Reference architecture for robot teleoperation: development details and practical use

Bárbara Álvarez*, Andrés Iborra*, Alejandro Alonso†, Juan Antonio de la Puente†

**Universidad Politécnica de Cartagena, Dpto.Tecnología Electrónica,Paseo Alfonso XIII, 50, 30203-Cartagena, Spain*

†Universidad Politécnica de Madrid, Depto. Ingeniería de Sistemas Telemáticos,Ciudad Universitaria s.n., E-28040 Madrid, Spain

***Abstract**: The need to avoid redundant efforts in software development has been recognized for a long time. Currently, work is focused on the generation of products that are designed to be reused. A reference architecture for robot teleoperation systems has been developed using the domain-engineering process and certain architectural patterns. The architecture has been applied successfully to the development of different teleoperation platforms used in the maintenance activities of nuclear power plants. In particular, this paper presents how the reference architecture has been implemented in different systems, such as the Remotely Operated Service Arm (ROSA), the Teleoperated and Robotized System for Maintenance Operation in Nuclear Power Plants Vessels (TRON) and the Inspection Retrieving Vehicle (IRV).*

***Keywords**: Software architecture; Robots control; Real-time systems; Teleoperation; Software engineering*

1. Introduction

Software reuse has become an important factor in current developments due to market competitiveness and time-to-market requirements. Although the interest in reusing software arose with the origins of the programming, this issue has not been practiced with success as yet (Prieto-Diaz, 1991) (Tracz, 1995). One reason is the difficulty of combining existing software components. In addition, although reusability at this level is a good practice, it is not enough. One way of raising the degree of reuse is to apply this approach to software architecture, which comprises the software components, their visible properties and their relationships (Bass, Clements & Kazman, 1998).

In the next section, the stages of the domain engineering process are described (Withey, 1994). Its goal is to obtain a *Domain Specific Software Architecture* (*DSSA*). This process was developed in the context of the program *Software Technology for Adaptable and Reliable Software* (Prieto-Diaz & Arango, 1991). Its success is derived from the fact that the software requirements can be satisfied in different ways, and implementation limitations restrict the ways in which these requirements can be fulfilled. Previous works on domain analysis did not allow for obtaining a design model.

The question is how to develop a software architecture that can be effectively reused. The common approach is to concentrate on a domain area. It seems feasible to reuse a software architecture in a set of systems with common features. Then, a suitable practice is to develop a reference architecture that takes into account the special properties of a type of system in a domain area. A reference architecture is defined as a division of functionality together with data-flow between pieces mapped onto software components (Bass et al., 1998).

The authors were involved in the development of a teleoperation system. In addition to the common functional requirements of this type of system, the product should be easily maintained and adaptable to different operational environments and robots. Many investigators have described robot-control architectures based on a specific operating system and programming language. In these cases, the usual approach is to replace the robot controllers by generic controllers (Hayward & Paul, 1986) (Hayward & Hayati, 1988), (Li, Tarn, & Berjczy, 1991). These systems are focused on robot control and tasks planning, and little support is given to teleoperation. In other cases, the system does not allow the operator to interact dynamically with the system (Brooks, 1986). In Albus, McGain & Lumia (1989) a reference architecture for robot teleoperation, that focuses on the control of autonomous systems, operating in non-structured environments is described. It uses complex and expensive artificial intelligence techniques. None of these approaches fulfils the non-functional requirements previously described.

The domain engineering process was used by the authors and, as a result, (A. Alonso, Álvarez, Pastor, de la Puente & Iborra, 1997) a Domain Specific Software Architecture (DSSA) for robot teleoperation systems was developed. Since the publication of that paper, the refer-

* Corresponding author: Tel: +34 968 325 654;
Fax:+34 968 325 345
E-mail address: barbara.alvarez@upct.es

ence architecture has been used with great success in the development of a number of teleoperation systems for different robots and tools. The goal of this article is to revisit the approach taken from a more abstract and mature point of view, and to describe the cases where it has been used.

