# Telerobotic Systems Design Based on Real-Time CORBA

**Michele Amoretti, Stefano Bottazzi, Stefano Caselli, and Monica Reggiani**
*RIMLab—Robotics and Intelligent Machines Laboratory*
*Dipartimento di Ingegneria dell'Informazione*
*University of Parma*
*Parco Area delle Scienze, 181/A*
*43100 Parma*
*Italy*
*e-mail:*
*{amoretti,bottazzi,caselli,reggiani}@ce.unipr.it*

Received 28 March 2004; accepted 28 March 2004

A new class of telerobotic applications is making its way into research laboratories, fine arts or science museums, and industrial installations. Virtual laboratories and remote equipment maintenance are examples of these applications, which are built exploiting distributed computing systems and Internet technologies. Distributed computing technologies provide several advantages to telerobotic applications, such as dynamic and multiuser access to remote resources and arbitrary user locations. Nonetheless, building these applications remains a substantial endeavor, especially when performance requirements must be met. The aim of this paper is to investigate how mainstream and advanced features of the CORBA object-oriented middleware can be put to work to meet the requirements of novel telerobotic applications. We show that Real-Time CORBA extensions and asynchronous method invocation of CORBA services can be relied upon to meet performance and functional requirements, thereby enabling teleoperation on local area networks. Furthermore, CORBA services for concurrency control and large-scale data distribution enable geographic-scale access for robot teleprogramming. Limitations in the currently available implementations of the CORBA standard are also discussed, along with their implications. The effectiveness and suitability for telerobotic applications of several CORBA mechanisms are tested first individually and then by means of a software framework exploiting CORBA services and ensuring component-based development, software reuse, low development cost, fully portable real-time and communication support. A comprehensive telerobotic application built based on the framework is described in the paper and evaluated on both local and wide area networks. The application includes a robot manipulator and several sensory subsystems under concurrent access by multiple competing or collaborating operators, one of which is equipped with a multimodal user interface acting as the master device. © 2005 Wiley Periodicals, Inc.

## 1. INTRODUCTION

In the past few years, a number of novel applications have emerged from the combination of distributed computing systems, Internet technologies, and robot teleoperation concepts. These applications, often broadly termed as "networked" or "on-line" robot systems,[1] include teleteaching/telelearning, virtual laboratories, remote and on-line equipment maintenance, supervision of robotic systems at distant sites, projects requiring collaboration among remote users, experts, and devices, and large-scale monitoring of scientific experiments, possibly including public outreach in media-covered events.[1–9] General advantages of telerobotic systems built on top of distributed computing systems technology are arbitrary location of users, dynamic and multiuser access to remote sites, and reduced system cost thanks to the exploitation of standard components and infrastructures. Application-specific advantages include access to remote expertise as needed (e.g., for telemedicine and telesurgery), backup operator sites (e.g., for teleprogramming), decreased cost of operator training.[10]

New remote robot operation applications depart in several ways from traditional robot teleoperation systems. In the traditional approach to robot teleoperation,[11] the system architecture ensures tight coupling of a single user device (master) and a server robot (slave) by means of a dedicated connection. The dedicated nature of the architecture makes this approach only suitable for critical teleorobotic applications.

New telerobotic applications often require the ability to connect multiple users at the client side, possibly operating from different locations and with commanding roles dynamically shifting during task execution. The teleoperated environment may even include multiple robotic and sensory devices, which can be controlled in a coordinated manner thanks to the availability of multiple operators. Another distinguishing feature of novel applications is their exploitation of standard components and technology as key interconnection infrastructure. Distributed systems technology allows a much broader set of applications to be tackled in an economically sensible manner. These telerobotic applications often become viable in the context of an existing infrastructure or a constrained budget, which leads to the development of heterogeneous systems built by integrating a mix of new and legacy equipment, with a variety of hardware, operating systems, and programming language to be taken into account. When users have geographically distributed locations, or when distance, accessibility, and cost factors dictate it, the interconnection channel can be implemented through the Internet.

