AN IMPLEMENTATION OF A TELEOPERATED ROBOT CONTROL ARCHITECTURE ON A PLC AND FIELD-BUS BASED PLATFORM

F. Ortiz, B. Álvarez, F. Losilla, D. Rodríguez[1], N. Ortega[1]

Universidad Politécnica de Cartagena,
División de Sistemas e Ingeniería Electrónica
Campus Muralla del Mar,s/n. Cartagena, E-30202, Spain
E-mail address: francisco.ortiz@upct.es
[1]IZAR Carenas Cartagena
Ctra. De la Algameca s/n. E-30205, Cartagena, Spain

Abstract: Teleoperated robots are used to perform hazardous tasks that human operators cannot carry out. The purpose of this paper is to present a new architecture (ACROSET) for the development of these systems that takes into account the current advances in robotic architectures while adopting the component-oriented approach. ACROSET provides a common framework for developing this kind of robotized systems and for integrating intelligent components. The architecture is currently being used, tested and improved in the development of a family of robots, teleoperated cranes and vehicles which perform environmentally friendly cleaning of ship-hull surfaces (the EFTCoR project). This paper summarises the features of this project and describes the implementation of ACROSET on a hardware platform based on a PLC (*Programmed Logic Controller)* SIEMENS SIMATIC S7-300 and the Field-Bus PROFIBUS DP.

Keywords: Architecture, Programmable Logic Controller, Field-Bus, teleoperation, robot.

1. INTRODUCTION

Teleoperated mechanisms, such as robots, vehicles and tools (or a combination of these), perform inspection and maintenance tasks in hostile environments. The capabilities and the areas of application of these systems grow from day to day, but so does their complexity. As stated in (Coste-Manière, 2000), one way of dealing with this complexity is *to use architectural frameworks and tools that embody well defined concepts to enable effective realization of systems to meet high level goals*.

There have been numerous efforts to provide developers with architectural frameworks of this kind (Bruyninckx, 2002)(Nesnas, 2003). The objects of this paper are twofold: to present an architectural approach to the development of control units for these systems and to present an example of its use in the development of a real system. The architectural approach, ACROSET, is based on the latest advances in robotic architectures and adopts a component-oriented approach. ACROSET offers a way to re-use the same components in very different systems by separating the components from their interaction patterns. It also provides a common framework for developing robotized systems with very different behaviours and for integrating intelligent components. The architecture is currently being used, tested and improved in the development of a family of teleoperated cranes and vehicles for environmentally friendly cleaning of ship hull surfaces (the EFTCoR project).

This paper is structured in five sections. Section two presents the architectural drivers that have guided the design of ACROSET and a brief description of the EFTCoR missions and mechanisms. Section three describes briefly ACROSET and section four the way in which such architecture has been used and instantiated to derive the concrete architecture of the

EFTCoR system and also the hardware and software components used. Finally section five summarizes the conclusions and future works.

2. THE TELEOPERATION DOMAIN AND THE EFTCOR SYSTEM

Teleoperated systems cover a broad range of mechanisms (manipulators, tools, vehicles) that carry out inspection and maintenance activities in hostile environments. Usually these systems perform a small number of highly specialized tasks. Such specialisation implies the use of different processors, communication links, man-machine interfaces, sensors, actuators, control algorithms, etc, each of them appropriate for the kind of missions that each system should carry out.

Despite all these differences, teleoperated systems are normally very similar from a logical point of view, having many common requirements in their definition and many common components, either logical or physical, in their implementation. These similarities allow the designer to define a common architecture for all such systems. To be able to use such an architecture for all developments is extremely useful. It allows rapid development of systems and reuse of a large variety of components, with concomitant savings in time and money. Considering the differences among the systems mentioned above, it is clear that the main objective of the architecture is to deal with such variability. To achieve that, there are a number of points that must be considered:

- Very different systems should be able to use the same components. This implies that the architecture design should make a clear distinction between components and their interaction patterns.
- The component implementations could be software or hardware; it is very advisable that such components be COTS (Commercial off the Shelf).
- It should be possible to derive concrete architectures for operator-driven systems and autonomous intelligent systems.

