Modelling a Software Architecture for Robots Control using UML and COMET Architectural Design Method

Francisco J. Ortiz, Alejandro Martínez, Bárbara Álvarez, Andrés Iborra, José M. Fernández-Meroño

Universidad Politécnica de Cartagena, División de Sistemas e Ingeniería Electrónica, Campus Muralla del Mar, s/n. Cartagena E-30202, Spain
{Barbara.Alvarez, Francisco.Ortiz, AlejandroS.Martinez, Andres.Iborra,JoseM.Fernandez}@upct.es
http://www.dte.upct.es

Abstract. In this paper, a control system in the domain of teleoperated service robots is presented. A reference architecture - ACROSET - has been analyzed and designed following a concurrent object modeling and architectural design methodology (COMET) that uses UML as describing language. The architecture of the whole system has been implemented in a ship's hull blasting robot - GOYA –using Ada 95 and GLADE. Our previous experience in developing teleoperated service robots using Ada is also presented.

1 Introduction

The objective of this paper is to present the development process followed to obtain a control system architecture for teleoperated robots, using the Unified Modeling Language- UML [6], and to describe the software implementation of such architecture on an industrial PC with Linux using Ada 95 [1].

We have experience in the development of control systems with Ada[4]. In par-ticular, several teleoperation systems for maintenance activities in nuclear power plants have been implemented using Ada [3]. In figure 1, a scheme of a teleoperation system is shown. In general, this type of control systems consists of two units: teleoperation platform and control unit.

The operator is in charge of monitoring and operating the robot according to the in-formation provided by the teleoperation system. This system receives commands from the operator and performs the corresponding actions for executing them. For this pur-pose, it communicates with the robot control unit, which physically actuates on the robot to move it. The robot control unit makes some sensing from the robot in order to evaluate its global state and send this information to the teleoperation system, which uses it to represent graphically to the operator the state of the robot and ensure the correctness of its behaviour. Different tools are attached to the robot for performing the maintenance operations. The tools are operated in a similar way to the robot [12].

1.1Previous experiences

Some teleoperation systems that we have implemented using Ada are: ROSA (Re-motely Operated Service Arm) [2], IRV (Inspection Retrieving Vehicle) system [13] and TRON (Teleoperated and Robotized System for Maintenance Opera-tion in Nu-clear Power Plants Vessels) system [10]. The experience of using the Ada programing language has been excellent in all the cases. A reference software archi-tecture was obtained for the teleoperation platform [2] and the first implementation was carried out with Ada 83 for ROSA system, which is used for inspection and repair-ing of the tubes inside the steam generators. The lan-guage already provided excellent support for creating portable applications and the use of generic packages al-lowed us to reuse the ROSA code for working with differ-ent robot tools. The same code that was developed for ROSA system was reused later for implementing the other systems on different hardware platforms and with differ-ent operating systems.

It is very important to notice that the design and de-velopment process of the me-chanical system in teleoper-ated service robots it must lead us to the best solution that satisfy the functional requirements. In this process is necessary to choose the appro-priate actuator and sen-sors for each degree of freedom of the robot. The actua-tor and sensors system for a robot freedom degree could be really different (e.g pneumatic actuators, hydraulic actuators, electrical engines as asynchronous or synchro-nous motors, etc). Because of that, the control strate-gies could be very different as well.

1.2Reference Architectures

Our experience demonstrates that commercial choice of axis controllers cards, despite being a reliable and ro-bust solution, supposes a restriction when choosing the proper actuator system because of the necessity of use electrical engines as actuators.

