Appears in: *Proc. of the 11th Intl. Conf. on Advanced Robotics (ICAR 2003) June 30 - July 3, 2003 - Coimbra, Portugal*

Experience in Teleoperation System Design based on Real-Time CORBA

M. Amoretti, S. Bottazzi, M. Reggiani, and S. Caselli

RIMLab – Robotics and Intelligent Machines Laboratory
Dipartimento di Ingegneria dell'Informazione
University of Parma - ITALY
Email: reggiani@ce.unipr.it

Abstract

Novel remote operation paradigms, such as tele-teaching/tele-learning, virtual laboratories, and on-line robots, represent challenging distributed robotic applications, where developers must deal with heterogeneous, dynamic, and highly concurrent control systems. Distributed Object Computing can be essential to cope with the complexity of such systems, offering the many advantages of the Object Oriented programming paradigm.

In this paper, we describe the development of a teleoperated robotic application exploiting a framework for distributed telerobotics based on advanced CORBA features, such as Asynchronous Method Invocation and real-time priorities. The case study demonstrates the potential advantages brought by CORBA on the overall development process of telerobotic applications.

1 Introduction

Distributed computing systems and Internet technologies have opened new application perspectives to robot teleoperation systems. Examples of novel applications, often broadly termed as "networked" or "on-line" robot systems [9], are tele-teaching/tele-learning, virtual laboratories, remote and on-line equipment maintenance, and projects requiring collaboration among remote users, experts, and devices [1, 2, 9]. General advantages of telerobotic systems built on top of distributed computing systems technology are reduced system cost, arbitrary location of clients, dynamic access to remote expertise as needed, backup client sites (e.g., for teleprogramming), decreased cost of operator training [13].

Deployment of many new telerobotic applications largely depends upon the cost factor. In the traditional approach to robot teleoperation [20], a dedicated system architecture ensures tight coupling of a single client device (master) and a server robot (slave) by means of a dedicated connection. The dedicated nature of the architecture makes this approach to teleoperation economically acceptable only for critical applications. The viability of many new telerobotic applications thus rests upon exploitation of standard components and technology. Furthermore, several teleoperation applications can benefit, or even require, the ability to connect multiple users at the client side, possibly with dynamic commanding roles. Such a feature is clearly incompatible with a traditional master-slave architecture.

In order to make physical resources such as robots or sensors dynamically available to multiple users, they must be managed by software applications (termed *servers*) accepting incoming requests and providing control and arbitration over resource allocation. Users interact with *client* applications for task programming and monitoring. A key feature of the system architecture is thus the interconnection channel, which now must be shared among multiple clients and servers to enable distributed collaboration. When users have arbitrary locations, or when distance and cost factors dictate it, the interconnection channel is implemented through the Internet. The use of standard, general-purpose interconnection technologies, while providing essential functional advantages, exacerbates the latency and bandwidth problems that must be faced in all telerobotic systems. These problems compound with certain properties typical of new applications. In distributed telerobotic applications, systems are extremely *heterogeneous*: they are often implemented using previously available devices, based on specialized hardware acquired from different vendors, running different operating systems and programmed in a variety of languages. Moreover, these systems are also *dynamic*: in a general scenario, sensor and robot controllers can dynamically connect to the network resulting in changes in the number and location of peers, in service roles that can change at run-time, and in the need of load reconfiguration to improve distributed system performance. Clients can also register or disconnect at run-time and bid for

commanding or supervisory roles, leading to a variety of dynamic interaction patterns.

1.1 Middleware Software

Building a teleoperation architecture from scratch for heterogeneous and dynamic systems is not always feasible, due to economic and time constraints. Following a trend in modern distributed systems design, open, reconfigurable, and scalable architectures can be built using standard middleware software for distributed object computing. Available solutions include the JavaSoft's Java Remote Method Invocation (Java RMI), Microsoft's Distributed Component Object Model (DCOM) and OMG's Common Object Request Broker Architecture (CORBA).