The EFTCoR project (EFTCoR, 2002) addresses the development of a family of robots whose mission is to retrieve and confine the paint, oxide and adherences from ship hulls. The EFTCoR system is part of the European Industry's current effort to introduce environmental friendly ship maintenance. Although the EFTCoR family of robots are specifically designed for ship hull maintenance, they still present a broad spectrum of behaviours and degrees of complexity and as such provide an excellent test bench for a reference architecture. The sources of variability in EFTCoR are the following:

- Hull dimensions and shapes differ widely.
- Different areas of any given hull impose very different working conditions for robotic devices.
- Working areas differ in different shipyards or even within the same shipyard.
- There are operational differences between cleaning small, discrete areas (spotting) and full blasting.
- Other hull maintenance operations can be included, such as fresh water washing and painting before and after coating removal.
- The particular businesses and cultures of shipyards impose different requirements and priorities. All shipyards are interested in all repairs, but each one focuses mainly on a particular type of ship and maintenance task.

The tremendous variety described above generates very different problems, and these require different robotic systems, each suited to a given type of shipyard, a given type of hull, a given part of the hull, a given maintenance operation, etc.

It may then be impossible to design a single robotic system to perform all tasks, but it is still possible to design the different robotic systems in such a way that as many components as possible are shared. EFTCoR's robotized systems generally consist of a primary positioning system capable of covering large hull areas and a secondary positioning system mounted on primary system that can position a tool over a relatively small area (4 to 16 m2). Different combinations of primary/secondary/tool have been considered and tested (see Fig. 1).

Finally, it is important to stress that the EFTCoR is an industrial project and as such should use components that are common in industrial facilities (PLCs rather than work-stations, field buses rather LANs, etc.)

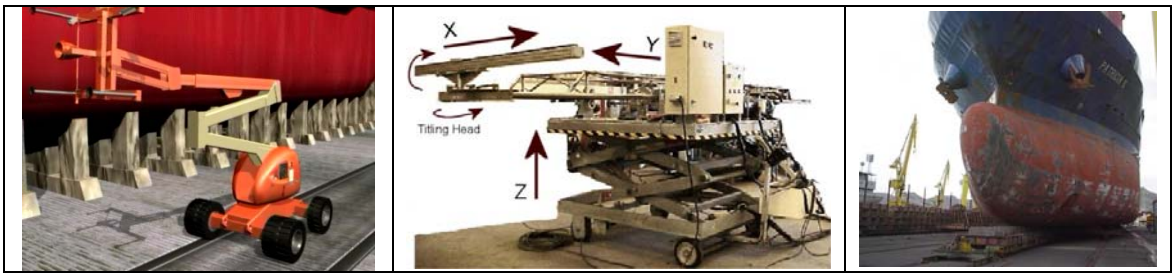

Fig. 1. Different primary/secondary/tool solutions for grit blasting.

3. ACROSET REFERENCE ARCHITECTURE

ACROSET (*Arquitectura de Control para Robots de Servicio Teleoperados*[1]) is a reference architecture for teleoperated service robot control units. The architecture emerged from previous works at the DSIE (División de Sistemas e Ingeniería Electrónica, Universidad de Cartagena, Spain) (Iborra, 2003), (Ortiz, 2000) and is currently being used, tested and improved in the EFTCoR project. ACROSET takes account of the sources of variability explained in section 2 and the architectural drivers developed to deal with them.