Section 2 shows the process followed to identify the components, relations and requirements of teleoperation systems. In Section 3, the way certain architectural styles were selected for the interaction among subsystems is described. In Sections 4, 5 and 6 several teleoperation robot systems developed with the proposed reference architecture are described. Section 7 includes some conclusions.

2. Domain engineering

The process of domain engineering covers all the activities required to build a common software core for a family of systems (Withey, 1994). This process was applied to building a reference architecture for robot teleoperation systems (Alonso et al., 1997). The following activities were performed (Fig. 1 shows this process):

1) Domain analysis: in order to identify the set of components that are common to teleoperation applications. The result was a domain model.
2) Domain design: in order to obtain a generic design, based on the previous result and on the study of patterns or common models.
3) Implementation of the design: based on reusable components that can be used in different products.

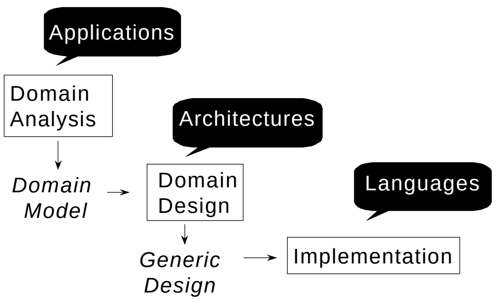

Fig. 1 Domain-engineering process

There are several methods of performing a domain analysis but most them do not offer techniques to capture and to represent the information related to a family of systems. FODA (*Featured-Oriented Domain Analysis)* (Kang & Cohen, 1990) offers these techniques and supports reusability, not only at the purely functional level, but at the architectural level as well. The FODA method proposes the use of certain models for performing a domain analysis. In particular, it proposes the following activities: (1) a context analysis, that allows the environment of the domain to be defined and (2) a functional analysis, in order to identify the similarities and differences of the existent systems. As a result of the application of this method to teleoperation systems, a generic specification of requirements has been obtained. In Alonso et al. (1997) functional and non-functional requirements for a general teleoperation system are described, as well as the basic functional requirements or services that the system should provide to the end users, and the timing requirements. In these systems, time requirements should be met in order to ensure that the information that the operator receives is valid and reflects the current state of the robot.

The following step in the domain-engineering process is domain design. There are many studies that describe how to perform a domain analysis, however these do not explain how to use the results of such an analysis to obtain a generic design. The approach described in Peterson & Stanley (1994) was used. It groups the elements that work together for performing a certain task in subsystems.

The next step in defining the reference model for teleoperation systems is to split the functionality among a number of components and to identify the data flow among the pieces. Fig. 2 shows how the teleoperation reference model is mapped onto software components. In short, the functionality of these components or subsystems is the following:

- **Graphical representation**. This subsystem is in charge of showing the operator the current state of the robot and the environment in which it is operating. It provides operations for initializing the representation and for updating the status of the robot, according to the information received from the remote control unit.
- **Collisions detection.** This subsystem provides the required functions for checking whether a given movement command is safe, in the sense that the robot does not collide with the operating environment or with itself.
- **User Interface**. It is in charge of interacting with the operator. It allows him to issue commands to the robot and to know their execution status.
- **Communications**. This subsystem embodies the communication protocol with the remote control unit. It provides a means to send commands and to receive status information. In this module, a mechanism for fault treatment should be included to guarantee safety, in case communication with the control unit is lost.
- **Controller.** It is the core of the system and it is in charge of executing the commands from the user, by using the other components and sending the appropriate orders to the robot. It also receives the robot´s status from the remote control unit, to check its consistency and to show it to the operator.

In this domain, it is common to have a robot with different controllers. This is true in the case of a moving arm, where it is necessary to consider the control of an articulated arm whose base can be fixed on a vehicle and whose end joint holds a tool for performing specific operations. In this case, the teleoperation system has to deal with three controllers and the proper synchronization mechanisms should be implemented.