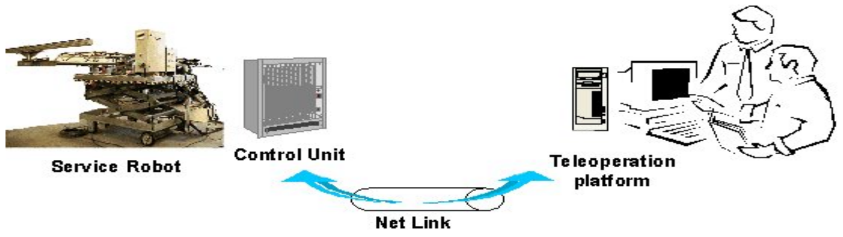
Because of this, a very important goal in our work is to show a reference architec-ture for control systems in the domain of teleoperated service robots. This architec-ture is not conditioned by the specific control strategy of the actuator system, the number of freedom degrees and the variety of tools that the robot manages.

A reusable reference architecture can be implemented in different hardware platforms and can be executed on many operating system. In many systems, secure and robust local control units were employed. These units were based on the use of elec-tromechanical elements for controlling the different robots. The local control unit communicates with the remote teleoperation unit, which offers a more complex functionality to the operator. However, the functionality of the control unit can increase if more flexible platforms are used. For example, ROSA control unit [3] was based on Vx-work and VME platform. The use of a special real-time operating system, which provides features such as kernel reliability, timers with enough precision, bounded kernel preemption and other characteristics, allows guarantee time requirements. However, the greatest disadvantage when using this type of solutions was that they were more expensive than other operating systems more widely used. They also presented distinct interfaces and development tools.

On the other hand, based on our experience, we can assure that there are control systems that have not such stringent safety and time requirements that justify the use of real-time operating systems. A failure in the system execution or sporadically missed time requirement does not imply an immediate threat. This is the case of a number of control applications which are supervised or teleoperated by human, such as robots for certain maintenance operations in nuclear power plants or robots for ship

hull blasting.

Nowadays, we can present our experience using Ada 95 for developing a new teleoperated robot: GOYA system (figure 2). In this work, some features of Ada 95 for object-oriented programming have been employed: tagged types, related concepts such as class wide and abstract types that did not exist in Ada 83. We have also used remote procedure call (RPC) through the distributed system annex. The obtained ar-chitecture for the teleoperation platform has been reused again in this system and a new reference architecture for the control unit has been de-



veloped in order to be re-used on different platforms [4].

Fig. 1. Teleoperated service robot system scheme.

In the next section, GOYA system is briefly described. The design process using UML [9] is presented in section 3. Section 4 describes the actual implementation of GOYA system, focussing on the control unit using Ada 95.

2 System Description

GOYA is a teleoperated system for blasting applied to hull cleaning in ship mainte-nance [12]. The main objective of this project was to develop a reliable and cost effective technology regarding hull grit blasting, capable to obtain a high quality surface preparation together with a dramatic reduction of waste and zero emissions to environment.

This technology was integrated in a full-automated and low-cost blasting system. Figure 2 shows the mechanical subsystem that consists of the following functional modules[1]:

–Elevation platform (z-axis): This mechanical part consists of a hydraulic elevation
system that is ascended or descended by a hydraulic actuator. The minimum height,
reached on the z-axis, is 800 mm and has a career of elevation of 2500 mm. There-fore, it is able to clean the fringe of the ship between 800 mm and 3300 mm high.

–Positioning arm (y-axis): It is intended to move away or approach the tilting head to the surface of the ship, on the y axis. It is built starting from two mobile guided rails, each one supported for a pair of skates. In their other end, the rails support a pneumatic cylinder without rod that carries the blasting tool. The useful career of the arm, on the y axis, is of 4000 mm from the end of the elevator table. The positioning arm is moved by an asynchronous motor engine, witch is commanded by a frequency variator. Once the titling head is touching the hull surface, the torque made by the arm increases up to a predefined value. While the torque is raising, the tilting head is mechanically auto-orientated in order to place the tool on the right position (perpendicular to the ship hull). The frecuency variator stops the motor when the limit torque is reached.

–Tool positioning cart (x-axis). The tool is mounted on a sliding cart that is moved
by a pneumatic cylinder without rod. This covers the x movement of the tool.

