12

A Reference Control Architecture for Service Robots as applied to a Climbing Vehicle

Francisco Ortiz, Diego Alonso, Juan Pastor, Bárbara Álvarez and Andrés Iborra
Division of Electronics Engineering & Systems - Universidad Politécnica de Cartagena
Spain

1. Introduction

Recent progress in mechatronics, perception and computing is opening up a number of new application domains for robotics, improving the way in which robots perform actions that release the human from dangerous or risky tasks. Nowadays, the field of service robotics is in continuous development, covering more and more application domains, from home to industry, and offering more and more capabilities in a reliable and user-friendly way. One of the new environments where robots are starting to appear is in the shipyard. Developing robots for working in shipyards is very challenging because of both the difficulty of the missions that robots should perform as well as the lack of robotic culture in this kind of industrial facility.

The authors' research group, the DSIE (*Division of Electronics Engineering & Systems*) at the Technical University of Cartagena, has a considerable experience in the development of software applications for teleoperated service robots, mainly for nuclear power plants (Iborra et al., 2003) and in shipyards industry (Fernández et al., 2004). The work presented in this chapter has been carried out in the context of the EFTCoR project *(Environmental Friendly and Cost-Effective Technology for Coating Removal)* (EFTCoR, 2005). The EFTCoR project sought to develop a solution for ships' hulls cleaning and for the retrieval and confinement of the oxide, paint and sea adherences resulting from the cleaning operations. For this purpose, several robots were designed, one of which being a climbing vehicle capable of positioning a grit-blasting tool onto ships' hulls. This chapter describes our experience in the development of the climbing robot and the software architecture designed for its control unit, ACROSET (Control Architecture for Service Teleoperated Robots).

Software architecture is one of the key elements of any robotic system. As technology evolves, it is possible to build systems that are capable of carrying out more complex tasks in more complex environments. But the new robot capabilities demand a great variety of components, both hardware and software, that must interact in diverse ways. Such components must be structured in a way that (1) the robot achieves its global functionality and (2) the system could be easily maintained and updated. The way in which components are organised is described by the architecture of the system. The importance of considering system architecture to handle the inherent complexity of robotic systems is well known (Coste-Manière & Simmons, 2000): overall system complexity can be reduced by dividing it

Source: Bioinspiration and Robotics: Walking and Climbing Robots, Book edited by: Maki K. Habib
ISBN 978-3-902613-15-8, pp. 544, I-Tech, Vienna, Austria, EU, September 2007

into smaller components with well defined abstraction levels and interfaces. The definition of a good architectural framework allows rapid development of systems, maintenance, scalability and reuse of a large variety of components, with concomitant savings in time and money.

As said before, the objectives of this chapter are twofold: to present the climbing vehicle (Lázaro) and the architectural framework used for designing its control unit (ACROSET). This chapter is structured in eight sections. Section two exposes the challenges and special requirements imposed by shipyards to design robots for cleaning ships' hulls. It also includes discussion on the state of the art on climbing robots for ship cleaning and the issues that, in our opinion, can be improved. Our contribution, the Lázaro robot, is described in section three, where two versions of this climbing vehicle are presented. In section four, the importance of software architecture for the development and maintenance of a robot is discussed, including a brief description of the latest frameworks for robotics and the possible contribution of ACROSET to the state of the art. The main characteristics, subsystems, components and design guidelines of ACROSET are presented in section five. The following section explains how this architectural framework has been used to develop the control unit of the climbing robots, and towards the end the chapter, some tests, results, lessons learned and conclusions are presented.

2. Challenges and Requirements to Design a Climbing Robot for Ship Hull Cleaning

2.1 Identifying the Problem

Main ship maintenance operations consist of periodical (every 4-5 years) removal of sea adherences and hull coating and subsequent hull re-painting afterwards. This process is carried out to preserve the hull integrity, guarantee sailing conditions, and to maintain a smooth surface, thereby minimizing fuel consumption, reducing operation costs and atmospheric pollution. Other maintenance operations are scheduled or even delayed to be done while the hull cleaning and re-painting operations are performed. Present technology for hull cleaning (Smith, 1999), mainly open-air grit-blasting or sand-blasting (see Figure 1), is very pollutant and environmentally unsound because residues of the process are thrown directly to the sea; for this reason it is progressively being forbidden in the most sensitive countries with a clear trend to being reduced in the rest until being definitively forbidden. In order to avoid residues emissions, grit blasting is being partially substituted by ultra-high pressure water blasting (Goldie - b, 1999). These systems avoid the pre-water cleaning required for hull desalinization as used with grit blasting; but, as reported by paint suppliers and ship owners, they do not show as good a performance and quality surface preparation as grit blasting systems achieve. This fact is causing that more and more ship owner move to shipyards where the open grit blasting is still allowed (Middle East, Far East, Korea and China), with loss of ship repair work in European yards (where open grit blasting is being prohibited).

Regardless of the technology used (grit or water) cleaning operations can be classified into two types: full blasting and spotting. Full blasting is the cleaning of a small number of very large areas (ultimately only one area consisting of the whole hull), while spotting is the cleaning of a very large number of very small areas (spots) scattered along the hull.

Figure 1. Present method for hull blasting: a dangerous manual task

In most cases the operations are carried out by hand. Different circumstances, such as very complex work environment inside the docks, vessels of many different shapes and size (see Figure 2), etc. make it difficult to automate maintenance operations. Despite these problems, several robotic solutions exist using both grit and water technologies. Solutions using grit based technology are usually restricted to full blasting in vertical surfaces by means of heavy turbines supported and positioned by large cranes. Water based solutions are lighter and can be positioned by relatively small vehicles. These vehicles can reach all hull areas (vertical and shaped) and perform full blasting and spotting. However, until now, water blasting has been more expensive and has not achieved the performance and quality of surface preparation that grit blasting systems offer.

Figure 2. Different requirements for the cleaning devices: bow, bottoms and vertical surfaces

The paragraphs above show the context in which the EFTCoR project emerges. The EFTCoR project (EFTCoR, 2002) is part of the European Industry current effort to introduce environmentally friendly ship maintenance. Partners of the project are companies, shipyards and research institutions from different European countries. This project addressed the development of a family of robots for grit-blasting, whose mission was to retrieve and confine the paint, oxide and adherences from ship hulls and recycle the blasting material. Our research group had the responsibility of developing the robotic devices.

2.2 Requirements of the Application Domain

The use of robotic devices in shipyards is difficult due to the characteristics of the working environment and the nature of the maintenance operations that have to be carried out. First of all, robotic devices for hull cleaning should achieve the following requirements:

- The blasting material should provide the required surface quality (SA 2½) for painting afterwards.