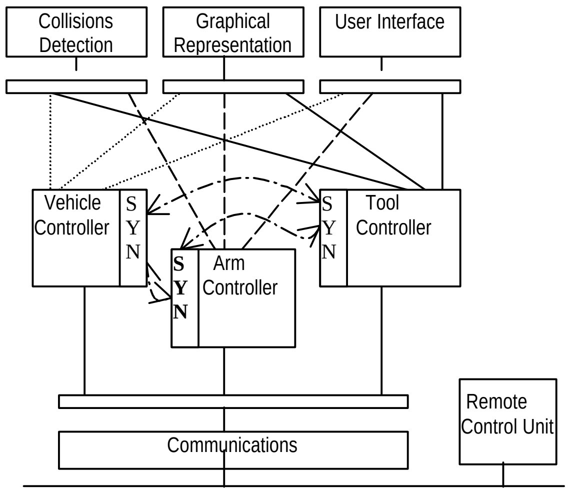

Fig. 2 High level architecture description.

Refinement of the reference model

In order to get the reference architecture it is necessary to specify how the different components are going to interact. For this purpose, the approach followed is based on characterizing the interaction between the components and selecting an appropriate architectural style, which defines a particular pattern for run-time control and the data transfer mode. The use of architectural styles allows the use of well-defined, well-understood and successfully tested interaction mechanisms. They help to develop systems with desirable properties, such as maintainability and portability.

Two architectural styles were used to define the interaction between the components: client-server (Berson, 1992) and communicating processes, according to the classification in L. Bass et al. (1998). The client-server style is appropriate when there is a component that provides a service and another that requests it. It allows the server or the client to be changed, as long as the defined services are requested and provided following the designed interface. This is the case in the interactions between the *Graphical Representation* and the *Collisions Detection* components with the *Controller* component. The first component provides services for showing the status of the robot. The second checks whether or not could result in a collision a possible robot movement. In these two cases, the interfaces were implemented based on messages, in order to allow distribution. The interaction is asynchronous, because the controller cannot be blocked for a long time while it waits for a service to be completed.

A general communicating processes style was used for the interaction between the *User Interface* and the *Communication* components with the *Controller*. This is because there is not necessarily a cause and effect relationship that guides these interactions Any of the components can take the initiative to send data. For example, consider the information that the *Communication* component sends to the *Controller*. It can be periodic status data or aperiodic alarms, caused by some unexpected robot behavior. On the other hand, the controller may send commands to the robot at any time. The same holds for the relationship between user interface and the controller(s). A controller sends sporadic messages to the user interface when the robot's status changes or some failure occurs in the system. The user sends commands to the controller when a certain robot behavior is required.

In the internal design of the components, two styles were used: layered and object-oriented. The layered style is useful to achieve portability and easy modification. It is based on structuring the components as a set of layers and it was used in the design of the *Communications* component. Only some of the layers need to be changed if the system has to be ported to different hardware or has to use alternative protocols.

The internal design and implementation of the rest of the subsystems is based on object-oriented and abstract data-types styles. These paradigms emphasize the bundling of data and how to manipulate and access that data. Data encapsulation promotes reusability and easy modification.

The robot's control should not be interrupted for any reason. Therefore, certain mechanisms have been introduced for decoupling the control task from the rest of the subsystems and for receiving the information from them. In this way, if a failure occurs in another subsystem, the controller can continue operating. These mechanisms are modules in charge of communicating with the rest and translating data. The communication of these modules with the interfaces is by means of procedure calls. When a message is sent to a controller, these decoupling modules store it in a buffer, so that the controller is not interrupted. Each controller can be seen as a control loop which continuously checks the received messages buffer.

The general controller that has been designed performs the following operations: (1) it receives commands from the operator, (2) it checks if the operations are feasible, (3) it simulates the movements before executing them, (4) it sends commands to the remote control unit, and (5) it updates the state of the system. This design has shown to be general enough to be used for the implementation of controllers for different robots. In case a robot has several controllers, as mentioned in the previous section, step 2 includes synchronizing them.