[1] A more detailed description of the mechanical system is shown in [11]

–Tool. The abrasive material is shut against the ship hull through a hose. Its opening and closing is controlled by a pneumatic system.

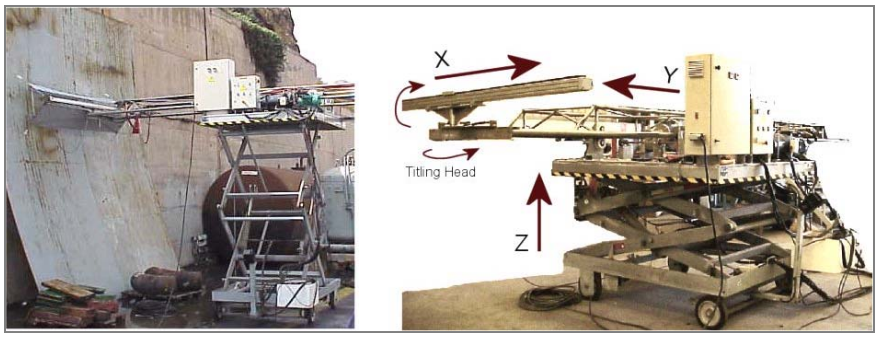


Fig. 2. GOYA system. On the right, the robot with *xyz* positioning possibilities. On the left, one of the initial tests on shipyards. The titling head is adapted to the surface.

The control unit incorporates the possibility of working in two different ways: teleoperated and local modes. In the teleoperated mode, the operator monitors and operates the robot according to the information provided by the teleoperation system. This teleoperated mode will be the normal manner of operation. For security purposes, the control unit can control the robot without communication with the teleoperation system through a local and electromechanical interface based on buttons, switches, indicators and displays.

3 Design Process

One of the most important issues around software architecture is the description of the system structures under consideration. It is the basis for all design activities including comprehending, communicating, analysing, trading-off, as well as for modification, maintenance, and reuse. Similar to other models, the description can be based on mathematical, textual, or graphical notations, but in order to manage the complexity of a system, a complete architecture description should be divided into multiple views. Often, each architectural view includes a set of models that describes one aspect of a system. One well-known and widely used approach to multi-viewed architectural description is the 4+1 *View Model of Architecture* proposed by Kruchten [11]. This model has also been adopted in the development of *Unified Modeling Language* (UML) [7].

UML has emerged as a standard notation for conceptual modeling using the object-oriented paradigm. Taking into account the benefits of blending object-oriented concepts with concurrency aspects, the use of UML notation is quite helpful when designing distributed and real-time applications. The UML notation provides several diagrams [6] that allow us to represent static and dynamic properties of real systems and integrate them following the previous 4+1 architecture as we show in this section.

3.1 Concurrent Object Modeling and Architectural Design Method with UML

In order to obtain a reference architecture we have followed the COMET methodology (Concurrent Object Modeling and Architectural Design Method with UML) proposed by Gomaa in [9]. It is a design method for concurrent applications based on the USDP (Unified Software Development Process) and the spiral model of Boehm.

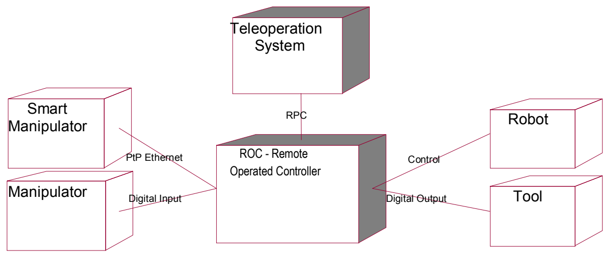


Fig. 3. Deployment diagram of the GOYA system. Smart manipulator is a PDA.

Starting from the system Use Cases, a static and dynamic design of the classes in the architecture can be derived until reaching the final implementation. Our goal is to reach a reference architecture for the design of control units in teleoperated service robot: ACROSET. In this paper, the process to obtain the reference architecture is presented, the architecture must be as complete as possible to be reused in other robots, perhaps more complex that the system presented here.