Sun's Java RMI (http://java.sun.com/products/jdk/rmi) provides a simple and fast model for distributed object architecture. It extends the well-known remote invocation model to allow the shipment of objects: data and methods are packaged and shipped across the network to a recipient that must be able to unpackage and interpret the message. The main drawback of the RMI approach is that the whole application must be written in Java. This constraint is troublesome in common heterogeneous environments of robotic applications, often incorporating legacy and specialized hardware and software components.

Microsoft's DCOM (http://www.microsoft.com/com/tech/DCOM.asp) supports distributed object computing allowing transparent access to remote objects. While DCOM overcomes RMI reliance on Java using an Object Description Language to achieve language-independence, it still has limitations concerning legacy code and scalability of applications. Developer's options are indeed restricted because DCOM is a proprietary solution mainly working on Microsoft operating systems.

When language-, vendor-, and operating system-independence is a goal, CORBA (http://www.corba.org) is a mature solution that provides similar mechanisms for transparently accessing remote distributed objects while overcoming the interoperability problems of Java RMI and DCOM. Several established middleware implementations binding a wide variety of languages and operating systems are currently available based on the vendor-independent specification promoted by the Object Management Group (OMG) (http://www.omg.org).

At the moment, CORBA seems the logical choice for building complex distributed applications. This paper focuses on the advantages brought by CORBA on the overall development process of a teleoperation system, described in the next section.

2 Development of a teleoperated robotic application

The teleoperation case study investigated in this paper is a modified master-slave system. The operator, provided with direct continuous teleoperated control of the remote device, is required to perform an easy peg-in-hole part mating task.

The telerobotic system is implemented with the experimental setup illustrated in Figure 1. The remote manipulator is an Unimation Puma 560, interfaced to an Intel machine with Solaris OS running the RCI/RCCL programming environment. The sensory system comprises a black and white camera and an IR proximity sensor mounted near the gripper of the manipulator, a video camera mounted on the ceiling shooting the testbed area, and a stereo vision system in front of the task site.

The remote environment can be monitored by several client stations through the broadcast of the data collected by the sensors previously listed. Each client station runs a Client Application allowing to graphically choose the required services among those avail-

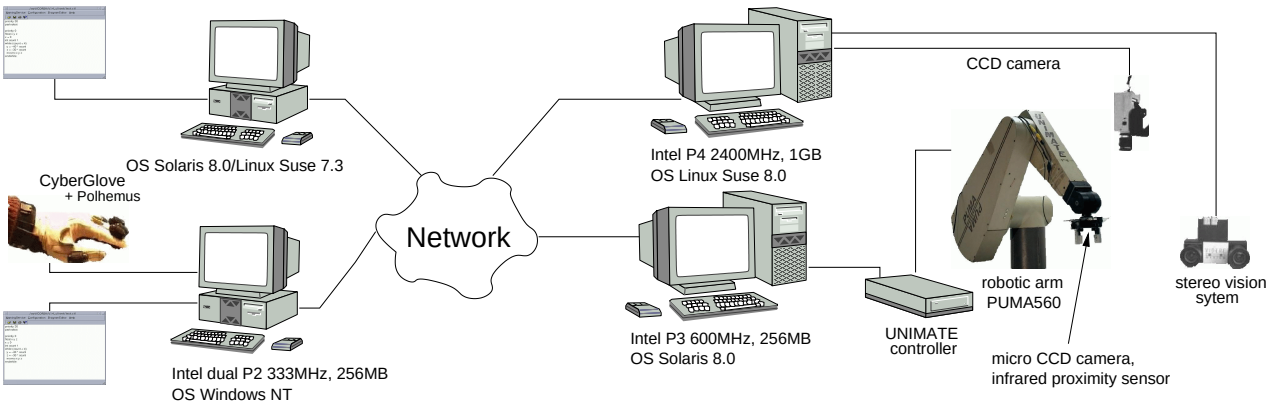

Figure 1: *The experimental testbed.*

able in the system. A more advanced station is also provided with a multimodal user interface including a virtual reality glove and a six d.o.f. tracker.