- Dust emissions should be eliminated or dramatically reduced.
- The cost and performance should be as good as that obtained with the manual operation.

Besides these general requirements, the application domain exhibits other characteristics that make the development of a single general-purpose robotic system capable of performing all the difficult tasks. This is due to:

- The dimensions and shapes of hulls are very different from one ship to another.
- The different areas of a given hull (bow, bottoms and vertical surfaces as Figure 2 shows) impose very different working conditions for robotic devices.
- The differences between the working areas that exist in different shipyards (see Figure 2) require the development of different solutions.
- The operational differences between full blasting and spotting. Full blasting demands devices capable of positioning large cleaning heads along large hull surfaces, while spotting (cleaning of isolated but very numerous points and small surfaces) demands devices capable of precise positioning and a very fast a small cleaning head.
- The possibility of considering other hull maintenance operations like fresh water washing and painting.
- The different cultures and business priorities of the different shipyards.

Table 1 summarises, as an example, the main requirements of two different shipyards (shipyards names are not shown for confidentiality) to show the differences of the requirements for the development of the EFTCoR systems.

Requirement	Shipyard 1	Shipyard 2
Cost	No more than current costs including pay-offs	Equal or better than the operative costs obtained with conventional blasting. Cost with abrasives should be drastically decreased.
Performance	5 m²/man-hour. Efficiency of the nozzle 10 m²/hour.	
Environmental constraints.	Reduction to dust emissions (at least 70%)	The amount of used abrasive to dispose should be reduced drastically.
Working area	Synchrolift. Ships very densely positioned. Removal of obstacles in working area supposes an organizational problem.	Very large dry docks, but available space limited.
Adaptable to full blasting and spot blasting	Spot work makes up 80% of the work.	Spot makes up 35% of the work and 48% of the blasting business.
Surface praparation	SA 2 1/2 (ISO 8501-1)	SA 2 1/2 (ISO 8501-1)
Adaptable to different hull maintenance operations	Water washing, painting	Water washing, painting
Adaptable to different types of hull and to different areas of the hull	125 m length 25 m depth 23 m breadth Adaptable to all hulls and to all surfaces	Very large tankers
Usability	Easy to operate	Easy to operate
Possibility of automation	Yes	Yes
Others	On line control of quality	Easy to transport and assemble

Table 1. Shipyards global requirements

2.3 State of the Art of Climbing Robots for Ships Cleaning

This section presents the most relevant climbing robots for hull cleaning. All of them use ultra-high pressure water. To our knowledge, there are no commercial climbing robots that use grit as an abrasive. Ultra-high pressure water blasting technology uses a head which contains a number of small rotating nozzles which send out water at ultra-high pressure, between 700 and 2500 bar. Unlike the open-air grit blasting cleaning system, ultra-high pressure water systems normally have a vacuum system for retrieving, filtering, separating and storing the residues produced during the cleaning process. The head with the cleaning tool, which is usually teleoperated, is normally fixed to the hull by means of permanent magnets.

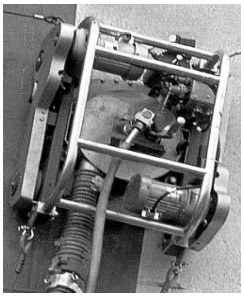
Figure 3 shows an excerpt of the cleaning systems that currently uses ultra-high water pressure. Figure 3-a shows the cleaning operation done manually by human operators while the rest of the pictures illustrate some of the cleaning robots currently available in the marketplace. Among them, the most effective and widely used is the Ultrastrip M3500, a system developed by Ultrastrip Systems (Ultrastrip, 2007) (see Figure 3 c and d). This teleoperated vehicle is built in aluminium and titanium and is fixed to the hull surface by combining a magnetic head and a vacuum system. Table 2 shows the main technical features of the Ultrastrip M3500 system.

It is also worth mentioning the Hydro-Crawler system developed by Dans Vandteknik (Hydro-Crawler, 2006) (see Figure 3 e and f); the HydroCat system of Flow International Corporation (Flowcorp, 2005) (see Figure 3-b); and the Octopus system of Cybernetix (Octopus, 2005) (see Figure 3 g and h).

Technical Specifications	
Maximum working pressure (bar)	3000
Working speed (mm/s)	510
Weight (Kg)	222
Cleaning head width (mm)	380
Clearing ratio (m2/h)	46 a 268
Dimensions: height x width x length (mm)	560 x 690 x 1710
Control	Teleoperated/Joystick

Table 2. Technical specifications of ULTRASTRIP M3500

a) Manual tools

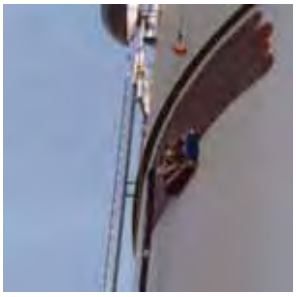


b) Flow HydroCat

c) Ultrastrip

d) Ultrastrip working

e) Hydro-Crawler



f) Hydro-Crawler working

g) Octopus

h) Operating Octopus

Figure 3. Ultra-high pressure water cleaning systems

2.4 Possible Contributions to the State of the Art

All the systems presented in this section are more or less similar in the sense that all of them use a vehicle with permanent magnets or vacuum devices to adhere to the hull of the ship and have a cleaning head that uses ultra-high pressure water technology to remove the paint, oxide and sea adherences from it.

The major disadvantage of all these vehicles and cleaning systems is the quality of the surface once the cleaning operation has been performed. Ultra-high pressure water blasting offers a surface quality of SA 1½ (ISO 8501-1), which is lower than the quality obtained using abrasives, e.g. grit, which achieves SA 2½, which in turn affects the paint adherence. The best known and most efficient of water-based systems, the mentioned Ultrastrip, presents a cost that prevents its use in most of the shipyards in Southern Europe.

The EFTCoR project tries to solve these problems by developing a family of low-cost systems to perform the cleaning operations in the different parts of the hull while achieving the adequate surface quality. One of the systems developed in the EFTCoR project was a climbing vehicle using abrasive (grit) for blasting instead of hydro-blasting to fulfil the requirements of surface quality imposed by the shipyards.

3. The Lázaro vehicle in the context of the EFTCoR Project

3.1 The EFTCoR devices

The requirements of cleaning operations in shipyards as exposed in section 2 show the difficulty of designing a general purpose system, or even defining a common body of general requirements that could be applied to all systems. For this reason, the EFTCoR project proposes different solutions for the different problems:

- Teleoperated or semi-automated cranes
  - For full blasting: a primary positioning system to position heavy burdens (turbines projecting grit) along large surfaces (the whole ship hull). This primary system is a special crane adapted to carry turbines or the secondary system (see Figure 4).
  - For spotting, a secondary positioning system, which can be mounted on the primary, is capable of positioning a light cleaning head with the precision required to move quickly from one spot to another over small surfaces (4 to 10 m^2) (Figure 4).