Several new telerobotic applications aim to provide access to remote robotic resources for a very large number of users, for example by accepting multiple observing users or by queueing requests to gain exclusive access. In these applications the user interface is often a major issue, both to enable naive users to easily access the remote system and to alleviate the impact of network latency and limited bandwidth by means of local predictive simulation. The user interface plays an important role also for execution of complex tasks (e.g., difficult assembly tasks, telesurgery), where operators should be provided with an immersive environment integrating rich sensory information returned from the remote site. When an open channel is exploited to access remote resources, security and authentication are also important issues.

The set of features outlined above and characterizing novel telerobotic applications are clearly incompatible with a traditional master-slave architecture. They rather depict a scenario like the one shown in Figure 1, with a community of users accessing multiple robotic devices in a dynamic but coordinated manner. Accomplishing these features in telerobotic systems must be achieved without dismissing the consolidated challenges of the field, which include ensuring real-time operation at the target, dealing with large time delays, coordinating multiple robotic devices, meeting some performance criteria (accuracy, speed), and, possibly, guaranteeing a degree of fault-tolerance for more critical applications.

Moreover, component-based development and software reuse are required to achieve the goals of portability and low development cost of novel telerobotic applications. Considering robots, sensors, and controllers as objects,[12] and networked robots as distributed objects[3] has been the first step towards the idea of open, reusable and scalable software architectures for teleoperation.

The purpose of this paper is to describe how mainstream and advanced features of the CORBA object-oriented middleware can be exploited to meet the requirements of novel telerobotic applications. We review a number of CORBA services and show how Real-Time CORBA extensions and asynchronous method invocation meet performance and functional requirements, thereby enabling teleoperation on local area networks. Additional CORBA services for concurrency control and large-scale data distribution provide an effective infrastructure to meet common requirements arising in geographic-scale access for robot teleprogramming. Limitations in currently