ACROSET is supposed to make it possible for very different systems to use the same components, and therefore the first step was to define the rules and common infrastructure that would allow components to be assembled or connected. To that end, the concepts of components, ports and connectors were adopted as defined in (Hofmeister, 2000). The connector concept allows components' functionality to be separated from their interaction patterns, because such patterns contained within the connectors. The notation followed to describe the components, ports and connectors is inspired by the 4 views of Hofmeister (Hofmeister, 2000) and ROOM (Selic, 1994), which extend the UML notation with stereotyped classes and special symbols. Rational Rose RT (Rational, 2002) was used to plot the diagrams.

The subsystems defined by ACROSET are shown in Fig. 2. The first subsystem of the architecture, which should be present in every system, is the *Coordination, Control and Abstraction Subsystem (CCAS).* The *CCAS* abstracts and encapsulates the functionality of the physical devices of the system. The *CCAS* is composed of *virtual* components defined by ACROSET, which can be implemented in either software or hardware. This subsystem breaks down into several components distributed in hierarchical layers (see section 3.1). Many of the components to be used in a robot control unit can be found on the market either as hardware devices and control cards or software packages for a given platform. Where COTS components are used, ACROSET offers two possible solutions: if the COTS component is commonly used, ACROSET defines its virtual counterpart; if not, the *Bridge* pattern is used to map an existing virtual component to the actual COTS interface.

For the last architectural driver (to deal with operator-driven and semi-autonomous systems), an *Intelligence Subsystem* (*IS*) is proposed. In this way, autonomous intelligence can be added if necessary, to act as another user of the *CCAS* functionality. This separation of intelligence and functionality enhances the modifiability and adaptability of the system to new missions and behaviours. The intelligence can be combined with the operator commands depending on the application or mode of operation. A *User Interaction Subsystem (UIS)* is proposed to interpret, combine and arbitrate between orders that may come simultaneously from different users of the system's functionality (*CCAS*), since the system does not concern itself with the source of the order.

Other important aspects besides the functionality or the intelligence of the system include the safety and the possibilities of configuration and management of the application. To differentiate between functionality *per se* and the monitoring of such functionality, a *Safety, Management and Configuration Subsystem (SMCS)* is proposed. Another function of this subsystem is to manage and configure the initialization of the application.

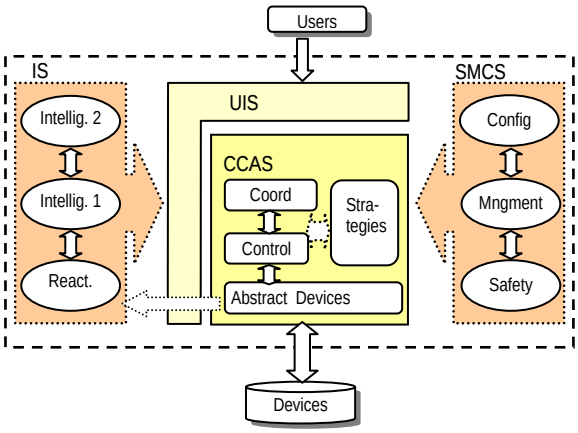

Fig. 2. An overview of ACROSET subsystems.

3.1 Components of the Coordination, Control and Abstraction Subsystem (CCAS)

The CCAS of a given system comprises components that are defined in four layers of granularity:

- Layer 1: Abstract the characteristics of atomic components, such as sensors and actuators.
- Layer 2: Simple Unit Controllers (SUCs).
- Layer 3: Mechanisms controllers (MUCs).
- Layer 4: Robot controllers (RUCs).

The simplest components modelled by the architecture are the sensors and actuators, which are defined at the lowest architectural level. The SUC components model the control over the actuators (Fig. 3). For example, there will be SUCs defined to control every joint of a given mechanism. The SUC generates commands for the actuator according to:

- The orders that it receives from another component (through the **`SUC_Control`** port).
- The information it receives from the sensors.
- Its control strategy.