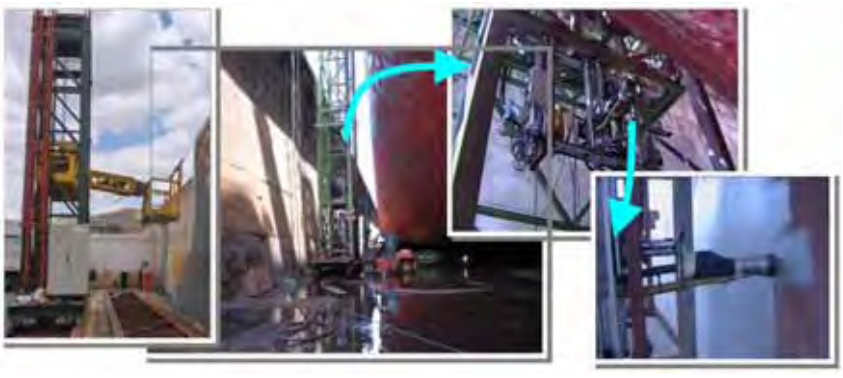


Figure 4. Primary system (special crane), secondary system (XYZ table) and tool

- Climbing vehicles (see Figure 5 and Figure 6), provided with a cleaning head, and that have been developed to reach those areas that were unreachable with a reasonable combination of primary and secondary positioning systems. The vehicle can be used all over the ship hull and can perform spotting tasks and full blasting. Although the performance for full blasting is considerably lower than that of using a turbine supported by a crane (primary system plus tool configuration), it represents a global solution suitable for all shapes that can be chosen depending on the shipyards' global requirements for a given job.

Two different versions of the vehicle will be presented: an experimental version (Lázaro I), where the execution platform is an on-board embedded PC and a pre-industrial version (Lázaro II) based on commercial motor drivers by SIEMENS.

3.2 Lázaro I: Experimental Prototype

The objectives of this prototype were (1) to build a vehicle capable of moving along the hull with a grit tool, (2) to test the execution platform and (3) to serve as a first and simple example of the application of ACROSET.

The mechanical structure of the vehicle is presented in Figure 5. It is a caterpillar vehicle capable of climbing along a hull thanks to permanent magnets that holds a grit-nozzle. The vehicle can be driven by a human operator and also performs some autonomous tasks, such as obstacle avoidance and simple pre-programmed sequences. The execution platform is an on-board embedded PC with a PC/104 expansion bus. It is based on an Intel, ultra low voltage Celeron microprocessor. The PC/104 bus is a widely used industrial standard with many advantages, such as vibration-resistance, modularity, mechanical robustness, low power consumption, etc., so it is an excellent bus for embedded systems. The expansion system is formed by an analog and digital I/O board featuring 8 analog inputs, 4 analog outputs, 3 timer/counter and 24 general purpose digital lines, and a PCMCIA expansion interface.

Figure 5. Lázaro I, CAD model and built prototype

The Lázaro I robot has two servomotors for controlling the wheels and one more for the orientation of the nozzle. The control of each servomotor is performed with the help of incremental encoders. Besides this, the robot also has a ring of bumpers and infrared sensors to stop in case it nears an obstacle or collides with one. The control algorithm is quite simple and it is performed by the software in the embedded PC. The chosen operating system is Real-Time Linux (Barbanov, 1997), which allows the possibility of having a real-time

application running while retaining all the power of a Linux distribution (though with some restrictions) underneath. This executing platform has been chosen in order to have as much flexibility as possible in the test and modification of the software architecture, including variations in control strategies, number of control threads, etc.

3.3 Lázaro II: Pre-industrial Prototype

Once Lázaro I had accomplished its objectives, an industrial enhanced version, the Lázaro II, was developed using as many COTS as possible in order to design a robust hardware and software platform (see Figure 6). Although the flexibility of this second prototype is lower than Lázaro I, it fulfills the industrial requirements of the partners.

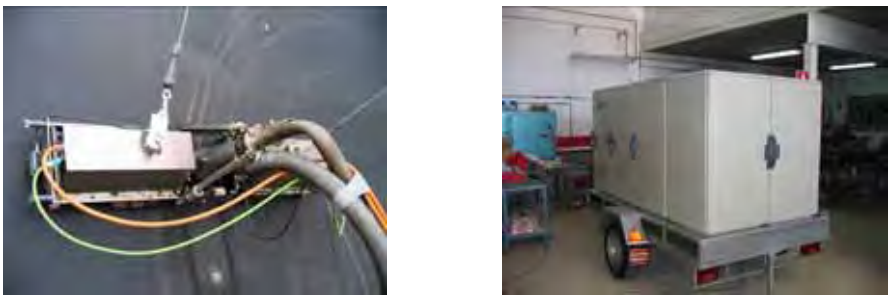


Figure 6. Lazaro II and portable control unit with enough space to place the robot inside

The Lázaro II has two servomotors from SIEMENS, two collision sensors and two inclinometers. The control and power units are placed outside the vehicle as Figure 6 shows. The control unit is formed by a SIEMENS programmable controller (PLC) and two drivers for the motors. It can be remotely operated by a human operator for spotting individual areas of the hull and it can also perform semi-autonomous full-blasting with the supervision of the operator. The cleaning head is formed by a nozzle for throwing grit and a vacuum bell to absorb all the residues.

NAVANTIA - Cartagena			
Blasting method		**Full (m^2/hour)**	**Spot (m^2/hour)**
Efficiency	Manual	25	17,5
	Lazaro II	**24**	**22,3**
	Cranes	**180**	**35**

Table 3. Comparison between performance of manual cleaning and automatic cleaning

The performance of each operation can be consulted in Table 3. This table shows the data corresponding to the shipyard where the tests were performed, Navantia-Cartagena. It

compares the data available in the shipyard with measurements made by the development team in several tests. In our opinion these results could be extrapolated and applied to most shipyards. Table 3 compares the efficiency corresponding to full and spot blasting operations performed manually, with those where the Lazaro II and the EFTCoR cranes were used. In the worst cases, efficiency has been maintained and in some cases it has been increased. Even in the cases when efficiency or total execution is similar to the manual operation, the advantage of having a residues retrieving and recycling system supposes a strategic advance due to the clear trend of European regulations forbidding environmentally costly practices.