In the experiments described hereafter, local and remote computing systems are connected using a Fast Ethernet switch, which does not introduce substantial delay (the latency is less than $100\mu s$).

2.1 Client

To allow the clients to interact with the remote environment, program and monitor the task, each client station is provided with a *Client Application*. The Client Application is built upon CORBA services providing transparency about location and implementation of the available components. The user can search for components (CORBA objects) looking for a Naming Service through one of the menu items of the Graphical User Interface (Figure 2) provided with the Client Application.

The Naming Service, one of the basic CORBA Services, will then locate the objects based on their name and will return the reference to the remote object stored under that name. This allows dynamic configuration of the application (i.e. adding new sensoriality, moving available hardware) without any change in the client code.

The teleoperation application includes several heterogeneous sensors whose data must be broadcasted to the client stations and quickly returned to the user. Figure 2 shows the interface of the Client Application displaying the received information. To reduce latency of sensor return to the user, requiring concurrent update of multiple windows, the Client Application has been implemented with a multi-threaded structure, overlapping computation and communication.

For the definition of application tasks, the operator can choose between two alternatives. The simplest one is to submit a sequence of single commands (i.e., move the arm to position P, grab the video image, get the value of the proximity sensor, close the gripper,...). A more practical alternative is to define the application by editing a program, that will then be interpreted, using a C-like high level language. To execute the sequence of single actions, the Client Application can use Synchronous Method Invocation on remote objects, blocking itself until the server notifies the end of the requested activity. This wait-and-go approach, however, cannot be used in the proposed testbed application. Indeed, the client needs to invoke execution of concurrent actions at the server asking to receive data from multiple sensor sources while controlling the movement of the arm and of the gripper. As recently the Asynchronous Method Invocations (AMI) model, dealing with non-blocking invocation, has been introduced in the CORBA standard, the developed application has taken advantage of asynchronous calls implementing a *waiting rendezvous* strategy [3] to meet these requirements.

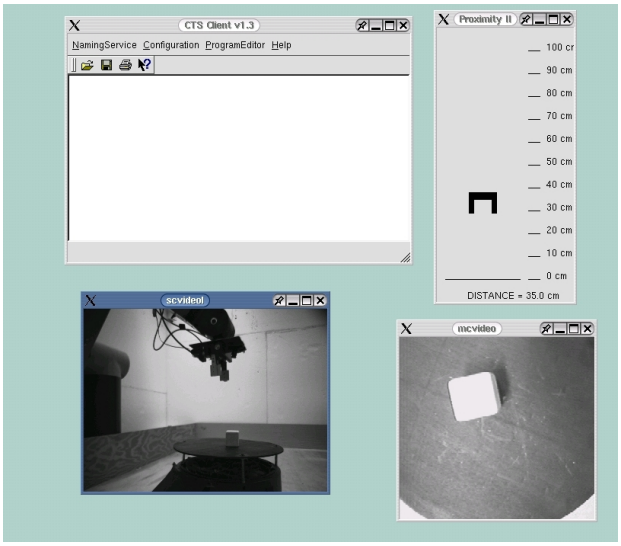

Figure 2: *A snapshot of the Client Application output with windows displaying sensory information received from the remote site.*

2.2 Server

As shown in Figure 1, the testbed includes a *Server Application* managing a Puma manipulator and several sensors according to client requests, whose order and urgency should be preserved.

To coordinate manipulator usage and guarantee its own consistency when accessed by concurrently executing computations, the Server Application exploits the CORBA Concurrency Control Service. This service uses locks to represent the ability of a specific client to access a specific resource in a particular way. Coordination is therefore achieved by preventing multiple clients from simultaneously possessing locks for the same resource if the activities of those clients might conflict [15]. Still under development is instead the use of the Security Service to provide protection against the possibility, for an authorized user of the system, to gain access to CORBA object methods that should be hidden to him. Currently, there are two separate Client Applications, one for standard users and the other for the supervisor user. The latter can also stop the whole application in emergency case, but a safer and more appropriate design should instead exploit the Security Service.