In figure 3, a possible deployment diagram of the whole system is presented, where different nodes are included.

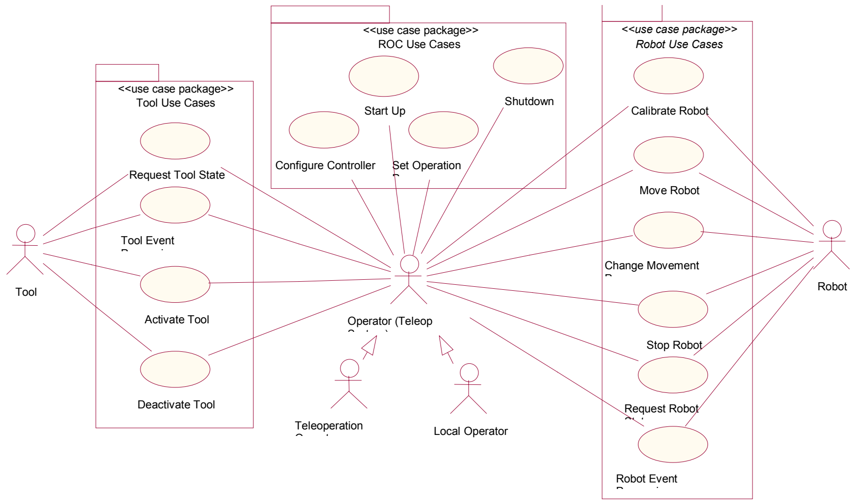


Fig. 4. General Use Cases of the system

Following the development process, once the requirements of the system are collected (functional and non-functional), we create a detailed tabular specification of the system functionality. It is divided into categories where attributes (as time response, fault tolerance, etc) are included. From such specification, the use cases of the system are extracted.

A system context class diagram is derived from use case diagram by considering the actors and which devices they utilize to interface with the system.

3.2 Discovering Classes

After the previous step, every Use Case is studied in order to obtain the objects that take part in it and the exchanging messages between these objects. This is the most complicated phase in the development process and it needs a big creativity effort from the designer. Several collaboration diagrams are a consequence of this study. Once the different objects of the system are extracted from the collaboration diagrams, the classes of the system can be proposed as a generalization of objects.

One of the main objects composing the control unit is the *Joint_Controller*, which has to implement several methods as *move_to, stop,* etc. Therefore, the control architecture is based on the class *Joint_Controller*, defined as interface or abstract class. Each controller could be different, so it will be an implementation of

Joint_Controller, giving the same interface to the rest of the system. It will be as many controllers as joints the robot has, one for each joint. Each of them implements its own control algorithm, which could be only software or an interface to a hardware control board. It is clear then, that if a coordinated movement is needed, there should be a coordinator of controllers, as shown in

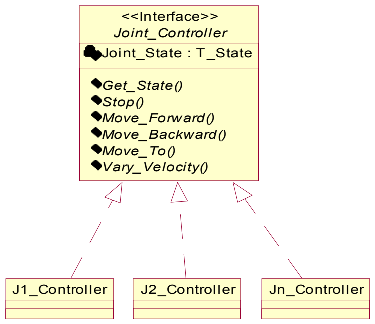


figure 6. This figure represents the class diagram of the architecture. The class *Joints_Coordinator* offers different basic methods of coordination between joints.

Fig. 5. Joint_Controller implementation diagram

The class *Tool_Controller* is similar to *Joint_Controller*, excepting the object to control. In the last case it is dedicated to the tool, implementing a different controller for each possible tool that could be managed by the robot. The same remark could be done for *Tools_Coordinator*.

The process coordinator establishes the highest level in this architecture. Although the domain of the application is teleoperated service robots, there are several processes that can be performed in an autonomous manner. *ProcN_Coordinator* implements one of these processes. For each one of the possible autonomous processes, there should be a different Process Coordinator, changing in run time depending on the process.