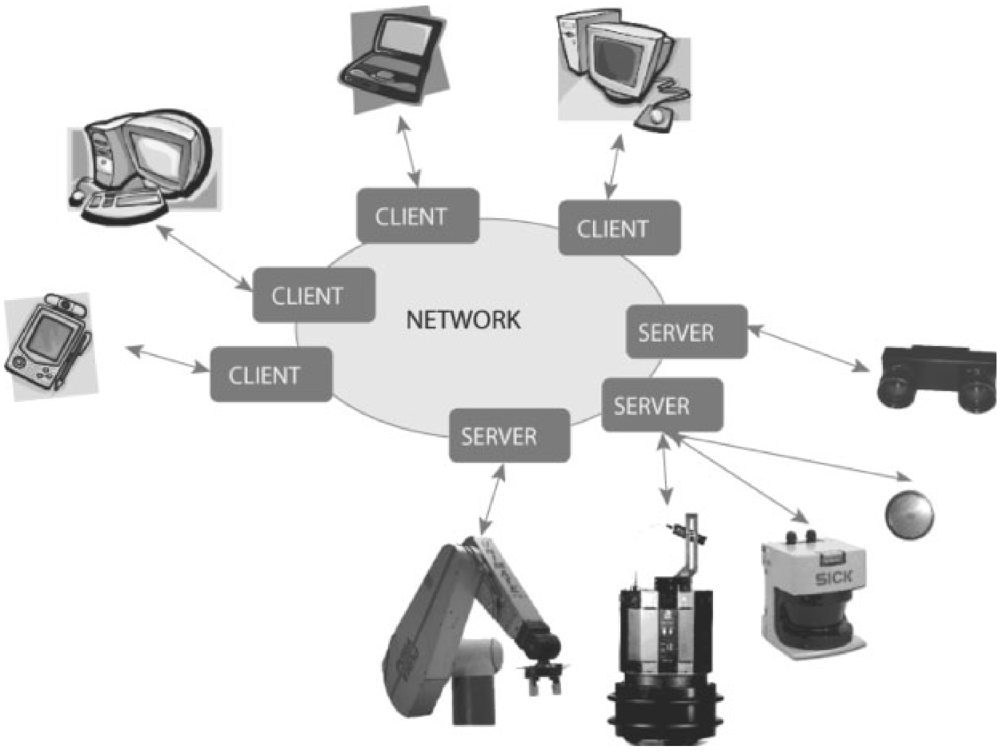

**Figure 1.** Scenario for novel telerobotic systems.

available implementations of the CORBA standard (e.g., for fault-tolerance and security) are also discussed.

In this paper, we perform a thorough assessment of required CORBA services once integrated in the context of a software framework for telerobotic systems.[13,14] We also report results from a teleoperated robot manipulation application with multiple concurrent and cooperating users. The application has been built using the CORBA-based framework and experimented on both local and geographic area networks.

The remaining of the paper is organized as follows. Section 2 describes the requirements of telerobotic architectures for new applications. Section 3 summarizes related distributed computing technologies and briefly introduces the general features of the CORBA standard. Section 4 illustrates and evaluates CORBA services relevant to telerobotic applications. Section 5 describes a telerobotic application built based on a CORBA-based software framework and reports the obtained results. Section 6 provides links to other research and experimentation work in networked and on-line robotics. Finally, Section 7 concludes the paper and discusses its main results.

## 2. REQUIREMENTS OF NEW TELEROBOTIC SYSTEM ARCHITECTURES

In order to make physical resources, such as robots and sensors, available to multiple users, resources must be managed by software applications (termed *servers*) accepting incoming requests and providing control and arbitration over their allocation. Users interact with *client* applications for task programming and monitoring. A key feature of the system architecture in Figure 1 is thus the interconnection channel, which now must be shared among multiple clients and servers to enable distributed collaboration. Developing a suitable system architecture meeting client and server requirements remains a difficult and time-consuming task. In many cases, the expensive development of a new application can be avoided relying on previous experience or, even better, on a common framework from which specific architectures are instantiated.

Stemming from the features characterizing novel telerobotic applications, we identify the following set of requirements for the system architecture.

The systems must ensure **interoperability and location transparency** of resources and applications, enabling multiple operators to flexibly access available resources regardless of their physical location and heterogeneity. To achieve portability, peers (applications which can act as a client, as a server, or both) should be lightweight and runnable on several platforms (from desktop workstations to embedded systems), without increasing applications complexity. Support for different robot programming methods (on-line, off-line, hybrid) should be provided, along with transparency (i.e., all accessible applications should appear as if they were local).

Moreover, modern telerobotic systems (Figure 1) tend to be dynamic in nature, with users and physical resources, like sensor and robot controllers, which need to be connected/disconnected at run-time to the system (e.g., students or researchers from geographically distributed sites that want to access equipments which are only part-time available, in a virtual laboratory). Hence, **support for multiple clients** is required in the system, along with suitable **authentication and concurrency protocols** to ensure safe access to resources according to prescribed policies and priorities. Operators, in fact, may require or be entrusted with different access levels: from simple sensor monitoring, to robot teleoperation, to system supervision. Given the possibility of multiple concurrent clients, servers should also provide synchronization mechanisms for allocation of resources when they cannot be shared among peers.

Servers should ensure **real-time operation** to allow implementation of the appropriate control laws with guaranteed operation. Moreover, they should accept and manage client requests preserving their ordering, and exhibit differentiated reactions depending on their urgency. The dynamics of multiple client interaction results in changes in the number and location of peers and in the need to preserve performance of services also with variable system load. A telerobotic system should also provide operators with the guarantee of correct execution priorities of application tasks at the server. Consistent, end-to-end propagation of priority semantics is especially difficult when heterogeneous peers, with different operating systems and programming languages, must be accommodated.