The testbed application allows multiple clients to concurrently monitor and control the task, possibly resulting in a high number of requests to the server. Managing these requests while preserving their invocation order and urgency requires a server with an efficient multithreaded architecture. To meet this re-

quirement, the Server Application has been implemented using the Thread Pool mechanism, available in RT CORBA [7]. With this mechanism, a group of threads is statically created at start-up time thus avoiding the overhead of thread creation/destruction at run-time and constraining the maximum number of threads on the server.

The Server Application must also be sensitive to client request priorities. Indeed, supervisor's requests are more urgent than standard users' ones, and even the same client can send requests of different urgency. To support priorities of client requests, we take advantage from a specific feature of the RT CORBA standard, Priority Mapping, to convert CORBA priority levels assigned to CORBA operations to OS native priority levels or vice versa. RT CORBA introduces also two models that can be adopted on a per-method invocation basis: `SERVER_DECLARED` and `CLIENT_PROPAGATED`. In the `SERVER_DECLARED` model each object is associated to a priority level during its creation, whereas in the `CLIENT_PROPAGATED` model the Client Application establishes the priority of each method invocation, and the Server Application must honor this priority.

A functionality implemented in the server thanks to RT CORBA priority features is the "Emergency Button" object, whose only method *push()* stops the application in emergency situations. The priority of this method is `SERVER_DECLARED` as it must always be executed at the highest available priority to immediately stop the system. To deal with the possibility that the server cannot immediately execute the method due to priority inversion or thread being exhausted, Thread Pool with Lanes, another feature of RT CORBA, has been exploited. As threads in separate Lanes are associated to different, non-overlapping ranges of priorities, the creation of a lane with a single thread at the highest priority reserved for the execution of the push method guarantees its predictable behavior.

2.3 Distributing sensor data

The problem of distributing sensor data to multiple clients is typical of new telerobotic applications and deserves a careful attention. For efficiency, data acquisition and distribution should avoid polling operations. Moreover, each client should be able to define the desired data receiving rate to avoid unnecessary distribution of data.

To achieve these goals, distribution of sensor data is based on the Observer pattern [8]. This pattern requires definition of two CORBA classes for each available sensor: the *subject* at the server side and the *observer* at the client side. An example of the implemented solution for a hand-eye microcamera follows (Figure 3). To receive video data from a camera, a Client Application (C2) calls the method `attach(Video x)` (Figure 3-1) on the Camera object (the *subject*), passing a reference to a Video object (the *observer*) Each Camera object holds a list of all Video objects that have been attached. When video data are ready, the Server Application can invoke the `video_obj[i].draw(image_data)` (Figure 3-2) method of all "attached" video objects. The method `detach()` (Figure 3-3) allows the client to stop the video stream, removing its reference from the data distribution list.

3 Experiments and discussion

In [3] we proposed a software framework based on Real-Time CORBA and we assessed its effectiveness in the concurrent execution (with different priority levels) of simple telerobotic tasks. In this paper we have verified the completeness, modularity and flexibility of the framework implementing the peg-in-hole task, a complete telerobotic application with rich and heterogeneous sensorial data returned to multiple clients.

The development of a Server required only the implementation of CORBA objects for supported devices, i.e. no further code implementation is needed for the communication layer since the Server code is based on already available CORBA Services integrated in the framework.