3.3 Concurrent Tasks Structuring

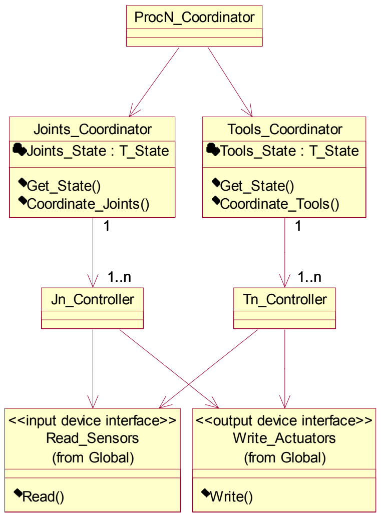
During the task-structuring phase, the concurrent task architecture is developed. As a consequence, the system is structured into concurrent tasks and the task interfaces and interconnections are defined. To help to determine the concurrent tasks, task-structuring criteria is provided by COMET to assist in mapping an object-oriented analysis model of the system to a concurrent tasking architecture.

For instance, depending on the characteristics of the I/O devices (asynchronous, passive, etc), one or more tasks will be chosen to read them. That is to say, if the sam-pling rate of two passive devices differs we should choose two different tasks, but if it is similar, it could be simplified in one task depending on the computa-tional necessi-ties of the system. See section 4.2 to complete these concepts.

3.4 Implementing the Design: Code Generation

Once the static and dynamic behavior of the system has been designed, it is time to implement it depending on the better deployment in each case.

As described above, all the analysis and design of the Software can be accomplished by means of a description language as UML is. We have got every diagram to explain the behavior of the Software we are designing. Part of the source code of the application can be obtained from the diagrams thanks to the Code Generation AddIn that UML tools have. We use Rational Rose 2000 and in this sec-



tion, some tips of the Code Generation tool will be ex-plained.

Fig. 6. Proposed architecture class diagram

The Ada Code Generator AddIn that can be found in Ra-tional Rose:

–Substantially reduces the elapsed time between design and execution.

–Produces uniformly structured source code files, promoting consistent coding and
commenting styles with minimal typing.

The code generated for each selected model component is a function of that com-ponent specification and code generation properties, and the model properties.

These properties provide the language specific information required to map the model onto Ada.

Usually we have a component view of the system where packages of the software to produce are displayed (see Fig. 7). The first step in code generation consists of assigning classes in the UML model to every module. If a class specification assigns it to a module, the Ada generator uses this information to determine where to generate the declaration and definition for the class.

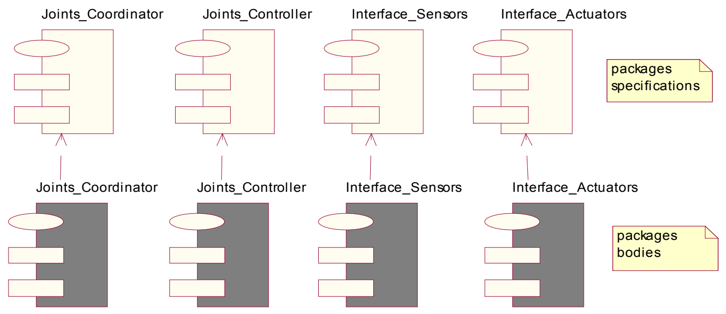


Fig. 7. Components view. Classes can be associated to this packages specifications and bodies

The declaration of the type representing that class is placed in the corresponding package specification, along with the other types assigned to the same package. The declarations of the subprograms associated with that class also go in the same package specification. The bodies of these subprograms are placed in the corresponding package body. Each class specification must contain the desired attributes, relationships and operations. The Ada generator uses this information to generate record components and subprograms.