Control of several robots and sensors teleoperated by multiple remote clients implies **concurrency in server operations** and a **multithreaded server structure**. This capability enables execution of parallel independent or coordinated actions in the target workcell, improves the response time perceived by operators, and simplifies CPU sharing among computational and communication services. Portable thread synchronization mechanisms should therefore be available to achieve reliable and efficient concurrency in servers.

The system should allow **asynchronous (non-blocking) client requests** to servers. This feature is not required for simple teleoperation applications consisting in the stepping of one action at a time, possibly in stop-and-go mode, but becomes necessary for advanced telerobotic scenarios where the user can invoke execution of multiple concurrent actions at the server.[5] Examples of such tasks are coordinated operation of multiple arms or concurrent sensing and manipulation.

In a telerobotic application, the timely availability of adequate sensory data so as to emulate the operator's physical presence at the remote site is often crucial.[15,16] In new telerobotic applications, **efficient and scalable data distribution** techniques must be available to return sensory information to a potentially very large and dynamic set of users. Regardless of their number, users must be provided with sufficient data for their real-time interaction with remote systems.

Finally, many new telerobotic applications have become viable thanks to the availability of standard infrastructures and technologies.[1] Furthermore, deployment of telerobotic applications often depends upon the cost factor. These considerations motivate the adoption of **components-off-the-shelf (COTS) and component-based design** in developing new telerobotic applications. Approaches promoting **software reuse**, such as developing a common framework and exploiting COTS, have the potential of drastically reducing development time and cost of new telerobotic applications.

Technologies coping with these requirements and suitable for the design of telerobotic systems are investigated in the following sections.

## 3. DISTRIBUTED COMPUTING TECHNOLOGIES FOR TELEROBOTIC APPLICATIONS

In the previous section, features and requirements for advanced telerobotic systems have been discussed. Building such a type of teleoperation architecture from scratch is often too demanding, 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 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 architectures. RMI extends the well-known remote invocation model to allow 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. Developers' options are indeed restricted, since 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. Moreover, its most advanced and recent features (Real-Time CORBA, AMI) provide functionalities almost essential in telerobotic applications, as will be discussed in the next section. At the moment, CORBA seems the logical choice for building complex distributed telerobotic applications, thereby satisfying the interoperability, transparency, COTS availability, component-based design, and software reusability requirements previously highlighted.

### 3.1. CORBA Architecture Basics

Implementing a distributed architecture with CORBA allows smooth integration of heterogeneous software components. To ensure portability, reusability, and interoperability, the CORBA architecture is based on the Object Request Broker (ORB), a fundamental component that behaves as a system bus, connecting objects operating in an arbitrary configuration (Figure 2).

To achieve language independence, CORBA re-

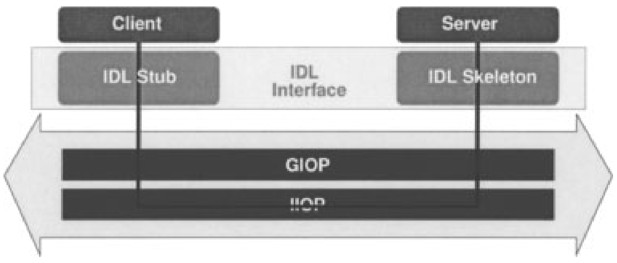

**Figure 2.** Basic CORBA architecture: Client/Server interaction through the ORB.

quires developers to express how clients will make a request to a service using a standard and neutral language: the OMG Interface Definition Language (IDL). After the interface is defined, an IDL compiler automatically generates client stubs and server skeletons according to the chosen language and operating system. Client stub implementation produces a thin layer of software that isolates the client from the Object Request Broker, allowing distributed applications to be developed transparently from object locations. The Object Request Broker is in charge of translating client requests into language-independent requests using the Generic Inter-ORB Protocol (GIOP), of communicating with the Server through the Internet Inter-ORB Protocol (IIOP), and of translating again the request in the language chosen at the server side.