In the context of the peg-in-hole experimental setup, we evaluated the influence of sensor feedback on operator's performance. To this purpose, we implemented several testbeds combining subsets of the available field sensors (on-board camera, IR proximity sensor and field-vision camera). The task involved insertion of a square peg with about 3 *mm* peg-hole clearance. After a training session of two minutes with the input device (glove and tracker), ten unskilled operators were able to successfully complete the peg-in-hole task. An analysis of the times to perform the

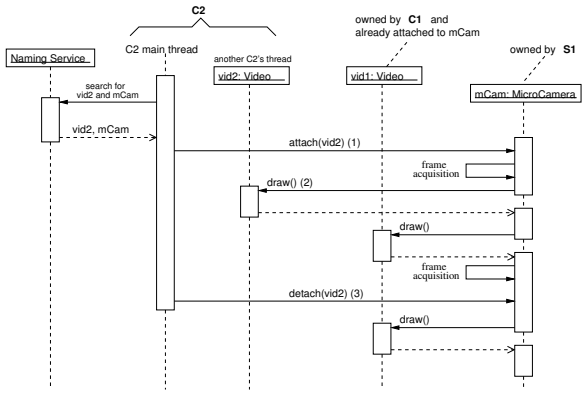

Figure 3: *Video broadcasting: interaction between the CORBA objects when two Client Applications (C1 and C2) are attached to the same Server (S1).*

tests revealed that operators provided with feedback from multiple sensors had a faster learning curve, i.e., faster decreasing experimental duration across consecutive peg-in-hole tasks.

The implementation of the various alternative testbeds exploited the capability of the framework to easily cope with changes in the architecture: new devices could be easily and quickly integrated in the Server, requiring only implementation of the interface and methods available to Client Applications. Moreover, the location transparency provided by CORBA and portability of the framework to a number of operating systems allowed the same code to be used in several Client stations and reallocation of sensoriality among heterogeneous Server stations.

The class diagram in Figure 4 shows an example of the use of the framework for the development of one of the studied testbed. The example involves two Server Applications, one for manipulator control and local sensoriality and the other for distribution of video images from a stereo vision system.

4 Related Work

The use of Internet for the execution of distributed task in telerobotic systems has rapidly increased during the last decade [6]. Many researchers have investigated the difficulty of performing assembly operations over the Internet as a result of communication delays and packet losses [13, 14, 19, 5]. Another stream of research has addressed the interaction between robot and web users, usually lacking technical skills, and therefore requiring easy-to-use command interfaces [9, 2].

A drawback of several early Internet telerobotic architectures is that they are rebuilt from scratch, and based on Socket API or on CGI. These technologies are suitable for development of simple interfaces but are clearly inadequate in providing fundamental services for complex distributed applications [12]. These problems can be overcome by integrating COTS components to reduce the software life-cycle costs, but only recently object-oriented middleware, such as CORBA, has been used to develop telerobotics systems. Initial work on teleoperation with CORBA took a simple approach to ORB technology, ignoring fundamental components such as the Naming Service for location transparency [4], or exploited CORBA only for interoperability of previously developed components [17].

Following these experiences, other investigations used CORBA to achieve interoperability and location transparency in their applications and to exploit other useful CORBA Services [6,16,10,11,21]. Nevertheless, to our knowledge, they do not take full advantage of the multithreading and asynchronous method invocation extensions of the CORBA standard.

Nonblocking asynchronous communications have been viewed as essential to build a distributed robotic system [6], but unavailability of this feature in earlier CORBA implementations required either exploitation of non-standard middlewares [6] or use of CORBA oneway methods [16] for nonblocking communica-