The Ada generator uses the specifications and code generation properties of com-ponents in the current model to produce Ada source code. For each class in a Rose model, this generator produces a corresponding Ada type. Associations, relationships, and attributes are translated to components of that type.

The implementation files are generated simply in one mouse click. These files contain one package body, with the appropriate ‘with’ clauses. This package body contains global declarations, skeletal subprogram, tasks and protected object bodies, and code regions. The code generator provides a complete body for some of the subprograms it generates. For other subprograms, including the user-defined ones, it only produces a skeletal body. In all cases, the generated bodies contain protected code regions. By placing each subprogram implementation within its code region, this implementation code is preserved when code is regenerated from the model. We have to remark that Rational Rose 2000 can only generate ‘skeletons’ of the program, the code necessary to perform the dynamic behavior of the system has to be ‘handily’ programmed. In any case, if a new class is introduced, by means of reverse engineering, Rose can reflect the change in the model.

All generated files are placed in a hierarchy of directories that correspond to class categories and/or subsystems in the model.

4 Implementation Details Using Ada

The main components of the Goya system are the Teleoperation Platform and the Control Unit, linked by Ethernet, and finally the mechanical system of Goya robot.

1. Teleoperation Platform: the operator commands remotely the robot through it. It has been implemented by a workstation SGI with Irix 6.5.8. There are three main process running on it:

–Graphical user interface, which has been developed with GtkAda and Ada95.

–Kinematic control module, through GRASP, a commercial software intended to design and simulate robots.

–Teleoperation platform controller, developed with Ada 95. This controller communicates with the two process, described above, with a communication protocol using TCP sockets. Using Ada 95 has facilitated the implementation of reading and writing tasks in different communication channels. Furthermore the marshalling and unmarshalling of data types exchanged between different processes developed in C and Ada 95. To communicate with the robot control unit, distributed system annex (GLADE) [14] with Ada 95 has been employed. We have proved the benefits of using GLADE instead of developing our own protocol based on TCP sockets as we did in previous projects.

2. Control Unit, implemented with Ada 95 on an Anvantech industrial PC. Goya system is a service robot that works at low speed. Once we have found out the critical tasks, we have estimated that their response time are wide enough to allow the use of GLADE and Linux on the

industrial PC. It is an operating system that doesn’t have real-time characteristics. Because of an economic criterion and its well-known features, Linux (Debian distribution) becomes the ideal operating system for this application. The compiler version for Ada 95 and GLADE were 3.14a from ACT. We have used digital input/output cards and encoder cards mounted on the PC. Each card has its own address space mapped into the PC memory. The manufacturer provides the card’s control drivers with C functions, following the files treatment from Unix (open-read/write-close). Thanks to the Ada 95 advantages for interfacing with other languages, as C, it has been easy to export C functions by means of ‘‘pragma export’’. In this way, we have Ada functions to manage directly the hardware.

4.1 Control Unit Architectural Description

The reference architecture explained in section 3.2 has been implemented in Goya system. The Goya robot has three freedom degrees (xyz) and one tool. Then, four controllers are necessary, one for each freedom degree and one for the tool. In figure 5, a class diagram is shown with *Jn_Controller* and multiplicity 1..n; the implementation in an object diagram for this particular robot leads to: J1_Controller for the elevation platform (z-axis), J2_Controller for positioning arm (y-axis) and J3_Controller for tool positioning cart (x-axis) mounted on the titling head. We only have one tool in this robot, so the multiplicity of *Tn_Controller* will be 1: T1_Controller for the blasting tool. Over this joints controllers there is a coordinator object (Joints_Coordinator) that is required to coordinate movements. This abstract class is implemented with the appropiate procedure *Coordinate_Joints* for this robot. The Tools_Coordinator is not necessary in this application because we have only one tool, but finally it is implemented to respect the architecture, offering the same interface to the rest of the application in prevention of later modifications and improvements of the robot and anticipating possible tool interchanging.