Together with the Object Request Broker, the architecture proposed by OMG introduces several CORBA Services, providing capabilities that are needed by other objects in a distributed system. We refer to the literature for additional information on general features of the CORBA Standard.[17] Based on CORBA technology, we have developed an object-oriented framework for telerobotic applications. Some of the features of the framework have been described in refs. 13 and 14. The implementation of the framework is written in C++ and based on The ACE ORB (TAO),[18] a freely available, open-source, and standard-compliant real-time implementation of CORBA.

## 4. CORBA FEATURES FOR TELEROBOTIC APPLICATIONS

This section discusses recent CORBA features, introduced starting from version 2.4 of the Standard,[17] and other CORBA Services relevant to telerobotic systems. Features and Services are evaluated by means of simple distributed applications designed to assess their suitability for telerobotics.

### 4.1. Concurrency Among Clients

A requirement of advanced telerobotic systems is to manage input from all operators while generating a single and coherent control sequence for each robot, allowing collaborative and coordinated teamwork.[19]

As a basic functionality, server application must ensure atomicity of calls to library functions devoted to the interaction with the robot controller, regardless of the server thread-safety. Potential conflicts arising from multiple Clients can be avoided forcing an exclusive access to library functions through the RTCORBA::Mutex construct, implementing the mutual exclusion lock. The server side is solely responsible for the implementation of this functionality, since Mutexes are introduced in the servant code.

In addition to ensuring single command consistency and atomicity, concurrent access control at session level must be implemented to guarantee full robot control without undesired interferences from other operators. Implementation of a coordination scheme allowing multiple clients to control a single robot through a coherent and logically safe pattern of interaction can be obtained exploiting the CORBA **Concurrency Control Service**.[20] This Service allows several Clients to coordinate their concurrent accesses to a shared resource so that the resource consistent state is not compromised. The Concurrency Service does not define what a resource is. It is up to the developer to define resources and identify situations where concurrent accesses to resources conflict.

The coordination mechanism provided by the Concurrency Service is the lock. Each shared resource should be associated with a lock, and a Client must get the appropriate lock to access a shared resource. Several lock modes (read, write, upgrade, intention read, intention write) are defined, allowing different resolution of conflict among concurrent Clients. The specification defines two types of Client for the Concurrency Service: a transactional Client, which can acquire a lock on behalf of a transaction, and a non-transactional Client, which can acquire a lock on behalf of the current thread. Our tests adopt a non-transactional style, since most available RT CORBA implementations do not support transactional Clients yet.

In a scenario where several users compete to control a single robot and/or access data from multiple sensors, exclusive control of the robot must be granted only to one user in a given interval of time, while simultaneous read of sensor data should be allowed to other users as well. The scheme in Figure 3 shows how to cope with this requirement using the

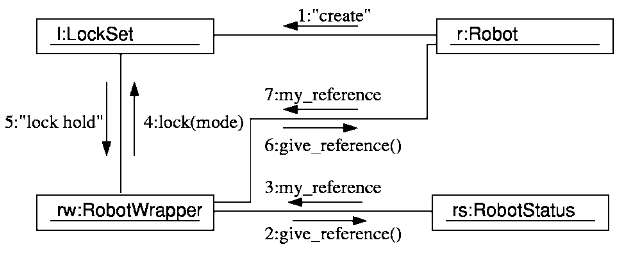

**Figure 3.** UML Collaboration diagram describing a Client (whose core is an object of the RobotWrapper Class) that asks a lock before it is able to control a Robot object.

Concurrency Service. For each robot a Robot and a RobotStatus objects are created. The RobotStatus class maintains information about a robotic device, whereas the Robot class controls movements and sets parameters. Then, for each Robot object, a CORBA::CosConcurrencyControl::LockSet object is created and registered in the Naming Service.