4. Architectural Frameworks for Robotic Systems Development

As mentioned before, robotic systems comprise hardware and software elements that interact in complex and diverse ways. Architecture handles the inherent complexity of robotic systems by dividing it into smaller components with well defined abstraction levels and interfaces. When trying to define the software architecture for the EFTCoR devices, some requirements must be kept in mind:

- High variability of functionality and physical characteristics.
- Different combinations of vehicles, manipulators and tools.
- A large variety of execution infrastructures, including different kinds of processors, communication links and human machine interfaces.
- A large variety of sensors and actuators.
- Different kinds of control algorithms, from very simple reactive actions to extremely complex navigation strategies.
- Different degrees of autonomy, from operator-driven systems to semi-autonomous robots.

These requirements are common in the service robotics domain as they cover a broad range of mechanisms that carry out different activities in hostile environments. Usually, these systems perform a small number of highly specialized tasks. Considering all the sources of variability mentioned above, it is very difficult to design a single architecture flexible enough to deal with such heterogeneity. However, despite all these differences, such robotic systems have many common requirements in their definition and many common components, both logical and physical, in their implementation. Therefore, it should be possible to simplify the development of service robots by defining a flexible and extensible architectural framework to design systems with different requirements but sharing some characteristics. For the purposes of the EFTCoR project we considered that such an architectural framework should be devised according the following design goals:

- The framework should not impose a concrete architecture, but allow defining different architectures (different interactions and constraints) depending on the concrete application requirements.
- It should be possible to reuse components in systems with different architectures. This implies that a clear distinction should exist between the components and their interaction patterns.
- The implementation of components may be software or hardware; it is highly advisable that such components are COTS components.

- It should be possible to integrate "intelligence," or to interoperate with "intelligent systems".

Other robotic-framework developers, such as (Brooks et al, 2005), offer a more complete list of requirements, but for our purposes those listed above are enough.

4.1 State of the Art of Robotic Component Frameworks

There have been numerous efforts to provide developers with component frameworks to ease the development of robotic systems. Among these frameworks it is possible to highlight the following: OROCOS (Bruyninckx et al., 2002), CLARAty (Volpe et al., 2001), MCA (Scholl et al., 2001), ORCA (Brooks et al., 2005), CARMEN (Montemerlo et al., 2003) and PLAYER (Vaughan et al., 2003). All of them make very valuable contributions that simplify the development of these systems.

CLARAty (Coupled Layer Architecture for Robotic Autonomy) gives a very valuable global solution that considers low and high level issues (from architectural design to implementation), in a way that intelligent elements can be integrated where required. To our knowledge, the main drawback of CLARAty is that it is limited to the use of object-oriented technology. Object assembly depends upon the object implementation, not merely upon the object interface, significantly restricting the way in which object can be used as components, as it is explained in the following section.

The frameworks OROCOS, MCA and ORCA are component-oriented although their components rely on object-oriented technologies. As such, designers manage components as design units instead of objects. In addition, OROCOS proposes architecture-neutral components, similar to a library of components to build motion controllers, and MCA proposes a common software platform with different modules that can be organized and compiled to generate the robot control unit. However, OROCOS and MCA overly depend on a given infrastructure (specifically Linux and C++ language). In general, the component approach involves choosing both a given component model and a certain execution infrastructure linked to such a model. This implies that components are not exchangeable from one framework to another. ORCA relies on a middleware to broaden the number of execution platforms and programming languages. In the field of mobile robots, CARMEN and Player provide repositories of components and an infrastructure where such components can be deployed. They have recently been linked to the C++ language and Linux operating system.

4.2 Object technology versus components

When it comes to implementing a software architecture neither the object-oriented paradigm nor the modularity achieved by packaging functionality are usually enough to successfully achieve the objectives listed above. An object cannot be seen as a real component because the required services are not part of the specification of the object, rather they are scattered through the object implementation (Luckham, 1995). Component-Based Software Engineering (CBD) (Szyperski, 2002) aims at shifting the emphasis in system development from programming to composing, building software systems from a mixture of off-the-shelf and custom-built components. A component-oriented approach must assume:

- The system is built by composing and linking components using connectors.

- A component is defined in terms of its interfaces, which include both the required and provided services. These interfaces are the only way in which components can communicate with each other.
- Components should be interchangeable and can also be distributed among different computation units. Connectors mediate between components while at the same time hiding the communications that components make between themselves, for our purposes this leads us to consider connectors as a special type of component.

Most CBD approaches consider that components *should be* binaries units of deployment. However, we prefer to consider that they *could be* binaries units, but also design units, provided that they (1) encapsulate behaviour and data; (2) provide and require functionalities by means of ports; and (3) are subject to composition. Perhaps the main contribution of a component based paradigm is that it effectively allows the reuse of the same components across different architectures, even if they interact in different ways (using different connectors).

4.3 Possible contributions to the state of the art

Current component frameworks for robotic applications generally impose a concrete programming language and execution platform. The use of a middleware layer allows some of these frameworks to broaden the number of potential execution platforms, but again, in some situations, the middleware itself may not be compatible with the application requirements. In fact, this was one of the problems we faced when developing the EFTCoR family of robots. It would be preferable to be able to define components that are independent of both system architecture and execution platform, but that can simultaneously be (1) used to define different architectures and (2) be translated into concrete components executable in a given platform. This is the idea behind the architectural framework that we defined and implemented for the specification and development of the control software of EFTCoR devices, ACROSET (Control Architecture for Teleoperated Service Robots).

ACROSET relies on the abstract concepts of component, port and connector, offering a way to reuse the same components in very different systems by separating the components from their interaction patterns. ACROSET provides a common framework of abstract components which can be implemented in different ways (integrating software and hardware components, and even COTS), and running in different execution platforms, in order to develop teleoperated robots with very diverse behaviours. The abstract components could be instantiated to concrete components, implementing them as a combination of C++ objects, PLC function blocks, Ada packages or interfaces to COTS components (hardware or software), without having to be linked to a given infrastructure. In that sense, ACROSET can be defined as an abstract component framework which is platform independent.

However, although the capacity offered by ACROSET for describing the robotic systems architecture is valuable, the manual translation of the ACROSET abstract components into concrete, platform specific components is a difficult and error prone task. So, ACROSET will only show its full potential if we are able to find a way to automatically translate abstract components into concrete components. The adoption of the MDE (Model Driven Engineering) (Kent, 2002) approach is a key step to achieving this goal. This approach is in concordance with the current trends in software development, e.g. the OMG's (Object Management Group, www.omg.org) initiative MDA (Model Driven Approach) is a very

promising alternative to the traditional software development because it proposes model transformation as the central idea of the proposal and the separation between specification and implementation as its major claim. Using the MDE approach allow us to use the *ACROSET* abstract components to specify the architecture of different robots, while automatic model transformation will keep them synchronised with the implementation. Moreover, it is desirable (and in our opinion possible) to define different transformations to obtain implementation components according to the most suitable robotic frameworks.