The control policy is an interchangeable part of the SUC. For example, the **`Strategy`** of a given joint may be a traditional control (PID) or may be changed for a fuzzy logic strategy. The SUCs usually need to accomplish hard real time requirements and are therefore generally implemented in hardware.

[1] Control Architecture for Teleoperated Service Robots.

Where they are implemented in software they impose severe real time constraints on operating systems and platforms.

Defined at the third level of granularity is the Mechanism Unit Controller (MUC). The MUC component models the control over a whole mechanism (vehicle, manipulator or end effector). As Fig. 3 shows, the MUC is a logical entity composed of:

- An aggregation of SUCs.
- A Coordinator, which is responsible for coordinating SUC actions according to the commands and information that it receives. Its installed coordination strategy.
- Its coordination strategy.

The coordination strategy is an interchangeable part of the SUC. For example, the **`Strategy`** of a given manipulator may be a particular solution for its inverse kinematics, the coordinator strategy for a given vehicle could be a particular navigation strategy, etc.

Although the architecture defines the MUCs as relational aggregates, they can actually become components (hard or soft) when the architecture is instantiated to develop a concrete system. Whether or not the interfaces of the inner SUCs are directly accessible is a decision for the architecture instantiation. In fact, although MUCs may be implemented in either hardware or software, they are very commonly commercial motion control cards that constrain the range of possible commands to its internal components. COTS elements limit the flexibility of the approach, in the sense that they do not always provide direct access to their inner sub-components or to their inner state.

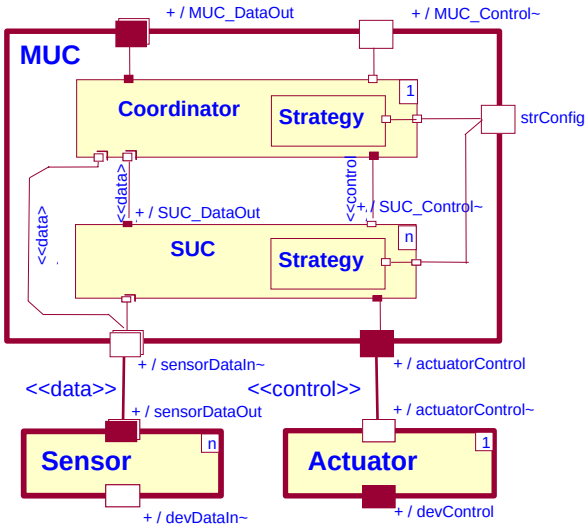

Fig. 3. MUC (Mechanism Unit Controller) and SUC (Simple Unit Controller).

Finally, at the fourth level, the architecture defines the RUC (Robot Unit Controller) component. The RUC component models the control over a whole robot, for example a robot composed of a vehicle with an arm and several interchangeable tools. As Fig. 4 shows, the RUCs are an aggregation of MUCs and a global coordinator that generates the commands for the MUCs and coordinates their actions, according to the orders and the information that it receives and its installed coordination strategy. This strategy is an interchangeable part of the RUC. For example, the **`Strategy`** of a robot composed of a vehicle with a manipulator could be a generalised kinematics solution that contemplates the possibility of moving the vehicle to reach a given target. Like the MUCs, the RUCs are logical components that can become physical components depending on the concrete instantiation. In general, the RUC is quite a complex component that comprises hardware and software components and can expose a wide variety of interfaces depending on the complexity of the controlled system.

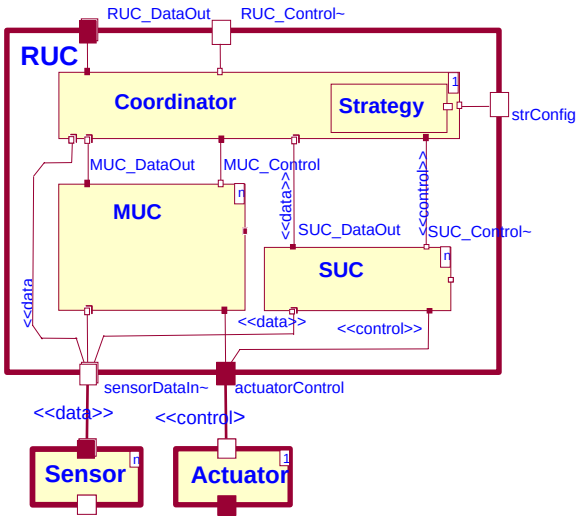

Fig. 4. RUC: Robot Unit Controller.