At the client side a RobotWrapper object contains all the references to CORBA objects and interacts with the Concurrency Control Service to enforce the correct concurrent access. As shown in Figure 4 (scenario 1), the client invoking commands on the Robot object holds a write lock ensuring exclusive control. Indeed, as the write lock conflicts with any other lock, a client requesting a lock on the same resource will be suspended waiting its release. Clients invoking methods on the RobotStatus object, instead, are not required to hold locks as the class has only "accessor" methods.

To preserve generality and cover a wider domain of applications,[19] an alternative scenario can be outlined, where a group of users want to control a single robot in a collaborative way (e.g., a "primary" operator with some "backup" operators), while preventing further operators from obtaining exclusive control of the robot. In this scenario (Figure 4, scenario 2), a collaborating client holds a read lock. Since the read lock

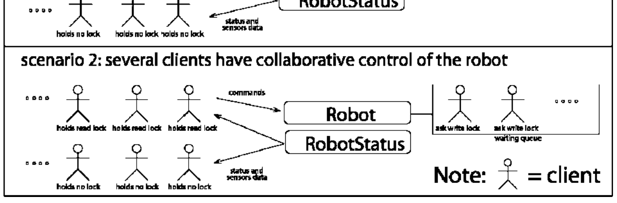

**Figure 4.** Two scenarios of concurrent access to a single robot device from several Clients.

conflicts only with write and intention write locks, it allows sharing of robot control with other clients holding read locks, whereas client requesting exclusive control through a write lock are suspended in the waiting queue.

We defer the experimental evaluation of access control mechanisms for multiple concurrent clients to the complete telerobotic application experiment in Section 5.

### 4.2. Concurrency in the Server and Real-Time Guarantees

Control of several robots and sensors teleoperated from multiple remote clients requires one or more multithreaded servers allowing concurrency among actions. Moreover, servers should be able to discriminate among services, granting privilege to critical tasks (emergency stop, reading of safety sensors), and should avoid priority inversion, with low-priority tasks blocking high-priority ones.

With the Real-Time CORBA specification,[21] OMG released a set of standard CORBA APIs for multithreading, thereby avoiding the use of proprietary ORB features to program multithreaded real-time systems. The core mechanism of RT CORBA is the **Thread Pool**, enabling preallocation of Server resources. With the Thread Pool mechanism, a group of threads is statically created by CORBA in the server at start-up time. These threads are always ready to be bound to requested methods. To achieve predictability, a fixed cap is set for dynamic threads, which are created only once static threads are exhausted. The Thread Pool avoids the overhead of thread creation/destruction at run-time and helps in guaranteeing performance by constraining the maximum number of threads in each host.

Under the extreme condition where its whole set of threads has been bound to low-level requests, a server could miss the deadlines of high-priority actions, a situation clearly unacceptable in a robot teleoperation system. To avoid depletion of threads by low-priority requests, a Thread Pool can be further partitioned in **Lanes** of different priority. This partitioning sets the maximum concurrency degree of the server and the amount of work that can be done at a certain priority. Partitioning in Lanes and related parameters cannot be modified at run-time; the only freedom is reserved to higher priority methods which can "borrow" threads from lower level Lanes once their Lane is exhausted.

Servers in telerobotic applications should also provide clients with the guarantee of correct execution priorities of application tasks. Heterogeneity of nodes, in distributed applications, precludes the use of a common priority scale, forcing users of earlier CORBA versions to concern about low-level details of threads on different OSes. The Real-Time CORBA **priority mapping** mechanism converts CORBA priority levels, assigned to CORBA operations, to OS native priority levels (and vice versa).

Teleoperation applications also require task execution at the right priority on the Server side. RT CORBA defines two invocation models: SERVER_DECLARED, in which objects are created with assigned execution priority, and CLIENT_PROPAGATED, in which the Client establishes the priority of the methods it invokes, and this priority is honored by the Server.

Thanks to the Thread Pool and Priority features, a server based on RT CORBA can thus guarantee real-time execution of critical computations and achieve coherent, end-to-end priority semantics.