This kind of system may have time requirements that must be met. The *Rate Monotonic Analysis (RMA*) (Klein, Ralya, Pollak, Obenza & González-Harbour, 1993) theory allows the designer to reason with confidence about timing correctness at the tasking level and to analyze whether tasks deadlines can be met. In this way, a reference architecture has been developed and a framework for analyzing its timing response has been built, as described in Álvarez, Alonso & de la Puente (1998).

The three teleoperation systems where the reference architecture is used do not have hard real-time requirements. The three robots move quite slowly and hence the time requirements were relatively long and soft. In addition, there was no special safety requirement. As a result the only component with hard real-time requirements was the *Remote Control Unit*.

The rest of the article presents the development of different teleoperation systems using this reference architecture.

4. The ROSA system

The *Remotely Operated Service Arm* (ROSA) system provides a remote user interface for controlling a jointed arm with six axis (Fig. 3). Different tools are used for inspecting and repairing the tubes in the steam generators.

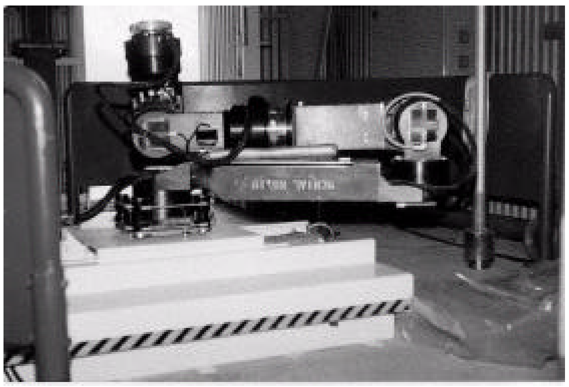
Fig. 3 The ROSA arm.

The development platform is a Hewlett Packard 9000 model 725 workstation, with the HP-UX operating system. A local area network connects the teleoperation platform with the robot control unit. The remote controller is based on an Heurikon HK68/V30XE processor, a VME bus, an amplifier, and several control cards attached to it.

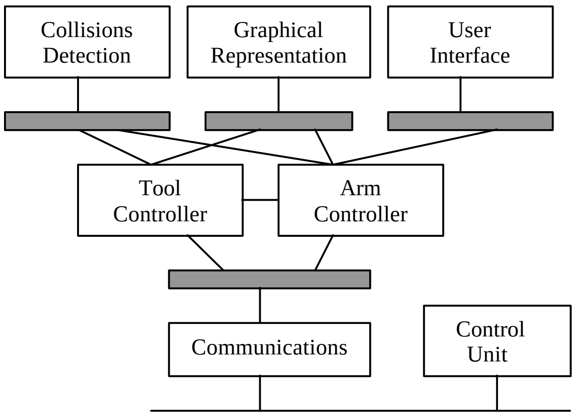

Fig. 4 High-level architecture for ROSA.

Fig. 4 shows a high level description of the architecture for the ROSA system. This scheme includes two controllers: a robot and tool. A generic controller was developed in Ada, since its modular characteristics and generic facilities make the development of reusable code easier.

The implementation of the components for the graphics and the collision detection were based on a commercial tool called Robcad. It allows the user to define the robot and its operation environment in 3-D. It automates the operations for collision detection and its graphical output is excellent. These components were developed in the programming language C++, since the libraries offered by this commercial tool were developed in this language.

The user interface was developed in the programming language ANSI-C. The resources offered by X Windows were used by means of User Interface Language (UIL) and Motif.

The rest of the architecture has been implemented in Ada. In particular, the communications subsystem was based on TCP and UDP sockets. A package of asynchronous real-time Ada drivers for interconnected systems exchange (PARADISE) was used. This package offers an interface to the communication routines of the Unix operating system. Generic modules were developed for the interfaces among the subsystems. They offer services for communication among processes.

For the planned steam generator maintenance operations, it was necessary to develop a number of mechanical tools to perform specific operations in the tubesheet of steam generators, such as: detecting the wrong tubes. An inspection process based on eddy currents is used, to cancel the wrong tubes, to recover them, to drill plugs and to place the nozzle-dams.