The top layer is the Process_Coordinator. We have in this application an object that has implemented a state machine performing the automatic sequence for blasting a complete hull panel. The interface offered by Process_Coordinator is the same for any layer that accesses to the controllers , so every control order, not only coordinated ones, but even control for individual joints pass through the Process_Coordinator. The same could be said for Joints_Coordinator. We have created layers with the same interface to the upper layer.

Jn_Controller and Tn_Controller are protected objects as Read_Sensors, Write_Actuators are.

4.2 Control Unit Tasks Model

In Fig. 8 the tasks model in this application is presented. There is a task for Process_Coordinator, a task for Joints_Coordinator and two tasks for each controller (one for reading sensors state and one for writing actuators). It must be noticed that *writing tasks* are not periodic, they are suspended by means of a protected entry with a barrier.

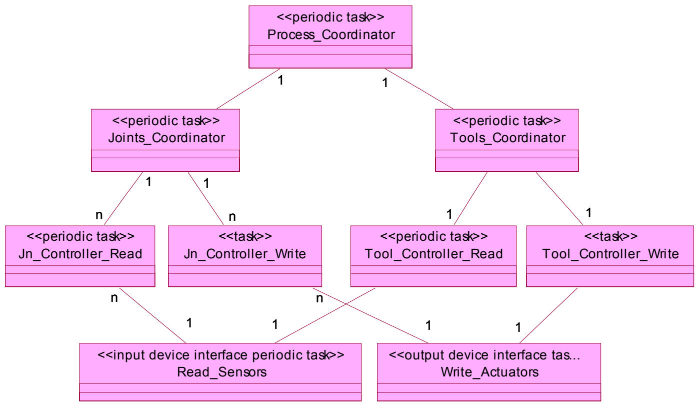


Fig. 8. Tak diagram. Different stereotypes are used.

All these tasks are needed because we are controlling different joints, many times in a simultaneous way or even the operator could give orders to any joint while other joint is moving. Coordination is needed to perform coordinated movements with different strategies of control. The robot can implement also some autonomous operations, that is why the system needs also a *Process_Coordinator* These tasks are encapsulated in the objects shown in figure 6.

A periodic task that reads sensors and a non-periodic task that writes actuators perform the interface with hardware devices. Following the task structuring criteria from Gomma [9] we have chosen only one task for reading sensors because the actualization period in I/O cards is the same. The process of writing is similar, the writing task is activated when there is an entry for next operation.

The Read_Sensors object is implemented as a <<protected>> object using one important feature in Ada 95. In this manner, all the controllers can read at the same time by means of function Get_State(). The data of sensors in this protected object is actualized by the

Read_Sensors task. Being protected we can assure that the different hardware does not write at the same time the data, avoiding the danger of loosing information. It is necessary to remember that in this application the controllers share the I/O hardware (digital cards), in the same card we have input from platform, arm
and head. We assure that every controller accesses properly to its resorts with the pro-tected object Read_Sensors.

The three tasks Jn_Controller_Read are periodical, they are continuously checking the state of the sensors, but the writing tasks are active only when there is an order of movement for the actuators. There is also another task (Write_Actuators) to write the orders to the hardware, which is activated only when there is an entry (also protected entry).

In the case of sensor data and actuation, communication between tasks is performed by means of information hiding objects. As it has been mentioned, there are protected objects to pass information between the controllers and the hardware interface.

Messages are used for communicating *Coordinators* tasks and *Controllers* tasks. There is no need to introduce additional queue object because the operator orders are queued in the *Teleoperation Interface* system. If necessary, a buffer can be implemented in *ProcN_Coordinator* (fig 6). In any case, writing attempts in any protected object would be queued in a *FIFO* manner.

4.3 Using the Ada 95 Distributed System Annex: GLADE [13]

A goal in this reference architecture is to give the same interface to the local system and the teleoperation system. This interface is a set of procedures, to send commands, and a function to get the actual robot's state. The only difference is that the local system accesses directly to this procedures and function, meanwhile the teleoperation system accesses remotely.