Another client mechanism enabling to take advantage of available concurrency in the server has been introduced in the CORBA 2.3 Messaging Specification.[17] Standard service requests in CORBA systems rely on the Synchronous Method Invocation (SMI) model, that blocks the client until the server notifies the end of the requested activity. Non-blocking invocations with earlier CORBA versions either relied on methods not guaranteeing the delivery of the request or on techniques requiring significant programming efforts and known to be error prone.[22]

The **Asynchronous Method Invocation (AMI)** model[23] provides a more efficient way to perform nonblocking invocations with either a Polling or a Callback approach. The AMI interface allows a CORBA-based system to efficiently activate multiple concurrent actions at a remote teleoperated site. Moreover, as AMI and SMI share the same object interface, clients can choose between synchronous or asynchronous calls while server implementation is not affected.

The AMI interface, however, cannot guarantee that a set of parallel actions will be actually executed at the "same" time. When a server receives non blocking requests from a client, it dispatches them to the Thread Pool according to their priorities and they are started as soon as possible. Due to the communication delays of the distributed system, requests will reach the server at different and unpredictable times. Synchronized parallel action execution is thus unlikely. This behavior is not satisfactory for many robotic applications, where a set of parallel actions

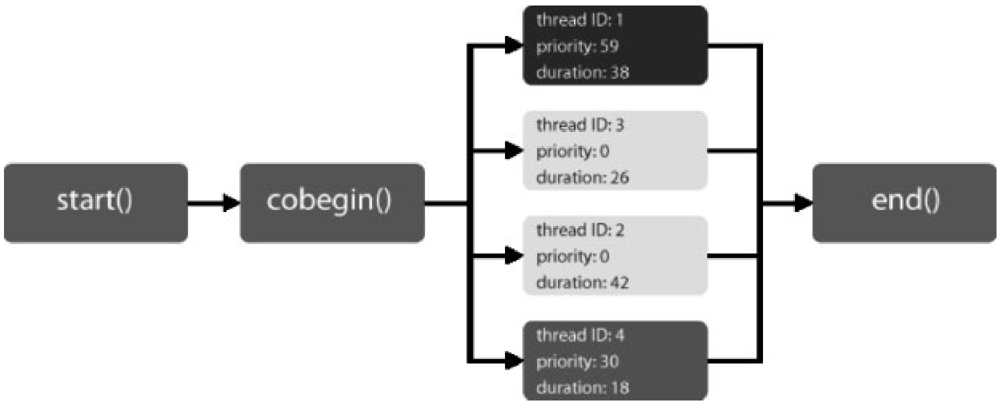

**Figure 5.** Precedence graph of a concurrent task, consisting of an initial action start() followed by four concurrent actions with different priority.

must often begin at the "same" time and coordination of their execution is required to ensure logical correctness or safety. This is the rationale for the introduction of a *waiting rendezvous* strategy[24] in a telerobotic framework. An instruction termed cobegin(n) prefixes the invocation of parallel actions in a server, acting as a barrier for the next *n* method invocations, whose execution, therefore, does not start until all calls have reached the server. Without cobegin(n), the server schedules actions invoked by AMI requests as soon as they arrive.

### 4.2.1. Experimental Evaluation

In the experiment reported in this section, a server application controls a manipulator and the sensors which are directly related to the manipulator. The Thread Pool is set up to include three Lanes (low, medium, and high priority). Low and medium priority Lanes supply threads for the execution of requested actions. The high-priority Lane supplies threads for emergency actions, so as to guarantee their immediate dispatch. The scheduling algorithm is a Priority Level Round Robin (SCHED_RR), which is available in any POSIX-compliant operating system.

Many experiments involving simulated workload have been carried out to evaluate the correctness and robustness of the server, which has been tested with a variety of sets of concurrent actions, with different priority levels and synchronization requirements. A goal of these experiments was to verify the effectiveness of cobegin(n) in avoiding priority inversion in the execution of parallel actions. One of the experiments is described in Figure 5, showing the precedence relations, duration and priority of each method call. The correct outcome of this experiment requires that the four concurrent methods be executed according to their priority. Figure 6 compares two experimental executions of the task. Without cobegin(n) (top diagram), the medium priority action (ID 4), whose request is the first reaching the server, is executed before the high priority action (ID 1). With cobegin(n) (bottom diagram), the priority of threads is always guaranteed and no priority inversion occurs.