3.2 *The Intelligence Subsystem (IS)*

It is beyond the scope of this paper to give a detailed explanation of the *IS*, but we do offer some considerations at this point. The *CCAS* is well suited to operator-driven systems and systems where the reactive or autonomous behaviour responds to simple rules that can be added to controllers and coordinators. However, there are systems where the autonomous behaviour is anything but simple. In such cases, the *intelligent component* needs to integrate more information and access more functionality than what is embedded in a given component. The approach adopted here is to *superimpose* "*intelligent*" autonomous behaviour and operator-driven behaviour, and to provide the means of integrating both and resolving the potential conflicts. This approach does not entail any change in the components defined so far, but it does entail new sources of commands for them.

This scheme is similar to the CLARAty architecture (Coupled Layered Architecture for Robotic Autonomy) (Nesnas, 2003) used for Mars rovers. CLARAty distinguishes a *Functional Layer*, where

the components of the system are defined, and a *Decision Layer* that encapsulates the subsystems responsible for planning and executing the missions. Unlike CLARAty, where some autonomous behaviours can be added to the functional layer, in our approach the system's intelligence is completely separate from its functionality because it has been designed for the domain of teleoperated robots rather than autonomous systems, so the conception of the intelligence as another user of the functionality, like the teleoperator, was considered a more important issue.

4. IMPLEMENTATION OF ACROSET ON A PLC AND FIELD-BUS PLATFORM

ACROSET is being used for a real system, the family of robots in EFTCoR project (see section 2). One of the teleoperated robots of the project is in a very advanced development state. An instantiation of ACROSET is being implemented on the control unit of a system composed of a XYZ table holding the cleaning tool. This tool consists of an enclosed nozzle for making the blasting and recovering of residues. The system can be driven by a human operator and it can also perform some autonomous tasks. The XYZ table is supported by a commercial crane whose control is not considered in this instantiation.

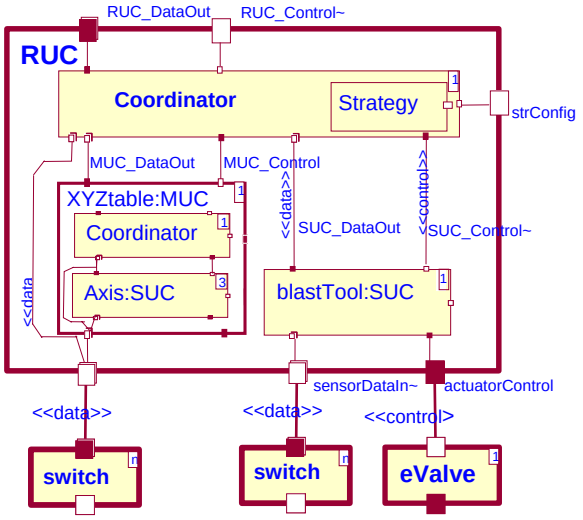

Fig. 5. Components of CCAS in XYZ table Control Unit.