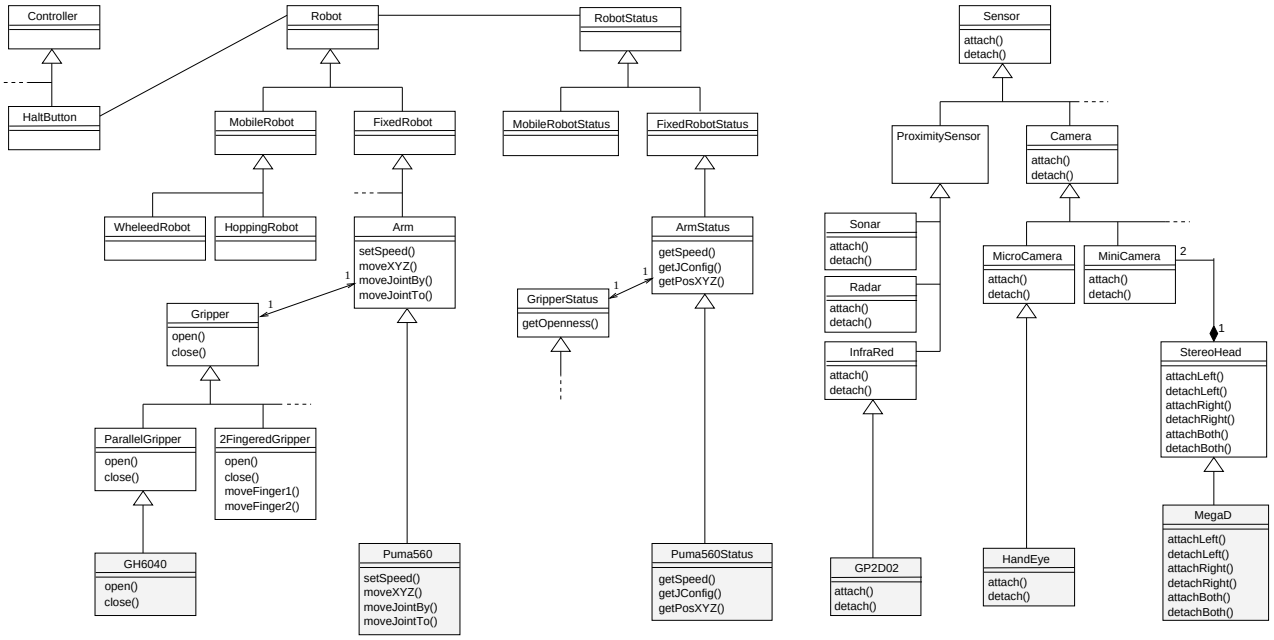

Figure 4: *The application task Class Diagram.*

tions. The second solution is quite questionable due to the "best effort" semantics of oneway operations, which could result in a loss of calls or in a change in the order of packet reception. With the current AMI invocation model, these implementation choices are no longer justified. We believe that these features provide fundamental advantages in the development of distributed real-time and embedded systems.

As shown in this paper, Real-Time CORBA ORBs allow these systems to use multithreading while controlling the amount of memory and processor resources they consume. Previous CORBA releases, instead, did not provide standard mechanisms to deal with request priority and multithreading. These features are needed, however, to reduce unbounded priority inversion without the need of costly hand-crafted mechanisms [18].

5 Conclusion

The growing popularity of the CORBA architecture as middleware for distributed real-time and embedded systems has made it an interesting candidate for development of telerobotic applications. In this paper, through the development of a realistic telerobotic experiment, we investigated whether two recent extensions of the CORBA standard (Asynchronous Method Invocation and Real-Time CORBA) make it more suitable for the needs of telerobotic applications.

Our experience in the development of the case study indicates that the latest CORBA specifications do reduce the programming effort in complex applications, hiding to developers details regarding the communication among distributed objects. Of course, CORBA does not implement an application by itself, and additional work is required to develop a telerobotic system. Our experience has led to a software framework [3] that achieves flexibility, portability, openness, extensibility, and reusability of the application. We are currently working on the framework to improve the distribution of sensor data and to take advantage from other CORBA Services coping with fault-tolerance and security issues.

6 Acknowledgments

This research is partially supported by MIUR (Italian Ministry of Education, University and Research) under project RoboCare (A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components).

References

[1] *WS2001: International Workshop on Tele-Education in Mechatronics Based on Virtual Laboratories*, Weingarten, Germany, July 2001.