5. ACROSET: Reference Control Architectural Framework for Teleoperated Service Robots

ACROSET comprises a reference architecture and an abstract component framework which allows the definition of different architectures in a platform-independent way. In addition, it proposes a set of subsystems to organize the functionality of the whole system. These subsystems were defined following the ABD method (Bachmann et al., 2001), which helps in choosing an architectural option to fulfil the given requirements. The subsystems defined by ACROSET (shown in Figure 7) are the following:

- The Coordination, Control and Abstraction Subsystem (CCAS).
- The Intelligence Subsystem (IS).
- User Interface Subsystem (UIS).
- Safety, Management and Configuration Subsystem (SMCS).

A detailed explanation of these subsystems can be found in (Álvarez et al, 2006). In this section we will try to give an overview of them, especially of the CCAS.

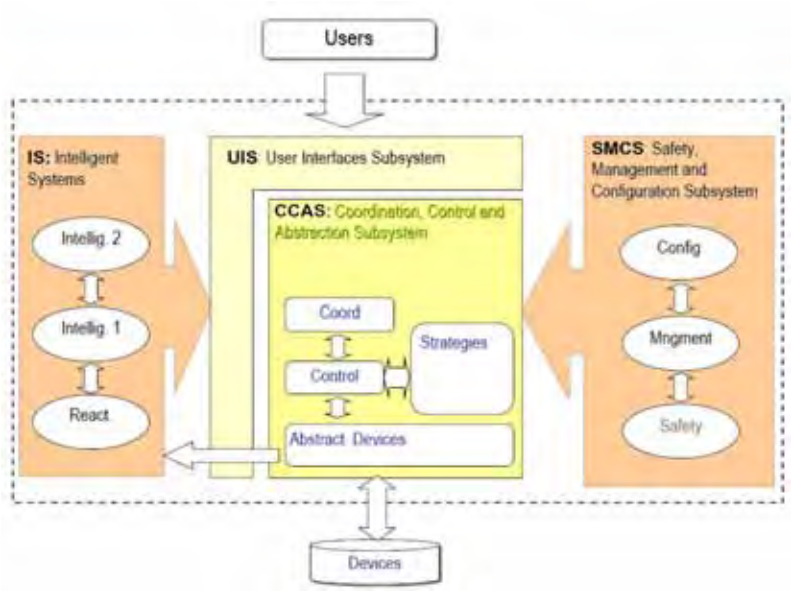


Figure 7. An overview of ACROSET subsystems

The CCAS subsystem abstracts and encapsulates the functionality of the system's physical devices. This subsystem breaks down into a hierarchy of control components that model the different control loops inside a robot. The (abstract) components can be finally implemented in either software or hardware, but all the components of the CCAS and their relationships are independent of the final implementation. Thus, as section 6 will show, the same (abstract) architecture can be reused in different platforms.
The Intelligence Subsystem (IS) allows the integration of components that perform (semi-) autonomous operations and act as another user of the CCAS functionality. The CCAS is well suited for operator-driven systems and systems where the reactive or autonomous behaviour responds to simple rules that can be added to CCAS. However, there are systems where the autonomous behaviour is anything but simple. In such cases, the intelligent component needs to integrate more information than that which is embedded in a given component. The approach adopted in ACROSET is to superimpose "intelligent" autonomous behaviour and operator-driven behaviour, and to provide the means to integrate both and resolve the potential conflicts by means of "arbitration" components (which can also be considered complex connectors). This separation between intelligence and functionality enhances the modifiability and adaptability of the system to new missions and behaviours, but compel us to define a subsystem that mediates between the intelligence subsystems and functionality provided by CCAS, the UIS.
The User Interface Subsystem (UIS) is intended to interpret, combine and arbitrate between orders that may come simultaneously from different users of the CCAS. These users can be human operators or the "intelligent subsystems" of the IS. The CCAS does not concern itself with the source of the order. In the simplest systems, the UIS simply separates the control logic from the user interfaces facilitating the addition and the change of man-machine interfaces. In the most complex cases the UIS includes special components, that we call *arbitrators*, which merge commands coming from several sources following different strategies (to select the right source depending on the control mode, merge behaviours, etc.) and provide a unique command to the CCAS components that remain unchanged.
The Safety, Management and Configuration Subsystem (SMCS) manages and configures the application and separates the functionality *per se* from the monitoring of such functionality. The SMCS is connected directly to CCAS without the mediation of the UIS.

5.1 Components of the CCAS

The Coordination, Control and Abstraction Subsystem (CCAS) comprises a set of components that encapsulate the functionality of the control unit of a robot. They are defined in four levels of granularity:

- Hardware Abstraction Layer.
- SCs: Simple Controllers.
- MCs: Mechanisms Controllers.
- RCs: Robot Controllers.

A very simple CCAS is shown in Fig. 8. The notation used makes explicit the components, ports and connectors and it is inspired by the 4 views of Hofmeister (Hofmeister et al., 2000) and ROOM (Selic et al., 1994).
The simplest components modelled by ACROSET are sensors and actuators, which are encapsulated in the Hardware Abstraction Layer. This layer abstracts the main

characteristics of the hardware of the robot and exposes a set of ports and interfaces to the rest of the components of the CCAS so they can easily use the hardware of the robot.
SC components model the control over a single actuator and offer, through the use of the *Strategy* pattern (Gamma et al., 1995), the possibility of changing the control algorithm at run-time; for instance, the strategy of a given joint may be a traditional control (PID) or may be changed for a fuzzy logic strategy. SCs usually need to accomplish hard real-time requirements and are therefore generally implemented in hardware. In this case, the software SC component acts as a mere proxy of the hardware one.
MC components model the control over a whole mechanism (vehicle, manipulator or end effectors). MCs are logical entities composed of an aggregation of SCs and a Coordinator, which is responsible for coordinating the SCs. The coordination strategy is also an interchangeable part of the MC. For instance, if the MC controls a manipulator its strategy may be a particular solution for its inverse kinematics. Although ACROSET defines MCs as relational aggregates, they can actually become a component (hardware or software) when the architecture is instantiated to develop a concrete system. In fact, it is common that most of the functionality of a MC is provided by a commercial motion control card. When COTS are used the implementation should *bridge* the abstract interfaces of the abstract MC to the actual interfaces of the concrete COTS. Besides, it could be necessary some re-engineering depending on the limitations of the COTS interface.