The components integrated in the *Coordination, Control and Abstraction Subsystem (CCAS)* are shown in Fig. 5. The RUC encloses all the functionality required to drive the XYZ table and the tool. The RUC interface offers access not only to the global RUC commands but also to their inner components. The MUC and SUC included in the RUC control the XYZ table and the blasting tool respectively. The MUC coordinates three SUCs, one for each axis of the table. In this case the actuators are logically placed inside the SUCs and are accessed through the SUC interface. This is imposed by the system's hardware architecture. In this case, COTS hardware controllers have been used to control the electrical motors of the XYZ table. Therefore, the hardware that is abstracted is not merely an engine but a complete axis controller. The actuator is hidden to the control unit and the SUC is thus a *"hardware abstraction component"* contained in a MUC. The SUCs that control the axes therefore have a software part and a hardware part. The RUC and the MUC are implemented entirely in software.

Following ACROSET, the intelligence of the system, like the fault and configuration management, is located outside the CCAS. The two are respectively included in the *Intelligence Subsystem (IS)* and the *SMCS*, as shown in Fig. 2. The *real intelligence* included in the *IS* varies considerably from system to system. In this particular case the *IS* interprets a pre-programmed sequence of motions and orders that have been generated by a vision system. The vision system is an external system, running on a PC, that analyses the surface to be cleaned, generates trajectories and delivers them to the *IS*, which in turn delivers them to the UIS and supervises their execution. A human operator can supervise the movements commanded by the vision subsystem and take corrective action by means of a joystick. An arbitrator situated in the *UIS* determines which is controlling the system at all times (note that the *CCAS* only receives orders from one channel coming from the *UIS)*.

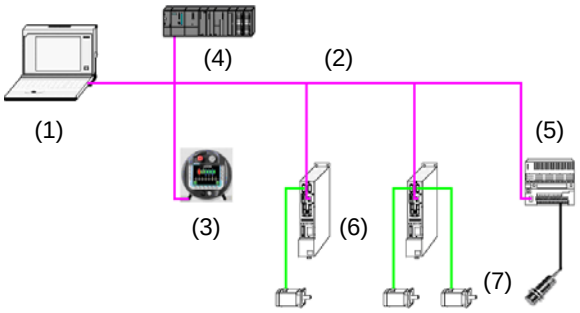

Fig. 6. Hardware architecture

In response to the special industrial requirements of the EFTCoR project, the system has been implemented using a PLC (SIMATIC S7-300 series) and a Field-Bus (PROFIBUS-DP) as shown in Fig. 6. The development environment is STEP 7. Each SUC, MUC and RUC has been translated to PLC Function Blocks (FBs) (SIEMENS, 2002). With the option of FB instantiation in SIMATIC S7-300 series, it is possible to program the PLC with a philosophy that is close to the object-oriented paradigm (each FB acts as a class which can be instantiated). For instance, a generic axis controller (SUC) has been defined (Fig. 7) to create three instances, the controllers (SUCs) for the X, Y and Z axes, each with their particular features. Specifically, most of these instantiations have been done through the use of multi-instances (SIEMENS, 2002) in order

to have more compact components and save PLC Data Blocks (DBs).

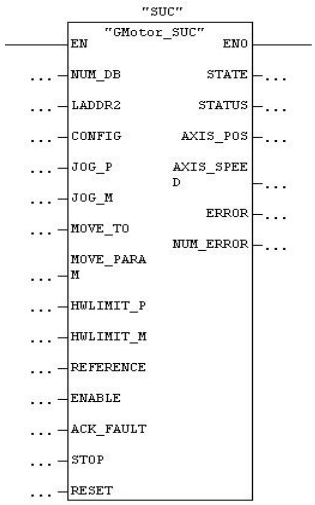

Fig. 7. SUC (Motor Controller) implemented as FB

The devices that compose the control system are the following:

- **PROFIBUS-DP** Field bus **(2** in Fig. 6**)**. It is especially adapted to the communication between automation systems and decentralized outlying stations.
- **Programming PG Unit (1** in Fig. 6**)**, in our case a PC-portable, connected via PROFIBUS with the SIMATIC S7 equipment, thanks to a CP 5512 connection card.
- **SIEMENS MOBILE PANEL 170 (3** in Fig. 6**)**, the operator panel that allows the user to interact with the system. A GUI programmed with the PROTOOL software runs on it.
- **PLC SIMATIC S7-314C-2DP (4** in Fig. 6**).** The control program runs on this CPU. It also integrates a DP interface, and acts as the master of the PROFIBUS-DP field-bus.
- **In/Out Modules (5** in Fig. 6**)**, for the connection of sensors and actuators present in the system. The CPU communicates via PROFIBUS with the decentralized periphery. This method reduces the wires and facilitates the addition of new sensors to the system.