[2] P. Backes, K. Tso, and G. Tharp. Mars Pathfinder Mission Internet-Based Operations using WITS. In *IEEE Int. Conf. Robotics and Automation*, May 1998.

[3] S. Bottazzi, S. Caselli, M. Reggiani, and M. Amoretti. A Software Framework based on Real-Time CORBA for Telerobotic Systems. In *IEEE Int'l Conf. Intelligent Robots and Systems*, 2002.

[4] R. L. Burchard and J. T. Feddema. Generic Robotic and Motion Control API Based on GISC-Kit Technology and CORBA Communications. In *IEEE Int. Conf. Robotics and Automation*, 1996.

[5] N.Y. Chong, T. Kotoku, K. Ohba, and K. Tanie. Virtual Repulsive FOrce Field Guided Coordination for Multi-telerobot Collaboration. In *IEEE Int. Conf. Robotics and Automation*, 2001.

[6] B. Dalton and K. Taylor. Distributed Robotics over the Internet. *IEEE Robotics & Automation Magazine*, 7(2):22–27, June 2000.

[7] V. Fay-Wolfe, L.C. DiPippo, G. Cooper, R. Johnston, P. Kortman, and B. Thuraisingham. Real-Time CORBA. *IEEE Trans. Parallel Distrib. Syst.*, 11(10):1073–1089, 2000.

[8] E. Gamma, R. Helm, R.E. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9] K. Goldberg and R. Siegwart, editors. *Beyond Webcams: an Introduction to Online Robots*. The MIT Press, 2001.

[10] H. Hirukawa and I. Hara. Web-Top Robotics. *IEEE Robotics & Automation Magazine*, 7(2):40–45, June 2000.

[11] S. Jia, Y. Hada, Y. Gang, and K. Takase. Distributed Telecare Robotic Systems Using CORBA as a Communication Architecture. In *IEEE Int. Conf. Robotics and Automation*, 2002.

[12] K. Kawabata, T. Sekine, T. Suzuki, T. Fujii, H. Asama, S. Kazumi, and I. Endo. Mobile robot teleoperation system utilizing a virtual world. *RSJ Advanced Robotics*, 15(1):1–16, 2001.

[13] R. L. Kress, W.R. Hamel, P. Murray, and K. Bills. Control Strategies for Teleoperated Internet Assembly. *IEEE/ASME Trans. Mechatron.*, 6(4):410–416, 2001.

[14] A. Levelé, P. Fraisse, P. Dauchez, and F. Pierrot. Modeling and Simulation of Robotic Tasks Teleoperated through the Internet. In *IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics*, 1999.

[15] Object Management Group. *The Common Object Request Broker: Architecture and Specification Revision 2.4*, October 2000.

[16] T. Ortmaier, D. Reintsema, U. Seibold, U. Hagn, and G. Hirzinger. The DLR Minimally Invasive Robotics Surgery Scenario. In *Work. Advances in Interactive Multimodal Telepresence Systems*, 2001.

[17] C. Paolini and M. Vuskovic. Integration of a Robotics Laboratory using CORBA. In *IEEE Int. Conf. Systems, Man, and Cybernetics*, 1997.

[18] I. Pyarali, D.C. Schmidt, and R.K. Cytron. Techniques for Enhancing Real-time CORBA Quality of Service. *IEEE Proceedings*, 2003.

[19] D. Schulz, W. Burgard, and A.B. Cremers. Robust Visualization of Navigation Experiments with Mobile Robots over the Internet. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1999.

[20] T. B. Sheridan. *Telerobotics, Automation, and Human Supervisory control*. MIT Press, Cambridge, MA, 1992.

[21] R. Siegwart, P. Blamer, C. Portal, C. Wannaz, R. Blank, and G. Caprari. RobOnWeb: A Setup with Mobile Mini-Robots on the Web. In Ken Goldberg and Roland Siegwart, editors, *Beyond webcams: an introduction to online robots*. The MIT Press, 2001.