Tools controllers have been developed for dealing with these tools. They are different instances of the generic controller described in the previous section. The required parameters of these software packages are related to the elements used to characterize the mechanisms (jointed arm and tools). These elements are specific commands and state machines. This allows the definition of: (1) robot status, (2) its evolution after a command execution and (3) the commands that are feasible for each state.

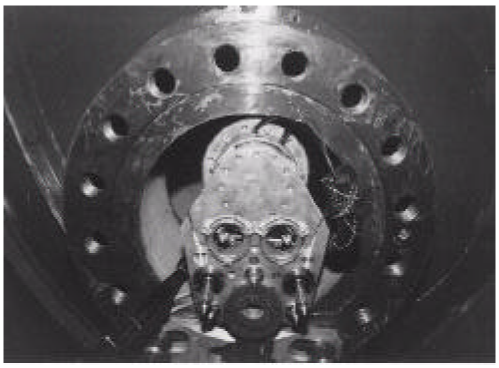
Fig. 5 Electro-disintegration machine.

A controller was developed for an electro-disintegration machine (Fig. 5). This tool was designed for the disintegration of unwanted elements which could be located in the tubesheet (plugs, drills, etc.). The tool consists of the following parts: a cone for its anchoring to the robot's end, a fixed platform on which the vision devices are installed, linear potentiometers for aligning it with the tubesheet and some camlocks for anchoring it to the tubesheet. The electro-disintegration head is located on the surface of a sliding table. Commands were provided to insert the electro-disintegration head in the wrong tube, to activate it, to set the intensity and voltage, and to load and unload the tool.

Two tool controllers are used to tighten and to control a gripper which is used for the positioning of the nozzledams. Fig. 6 shows a gripper. This tool consists of four pneumatic pistons. These activate four pincers that are used for positioning the nozzledams in the primary circuits.

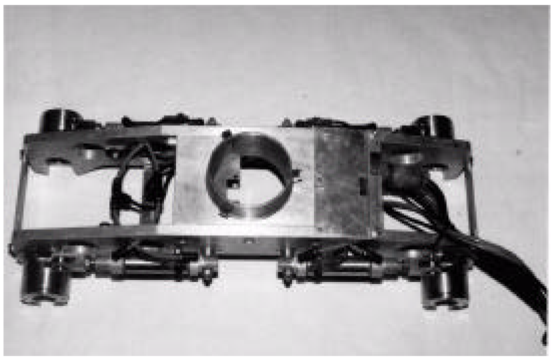
Fig. 6 Gripper.

The fasteners and nuts that hold the nozzledams are installed by means of a screwdriver (Fig. 7). In the case of the gripper, commands are provided to control the pincers and anchor the tool to the robot. To control the screwdriver, some commands to adjust the head, to screw and to unscrew are provided.

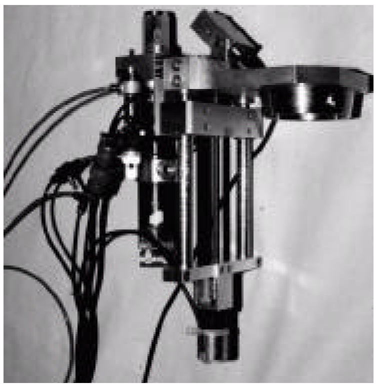
Fig. 7 Screwdriver.

A commercial machine was adapted for welding plugs in the tubesheet of the steam generator. Fig. 8 shows this tool. Commands to activate the head, to gauge the tool and to load and unload the tool are provided.

For each tool, a common model was used for developing the user interface and the generic communication module. In this case, appropriate parameters were provided for connecting with the tool processes of the remote control unit.