- **SIMODRIVE 611U Regulation Module (6** in **Fig. 6)**, for the control of the 5 servomotors of the system. Each module consists of a 2 axes card that includes two regulation systems working independently.

- **Sensors and Actuators (7** in Fig. 6**),** TELEMECANIQUE inductive proximity sensors can be found in the system (six in the X and Y axes and two in the Z axis, of the series XS1 and XS6); and 5 servomotors SIEMENS 1FK7 (3 for the XYZ table and 2 for the elevation platform) as actuators.

5. CONCLUSIONS AND FUTURE WORKS

The use of a common architecture for a domain or family of systems allows rapid developments and the reuse of components. This paper has presented a reference architecture for the development of teleoperated service robots control units (ACROSET), and also an application in the context of the EFTCoR project that show the ability of ACROSET to cope with the needs and requirements of the system and also the ability to be implemented on a PLC and field-bus platform. The separation of the conventional functionality of the systems (*CCAS*) from the intelligent behaviours greatly facilitates the addition of new functionalities and the maintenance of applications. Perhaps, the main contribution of ACROSET to the current state of the art is the use of a conceptual component oriented approach. This allows the components to be independent of the implementation language or hardware/software partition. This has allowed implementing those components as PLC blocks, not only as objects and classes, e.g. as CLARAty (Nesnas, 2003) does.

ACKNOWLEDGEMENTS

The DSIE wishes to thank the European Union (GROWTH), Spanish Government (CICYT) and the Regional Government of Murcia (Seneca Programmes) for their funding and support: G3RD-CT-00794, TIC2003-07804-C05-02 and PB/5/FS/02.

REFERENCES

Bruyninckx, H., Konincks, B. and Soetens, P. (2002), A Software Framework for Advanced Motion Control. Dpt. of Mechanical Engineering, K.U. Leuven. Belgium. www.orocos.org

Coste-Manière, E. and Simmons, R. (2000), Architecture, the Backbone of Robotic System. *Proc. of the 2000 IEEE international conference on robotics & Automation*, San Francisco, USA.

EFTCoR (2002). Environmental Friendly and Cost-Effective Technology for Coating Removal. V Framework Program GROWTH G3RD-CT-00794.

Hofmeister, C., Nord, R., Soni, D. (2000). "Applied Software Architecture", Addison-Wesley. ISBN 0-201-32571-3. USA.

Iborra, A. , Pastor, J.A., Álvarez, B., Fernández, C. and Fernández-Meroño, J. M. (2003). Robots in Radioactive Environments. *IEEE Robotics and Automation Magazine*, **vol. 10**, no. 4, pp. 12-22.

Nesnas, I. et al (2003). CLARAty: An Architecture for Reusable Robotic Software. Jet Propulsion Laboratory, NASA, Carnegie Mellon University.

Ortiz, F. et al (2000). GOYA: A teleoperated system for blasting applied to ships maintenance. *3rd International Conference on Climbing and Walking Robots*. ISBN 1-86058-268-0

Rational Software Co. (2002), www.rational.com.

Selic, B., Gullekson, G. and Ward, P.T. (1994). *Real-Time Object-Oriented Modeling.* John Wiley and Sons, New York. 1994.

SIEMENS (2002). SIMATIC - Working with STEP 7 5.2. *ref. 6ES7810-4CA06-8BA0*. www.siemens.com.2002.