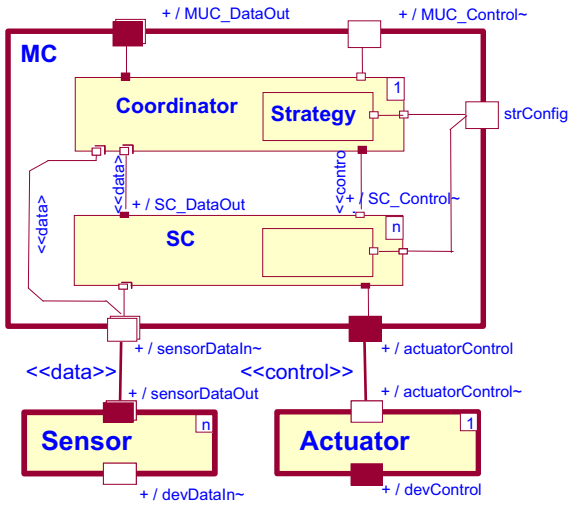


Figure 8. MC and SC over 1 actuator and N sensors

Finally, the architecture defines the RC (Robot Controller) component. RCs model the control over a whole robot, for example, a robot composed of a vehicle with a manipulator and several interchangeable tools. RCs are an aggregation of MCs and a global coordinator. In general, RCs are complex components that comprises hardware and software components and can expose a wide variety of interfaces, depending on the complexity of the controlled system.
Although the CCAS seems to follow a classical hierarchical organization, several innovative concepts have been incorporated, which mainly contribute to increasing the flexibility of the

implementation and the reuse of the adopted approach. The use of components, ports and connectors allow the change of the interaction infrastructure and protocols, and facilitates the distribution of the control application in different processing nodes. These changes may imply the modification of ports and connectors but not the modification of components once implemented, which increases their reuse.

6. Developing the Control Architecture for Lázaro using ACROSET

This section presents the abstract architecture of the Lázaro climbing vehicles and the two platform-specific implementations done for both (see sections 3.2 and 3.3). In this way, this section illustrates the flexibility offered by ACROSET since, although the software and hardware platforms are quite different, both vehicles share almost the same architecture, which is shown in Fig. 9 and Fig. 10 (only the CCAS subsystem is depicted).

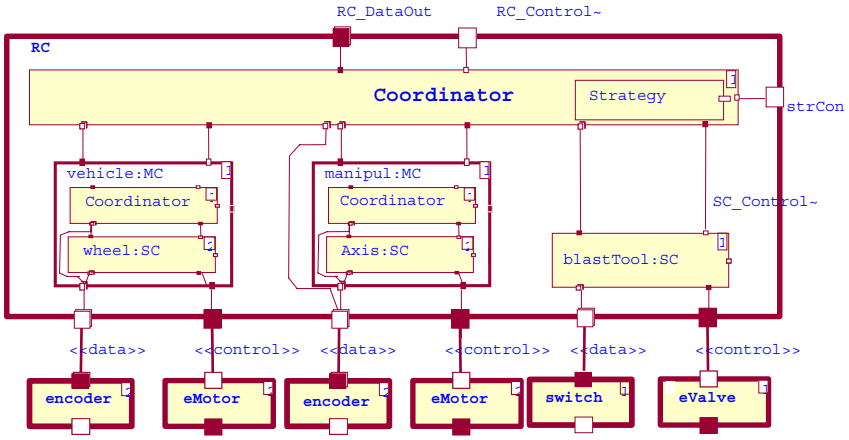


Figure 9. Components of CCAS for Lázaro I control unit

The Lazaro I architecture can be seen in Fig.9. Two different MCs have been included: one to control the vehicle and another to control the manipulator. The first contains one SC to control each of the electrical motors that move the vehicle. The manipulator MC coordinates two SCs, one for each manipulator axis. The CCAS also includes an SC for controlling the electro-valve associated with the blasting nozzle.

The Lazaro II architecture can be seen in Fig. 10. The vehicle controller is modelled by a MC. It contains two SCs to control each of the electrical motors that move the vehicle. The manipulator MC coordinates two SCs, one for each manipulator axis. The vehicle uses a tool that consists of an enclosed nozzle for making the blasting and a vacuum belt for recovering the residues. In this case there is not a manipulator.

Both architectures share the abstract components corresponding to the vehicle and blasting tool, although as is shown in the following sections the concrete components are completely different. The RC coordinator is slightly more complex in the case of Lazaro I, since it has to take into account the manipulator.

Though it is not shown in Fig.9 nor in Fig. 10, the architecture of both versions of Lázaro include an IS that provides two "intelligent behaviours": obstacle avoidance and execution of pre-programmed sequences of motions. The components of the IS that implement these

behaviours obtain the information they need from the vehicle sensors and generate commands to the CCAS. Integration between these commands and the operator commands is resolved by an arbitrator in the UIS.

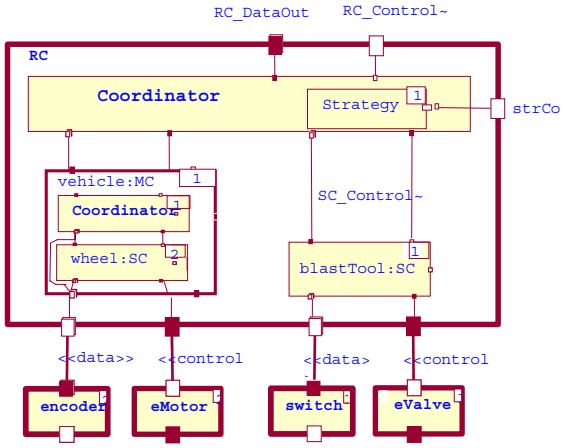


Figure 10. Components of CCAS for Lázaro II control unit

6.1 Lázaro I Implementation of the Architecture

The implementation of the CCAS for the Lázaro I was carried out in the Ada'95 programming language following the object-oriented paradigm. Components, ports and connectors have to be translated into classes and objects. An example of component implementation is presented below in Figure 11. The *Motor_SUC* class contains the ports showed in Figure 11 with stereotypes «*InPort*» and «*OutPort*», to get data (*Data*) or produce control (*Ctrl*) and to configure the SC (*Config*). Ports belong to the component and they are created and destroyed with it, they therefore have a composite relation.

The operations offered by the control ports match with the events sent by other components to the SC. Besides ports, class *Motor_SUC* contains the interchangeable *ControlStrategy* object (the control algorithm). The rest of components of the instantiation of ACROSET for Lázaro I have been built in a similar manner, extending their interfaces to the needs of the system. Notice that the SC interface remains similar in every component thanks to the method *processCommand()*, which processes any incoming event in its particular control inport. The implementation of *processCommand()* is different for each SC, MC and RC.