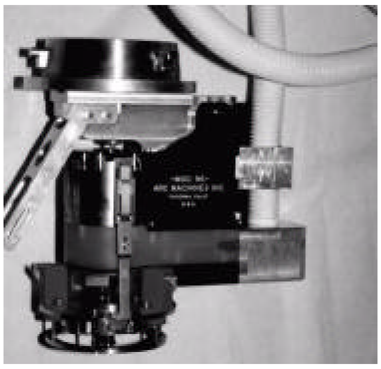
Fig. 8 Welding machine

5. The IRV system

The *Inspection Retrieving Vehicle* (IRV) system is a teleoperated vehicle with sensors, lights, cameras and interchangeable end-effectors. The vehicle can operate ten meters underwater, inside pipes of 400 mm and larger. This system is used for retrieving foreign objects from inside the primary circuit nozzles of nuclear power plants. Fig. 9 shows the vehicle.

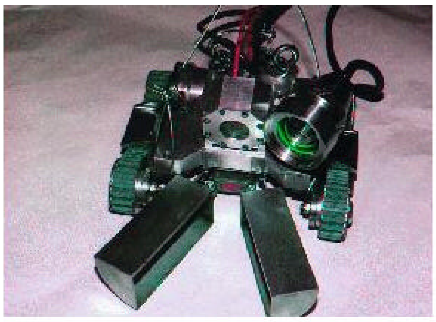
Fig. 9 Vehicle of the system IRV.

The hardware platform for this teleoperation system is the same as that in the ROSA system. Most of that system was directly reused. In particular, it was necessary to implement three controllers: for the vehicle, arm and tool. These controllers communicate among themselves in order to synchronize the operations of the controlled elements. In order to implement the controllers, the generic packages developed in the ROSA system were instanced for the IRV system. The generic parameters are related to the operations of the devices (commands and state machines).

The *Graphic Representation* component consists of an artificial vision system. Its function was to identify the edges of the objects and the most significant elements of the environment and to superimpose a wire frame representation of these onto real images provided by cameras (Iborra, Lázaro, Domínguez & Campoy, 1993). The input information for this module is sent, due to efficiency reasons, through an independent channel directly from the cameras. This subsystem is implemented on a different hardware platform (a personal computer). The rest of the teleoperation system runs on a HP 9000/725.

In the current implementation there is no collision detection subsystem, and the graphical representation subsystem does not include a model of the environment or devices. They are not necessary for the current operations, but they can be added using commercial tools. The user interface subsystem is based on Motif. The vision system and the communications modules have been developed in the programming language C. It is important to note that the changes in the *Graphical Representation* component did not require modification of the rest of the system.

6. The TRON system

Teleoperated and Robotized System for Maintenance Operation in Nuclear Power Plants Vessels (TRON) is a robotized system used for retrieving objects in the nuclear plant's reactor vessels. Fig. 10 shows this system. Due to human errors during the recharging fuel operation, objects can fall into the vessel. This system can be introduced trough holes, which are called bottom internals, to inspect the vessel and to recover the objects without having to dismantel the nucleus.

The whole system comprises a jointed pole, the end-effectors and a navigation system based on artificial vision techniques, that helps the operator to move through really complex environments (dark and full of obstacles). The pole consists of four joints. The end-effector and the inspection cameras are attached to the end link. The reduced dimensions of the inlet (3.8 cm) prevents the use of more complex mechanisms.

In this system, a real-time operating system was not required. The pole moves very slowly. Therefore, potential collisions with the environment can be detected and the motors can be disabled before it happens. For this reason, the controllers and user interface components were implemented on a Pentium PC, running Windows. The programming language was C++. New interface modules among subsystems were developed without modifying the communication mechanisms among them.

In this case, the architecture design was reused. The implementation language and the execution platform were different to those in the previous cases, hence there was no code reuse. The generic controller class was implemented at the top of the hierarchy class. The pole and end-effector controllers were derived from such a mechanism controller class. The object-oriented programming paradigms allow software designs to be adapted or extended if new functionality has to be added. As in previous systems, the pole and the end-effector were described in terms of their basic commands, their state machine and their structural and dynamic models.