We have taken advantage of using GLADE through the remote procedure call. Due to using GLADE to communicate the Teleoperation Platform Controller and the Control Unit, apparently the teleoperation Platform Controller is running on the industrial PC.

Although in the present implementation we have only one processor, the use of GLADE and this interface objects allows distributing easily the application in different processors.

5. Conclusions

Although the use of Ada in general industry applications is much less extended than other languages as C or C++, it is the language selected for the implementation of the system, due to some features that allow us to obtain an extra portability, maintainability and reliability. Some of these key issues in Ada are mechanisms for encapsulation, separate compilation and library management, exception handling or data abstraction.

Some features of Ada 95 for object-oriented programming have been employed:
Tagged types, related concepts such as class wide and abstract types that did not exist in Ada 83.

In general, the use of Distributed System Annex of Ada is not appropriate for developing hard real-time systems, but it is possible to use it to develop systems without stringent time and safety requirements as GOYA.

The use of GLADE and the well-interfaced structure of the proposed reference architecture allow distributing easily the application in different processors if needed. Thanks to RPC, the application works in the same manner in distributed systems than if it would be working in the same machine.

UML and Software development methods are indispensable to manage the complexity of big software products. The COMET methodology, used to obtain a reference architecture, and Rational Rose, with Ada 95 Code Generator, have been greatly useful to reach an implementation of a control unit in GOYA system.

References

1. Ada 95 Reference manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995. Available from Springer-Verlag, LNCS no. 1246.
2. Alonso, A., Álvarez, B., Pastor, J.A., de la Puente, J.A., Iborra, A. Software Architecture for a Robot Teleoperation System. 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control, (1997)
3. Alvarez B, Iborra A, Alonso A, de la Puente, J.A, and Pastor, J.A. Developing multi-application remote systems. Nuclear Engineering International. Vol.45, No. 548, (2000)
4. Alvarez B, Iborra A, Sanchez P, Ortiz F, and Pastor, J.A. Experiences on the Product Synthesis of Mechatronic Systems using UML in a Software Architecture Framework., ITM'01 Istambul, Turkey (2001)
5. Barnes, J. Programming in Ada 95. Addison-Wesley, 2nd Ed, New York (1998)
6. Booch G., Jacobson I., Rumbaugh J., Rumbaugh J. The Unified Modeling Language User Guide, Addison-Wesley Pub Co, New York, (1998).
7. Burns, A., Wellings, A.: Concurrency in Ada, Cambridge Univ. Press, Cambridge (1998)

8. Douglas, B.P.: Real-Time UML. Developing Efficient Objects for Embedded Systems, Addison-Wesley Object Technology Series, Reading, Massachusetts (2000)
9. Gomaa, H.: Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley Object Technology Series, Reading, Massachusetts (2000)
10. Iborra, A., Alvarez, A., Navarro, P.J., Fernández. J.M, and Pastor, J.A. Robotized System for Retrieving Fallen Objects within the Reactor Vessel of a Nuclear Power Plant (PWR). Proceedings of the 2000 IEEE Internat. Symp. Industrial Electronics. Puebla, Mexico, (2000)
11. Kruchten, F. – Architectural Blueprints – The "4+1" View Model of Software Architecture, IEEE Software, USA (1995)
12. Ortiz, F., Iborra, A., Marin, F., Álvarez, B., and Fernandez, J.M. GOYA - A teleoperated system for blasting applied to ships maintenance. 3rd International Conference on Climbing and Walking Robots, Madrid, Spain (2000)
13. Pastor, J.A, Alvarez, B., Iborra, A., Fernández, J.M. An underwater teleoperated vehicle for inspection and retrieving. First International Symposium on mobile, climbing, and walking robots. Brussels, Belgium (1998)
14. Pautet, L., Tardieu, S. GLADE user´s guide. Technical report version 3.14a. ACT.