To end the Ada-95 *interpretation* of the architecture, the objects previously identified are mapped onto an execution architecture, where concurrent tasks (threads), task interfaces and interconnections are defined. The driving forces behind the decisions for designing the execution architecture view are performance, distribution requirements and the runtime platform, which includes the underlying hardware and software platforms. Too many threads in a system can unnecessarily increase its complexity because of greater inter-task communication and synchronisation needs, and can increase the overhead of the system because of additional context switching. The system designer has to make tradeoffs between introducing enough threads to simplify and clarify the design while keeping their overall number low so as not to overload the system.

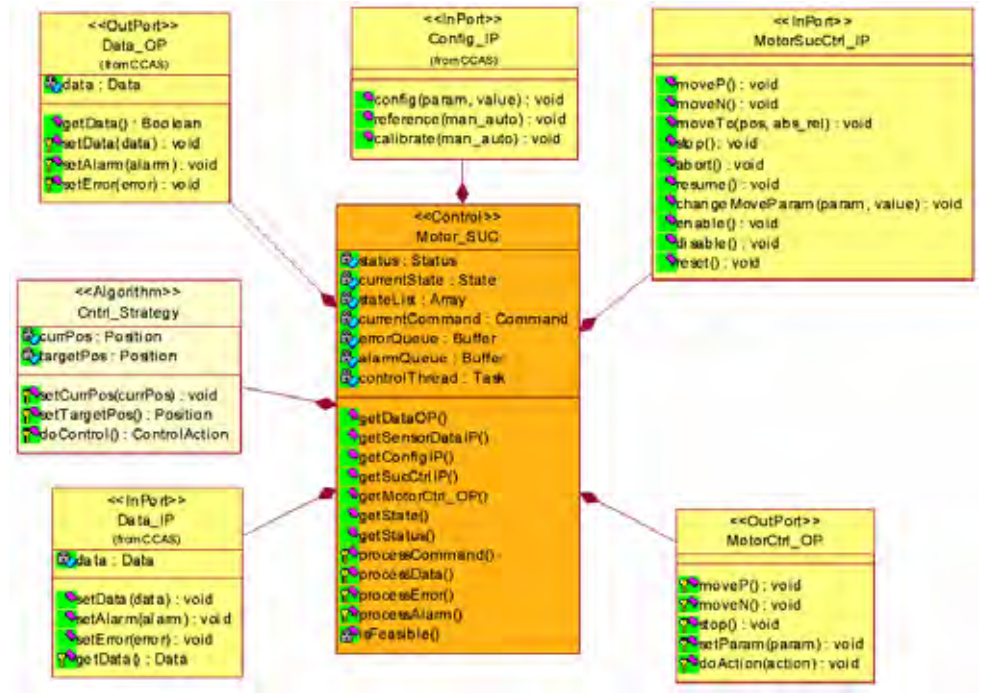


Figure 11. Implementation of a Motor SUC

6.2 Lázaro II Implementation of the 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. 12-a. The development environment is STEP 7 (SIEMENS, 2002). Each SC, MC and RC has been translated to PLC Function Blocks (FBs) (SIEMENS, 2002) as shown in Fig. 12-b. 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 (SC) has been defined to create two instances, the controllers (SCs) for every wheel, although in this case, both wheels are identical, the SC can be adapted to different wheels or axes simply by changing the associated DB (PLC Data Blocks).

Compared to the implementation of Lázaro I, it is clear that the translation of abstract component of ACROSET into concrete components in Lázaro II is totally different. It is important to state that even though the execution platform was so distinct from Lázaro I the design of the architecture for the second prototype was executed very rapidly starting as it did with the architecture of Lázaro I**.** The most difficult process was the translation of the mentioned ACROSET abstract components into concrete, platform specific components because it had to be carried out manually. For that very reason, we are currently researching an approach using Model-Driven Engineering (MDE) (Schmidt, 2006) in order to obtain transformations from model to text (code) that could lead to automated code generation, as it is explained in section 7.2.

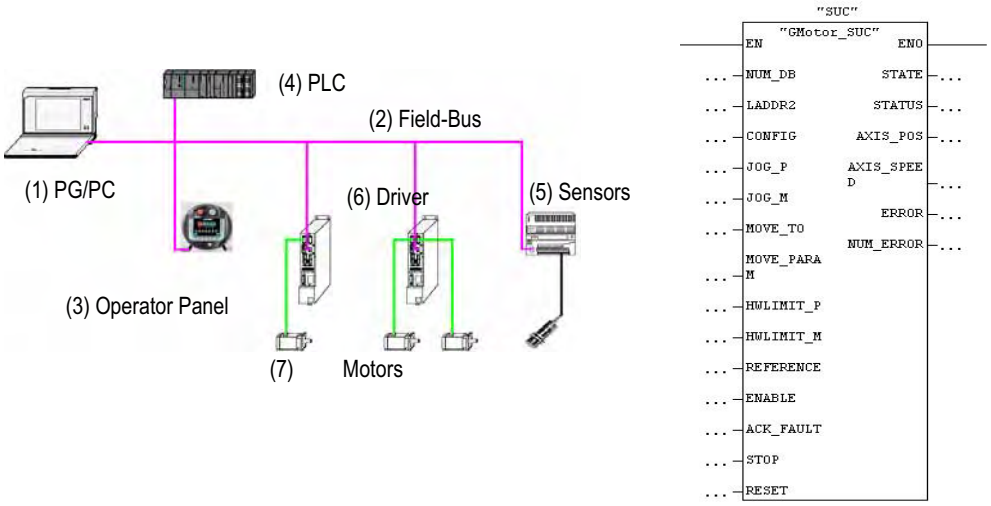


Figure 12. (a) Hardware architecture (b) SC (Motor Controller) implemented as a FB

7. Conclusions, Lessons Learned and Future Research

7.1 Conclusions and Lessons Learned

In this chapter we have described our experiences using an architectural framework in the development of robotic applications, with discussion of the importance of system architecture to handle the inherent complexity of robotic systems. Among the robots developed in the EFTCoR project, two versions of the climbing vehicle Lázaro have been described, starting from the special requirements of the shipyards to develop cleaning systems that can free human operators from those dangerous tasks, and, at the same time, minimizing the emissions of pollutants into the environment.

Among the many lessons learned in the development of such software architectures and frameworks it is important to highlight two in particular: (1) it is not feasible (at least not for us) to define a software architecture sufficiently generic to be adapted to the entire target domain, and therefore (2) it is more useful to follow approaches that allow developers to reuse components in different architectures. This is just what Component Based Development (CBD) and component frameworks propose.

Current component frameworks for robotic applications generally impose a concrete programming language and execution platform that may or may not appropriate for any given application, as described in section 4 . It would be desirable to be able to define components that are independent of both system architecture and execution platforms, and this is the idea behind ACROSET abstract components. ACROSET, as an abstract component framework, tries to overcome the difficulties found in the state of the art: (1) limitations of object-oriented technology; and (2) lack of portability of components from one framework to another.