The *Graphical Representation* and *Collisions Detection* components run on a HP 9000/725 work-

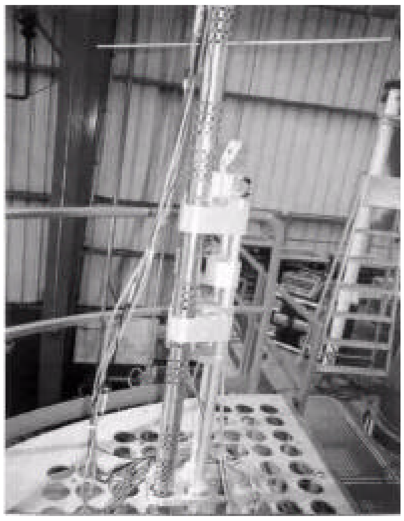
Fig. 10 TRON system.

station and the utilities provided by Robcad were used. In this case, the collision detection module is very simple because it does not require inverse kinematics. Communications links between processes running on the PC (user interface and controllers) and the processes running on the workstation (graphical representation and collisions detection) are done with TCP/IP sockets.

7. Applications development summary

This section, describes the differences and similarities between the previous systems. In this way, it will be easier to understand the development decisions taken. Table 1 tries to summarize this information.

Table 1. Characteristics of the robot systems

System	**ROSA**	**IRV**	**TRON**
Motion Speed	Slow	Slow	Slow
Real-Time Requirem.	Soft	Soft	Soft
Operation Environm.	Known	Unknown	Unknown
Execution Platform	HP 9000	HP 9000	Win. NT

The basic requirements for a teleoperation system are similar in most cases, to provide a means for allowing an operator to send commands to the robot and to retrieve its state information. For the three reuse scenarios, a common characteristic of the robots is that their speed of motion is slow. This implies that the real-time requirements are not very tight and therefore easy to fulfil. Hence, the worst response time required for a command to be sent to the robot or to receive the status was relatively long, and in the range of one to three hundred milliseconds. This implies that the hardest real-time requirements were to do with the control of the motors and the processing of the sensors of the robot. These functions are implemented in the *Remote Control Unit* component, which is outside the scope of this paper.

The working environments of the robots were another important issue that affected the decisions taken. In the case of the ROSA system, it was necessary to know very precisely the structure of the steam generator and the geometry of its tubes, in order to perform the maintenance operations correctly. This information allowed the environment to be modeled with an appropriate tool and to have this information accessible during the system's operation. The advantage of using the facilities of the commercial tool was that it was very easy to dynamically change the viewpoint of the representation. This facility was very appreciated by the operators that used the system.

In the case of the IRV and TRON systems, it was not feasible to know their operational environment with the same level of detail. For this purpose, cameras were mounted on the robots. Instead of showing the operator a graphical representation of the robot and its environment, the images taken by the cameras were used.

The final aspect that has an important influence on the system implementation was the execution environment of the teleoperation system. In the case of the IRV and the ROSA systems, exactly the same platform was used. Obviously, this allows the developers to raise the level of reuse, which dramatically reduces the development time and effort. In the TRON case, the requirements imposed by the contractor made it necessary to choose a different platform. This implied that although the reference architecture was reused, most of the code was rewritten.

8. Conclusions and future work

The application of the domain-engineering process and software architecting techniques were the basis for the development of a reference architecture for robot teleoperation systems. Several architectural styles have been selected for describing the interaction rules among the components. Each style depends on which qualities are required.

The suitability of this architecture has been validated by its use in the development of different products with the same basis and its application to other robots. The use of the Ada language facilitated the development of generic components that could be reused for different products of the same system. Furthermore, the interaction between components written in Ada with other subsystems written in C has not been a problem.

In the three systems, the software has been tested successfully with a real robot and a 1:1 mock-up of the nuclear plant part where it will work. The development of these tools and their control software results in the acquisition of a proprietary technology for maintenance works in nuclear plants. Due to the qualities of the reference architecture, the cost of this development was affordable.

