.

AUTHORS

Isidro Calvo Gordillo is with the University College of Engineering of Vitoria-Gasteiz, Department of Control Engineering. University of the Basque Country, Spain (email: isidro.calvo@ehu.es)

Itziar Cabanes Axpe is with the Faculty of Engineering of Bilbao. University of the Basque Country, Spain (email: itziar.cabanes@ehu.es)

Ismael Etxeberria Agiriano is with the University College of Engineering of Vitoria-Gasteiz, Department of Computer Languages and Systems, University of Basque Country, (e-mail: ismael.etxeberria@ehu.es).

Guillermo Sánchez Vitorica has been with the Faculty of Engineering of Bilbao. University of the Basque Country, Spain.

Ekaitz Zulueta Guerrero is with the University College of Engineering of Vitoria-Gasteiz, Department of Control Engineering. University of the Basque Country, Spain (email: ekaitz.zulueta@ehu.es)

This work was supported in part by the University of the Basque Country through grant EHU09/29. Received 26 May 2011. Published as resubmitted by the authors 26 October 2011.