ACROSET as a reference architecture guides the developer in the process of building a concrete architecture, guarantying that quality requirements are fulfilled as well as being

flexible enough to combine different components inside these subsystems. In addition it does not restrict the level of granularity that must be reached in every implementation.
With regard to the implementation of the architecture into different execution platform, section six demonstrated the way in which a similar definition of abstract components for two prototypes can be translated into very different implementations. ACROSET components are defined at a high enough level of abstraction to allow different implementations on different execution platforms, programming languages or hardware/software partitions (software objects, PLC function blocks, hardware components, COTS, etc). It is even relatively easy to distribute some software components to different processing nodes keeping the same conceptual model of the architecture, by simply changing the connectors between such components.

7.2 Future Research

Although the capacity offered by ACROSET for describing the robotic systems architecture has been very valuable, the translation of the ACROSET abstract components into concrete, platform specific components has been a difficult and error prone task. Therefore, after this experience, we believe that an approach like ACROSET will only show its full potential if a way of automatically translating abstract components into concrete components is found. The adoption of the Model-Driven Engineering (MDE) (Schmidt, 2006) approach is a key step to achieve this goal.
Currently, a MDE approach to developing the software architecture of robotic systems based on the abstract components proposed by ACROSET is being adopted. The tools and standards developed by the OMG allow us to design the architecture of a robot at a high level of abstraction and in a platform-independent way, and to successively transform these models until we obtain a textual representation (code generation), ready for compilation. By designing different transformations it will also be possible to map the ACROSET components to different robotic frameworks when needed. Although this work is still at an early stage, the results we have already obtained are more than promising.

8. References

Álvarez B.; Sánchez P.; Pastor J.A.; & Ortiz F. (2006). An Architectural Framework for Modeling Teleoperated Service Robots, *ROBOTICA - International Journal of Information, Education and Research in Robotics and Artificial Intelligence* ISSN 0263-5747, Cambridge University Press. Vol. 24, No. 04, pp. 411-418. July 2006.

Barbanov, M. (1997). *A Linux-based Real-Time Operating System*. PhD thesis, New Mexico Institute of Mining and Technology, June 1997.

Bézivin, J. (2005). On the Unification Power of Models. *Journal of Software and Systems Modeling*, Vol. 4, No. 2, pp. 171–188. doi: 10.1007/s10270-005-0079-0.

Brooks, A.; Kaupp, T.; Makarenko, A., Williams, S. & Oreback, A. (2005). Towards Component-Based Robotics. *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, pp 163- 168, Aug. 2005.

Bruyninckx, H.; Konincks, B. & Soetens, P. (2002). A Software Framework for Advanced Motion Control, Dpt. of Mechanical Engineering, K.U. Leuven. *OROCOS project inside EURON*. Belgium.

Coste-Manière, E. & Simmons, R. (2000). Architecture, the Backbone of Robotic System, *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*, pp. 505-513, April 2000, San Francisco, USA.

EFTCoR (2002) Environmentally Friendly and Cost-Effective Technology for Coating Removal (EFTCOR). Fifth Framework Programme, European Community, Subprogram Growth ref. GRD2-2001-50004, (Oct 2002).

EFTCoR (2005) EFTCoR Offcial Site. http://www.eftcor.com

Iborra, A.; Pastor, J.A.; Álvarez, B.; Fernández, C. & Fernández-Meroño, J.M. (2003). Robots in Radioactive Environments. *IEEE Robotics & Automation Magazine*. Vol 10, No. 4, pp.12-22. Dec. 2003.

Fernández, C.; Iborra, A.; Álvarez, B.; Pastor, J.A.; Sánchez, P.; Fernández-Meroño, J.M. & Ortega, N. (2005). Co-operative Robots for Hull Blasting in European Shiprepair Industry. *IEEE Robotics & Automation Magazine*, Nov. 2004. ISSN: 1070-9932

Flowcorp (2006) http://www.flowcorp.com

Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J. (1995). *Design patterns : elements of reusable object-oriented software.* Ed. Addison-Wesley Proffesional, 1995. ISBN: 0-201-63361-2

Goldie, B. (a) (1999) *A comparative look at dry blast units for vertical surfaces*, PCE, Jul 1999.

Goldie, B. (b) (1999) *Comparing robotic units made to clean vertical surfaces with UHP waterjetting*, PCE, Sep 1999.

Hofmeister, C.; Nord, R. & Soni, D. (2000). *Applied software architecture*. Ed. Addison-Wesley, 2000. ISBN: 0-201-32571-3.

Hydro-Crawler (2006) http://www.dansk-vandteknik.dk/e_hydro-crawler.htm

Luckham, D.; Vera, J. & Meldal, S. (1995). Three Concepts of System Architecture. *Technical Report: CSL-TR-95-674*. Stanford University, CA, USA.

Montemerlo, N.; Roy, N. & Thrun, S. (2003). Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. In *IEEE/RSJ Intl. Workshop on Intelligent Robots and Systems*, 2003.

Octopus (2005) http:// www.cybernetix.fr

OMG (2007), Object Management Group, Unified Modeling Language (UML) Superstructure Specification v2.1.1, formal/2007-02-05, 2007.

Schmidt, D. (2006): Model-Driven Engineering. *IEEE Computer*, Vol. 39, No. 2, IEEE Computer Society. ISSN 0018-9162. doi: 10.1109/MC.2006.58.

Scholl, K.U. Albiez, J. & Gassmann, B. (2001) *MCA: An Expandable Modular Controller Architecture*, Karlsruhe University, 3rd Real-Time Linux Workshop, Milano, Italy

Sendall, S. & Kozaczynski, W. (2003). Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, Vol. 20, No. 5, pp. 42-45, IEEE Computer Society. ISSN 0740-7459. doi: 10.1109/MS.2003.1231150.

Shaw, M. & Garlan B. (1996). *Software Architecture : Perspective on a emerging discipline.* Ed. Prentice Hall, 1996. ISBN 0-131-82957-2.

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

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, 2002. ISBN 0-201-17888-5.

Ultrastrip (2007) http://www.ecospheretech.com/htm/e_rov.htm

Vaughan, R.; Gerkey, B. & Howard, A. (2003). On device abstractions for portable, reusable robot code. In *Proc. of the IEEE/RSJ Intl. Conf. On Intelligent Robots and Systems* (IROS), 2003.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; & Das, H. (2001). The CLARAty architecture for robotic autonomy. In *IEEE Proceedings of the 2001 Aerospace Conference*, Vol. 1, pp 121-132, 2001 Montana, USA.