The reference architecture was demonstrated to be appropriate for systems such as those presented in previous sections. One common characteristic of these systems is their slow motion speed. The development team would like to use the reference architecture with faster robots. It could behave correctly, although some modifications may be necessary for guaranteeing time requirements, such as the need to use a separate computer for the graphical representation and using a real-time operating system.

There are plans to modify the reference architecture in order to allow it to teleoperate more that one robot from the same computer. This improvement will be probably used for dealing with a number of slow-motion robots designed for cleaning ships surfaces.

Finally, some components of the implementation of the architecture in Ada will be rewritten using the new language standard approved in 1995. In this way, it would be possible to take advantage of language features, such as the object-oriented facilities, hierarchical packages, and the real-time and distributed annexes.

9. Acknowledgments

This work has been partially supported by the Spanish Government Programmes for Research in Electrical Power (project PIE-041049), and for Technological Actuation in Industry (PAUTA projects 753/95 y 53/96). TRON is supported inside EUREKA–MAINE program (EU1565).

10. References

Albus, J. S., McGain, H. G., & Lumia, R. (1989), *NASA/NBS Standard Reference model for Telerobot Control System Architecture (NASREM)*, National Inst. Standards and Tech., Technical Report 1235, Gaithersburg, USA.

Alonso, A., Álvarez, B., Pastor, J.A., de la Puente, J.A. & Iborra, A. (1997). Software Architecture for a Robot Teleoperation System. *4th IFAC Workshop on Algorithms and Architectures for Real-Time Control*. Vilamoura, Portugal.

Álvarez, B., Alonso, A., & de la Puente, J.A. (1998). Timing Analysis of a generic robot teleoperation software architecture, *Control Engineering Practice* 6(6), 409-416.

Bass, L., Clements, P. & Kazman, R. (1998), *Software Architecture in Pratice*, Addison-Wesley. Masschusssets, U.S.A.

Berson, A. (1996). "*Client/Server Architecture*". McGraw-Hill. New York, U.S.A.

Brooks, R. A. (1986) A Robust Layered Control System for a Mobile Robot. *IEEE Transactions of Robotics and Automation*, 2(1).

Hayward, V. & Hayati, S., (1988) KALI: An environment for the programming and control of cooperative manipulators, *American Control Conference*. Atlanta, USA.

Hayward, V. & Paul, R. (1986). *Robot manipulator control under Unix: A robot control C library*. The International Journal of Robotics Research 5(4).

Iborra, A., Lázaro, M. A., Dominguez, S., Campoy, P., Alvarez, M. & R.Aracil (1993). An automatic system for the real time integration of live action and synthetic 3-D computer images. *4th Eurographics Animation and Simulation Workshop*. Grenoble, France.

Kang, K. C. & Cohen, S. (1990), *Feature-Oriented Domain Analysis (FODA) feasibility study*. Technical report, CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, USA.

Klein, M. H., Ralya, T., Pollack, B., Obenza, R. & M. Gonzalez Harbour, (1993). *A Practitioner´s Handbook for Rate Monotonic Analysis*. Kluwer Academics Publishers. Massachussets, USA.

Li, Z., Tarn, T. J. & Berjczy, A. K. (1991) Dinamic Workspace Analysis of Multiple Cooperating Robot Arms, *IEEE Transanctions on Robotic and Automation*, 7(5).

Peterson, A. S., & Stanley Jr, J .L. (1994), *Mapping a Domain Model and Architecture to a Generic Design*. Technical report, CMU/SEI-94-TR-008. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, USA.

Prieto-Diaz, R. (1991), "*Reuse in the U.S.A.*". In Proceedings of the 13th Annual International Conference on Software Engineering. IEEE Computer Society Press. Austin, USA.

Prieto-Diaz, R., Arango, G. (1991). *Domain analysis and software systems modeling*. IEEE Computer Society Press. Los Alamitos, USA.

Tracz, W. (1995) *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*. Addison Wesley Publishing Company. Massachusetts, USA.

Withey, J. V. (1994). *Implementing Model Based Software Engineering in your Organization: An Approach to Domain Engineering* Technical report, CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, USA.