## 4.3. Data Distribution

A data distribution subsystem for networked robot applications should be able to efficiently exchange significant amount of data from the sensors located at the remote site (*Suppliers*) to the operators

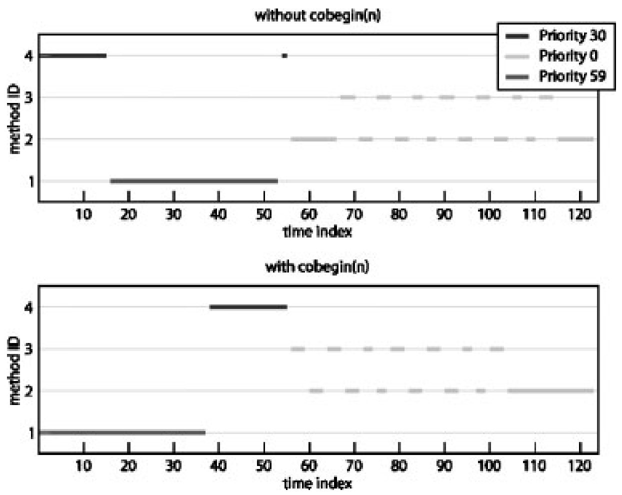

**Figure 6.** Experimental results of concurrent actions without (top) and with (bottom) cobegin(n).

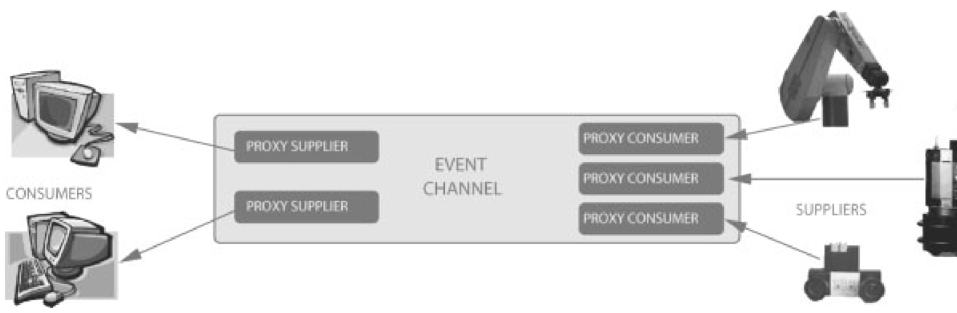

**Figure 7.** CORBA Event Service architecture.

controlling/monitoring the remote environment (*Consumers*). Moreover, it should be able to cope with the heterogeneity of consumers, its performance should scale with the number of connections, and it should minimize the load on both sides avoiding polling operations.

These requirements cannot be fulfilled by the classical Remote Procedure Call (RPC) mechanism that is, instead, suitable for transmission of control commands. Indeed, polling operations, required by RPCs, introduce well known saturation effects on both the network, due to the useless flow of requests and responses, and the Supplier, unable to answer to a large number of simultaneous Consumer requests.

Current solutions proposed by CORBA for transmission of time-critical streaming data are quite limited. A standard for Audio/Video Streaming[25] is included among CORBA specifications, but it only defines common interfaces for negotiation of the connection among distributed applications, lacking details on its use and control. Due to the weakness of this standard, most existing CORBA implementation do not take advantage of the CORBA Audio/Video Streaming specification to handle multimedia streams. It should be mentioned that OMG is currently working on a new standard to facilitate the exchange of continuous data. This new specification is now in the Draft Request for Proposal stage.[26]

A data distribution technique alternative to Audio/Video Streaming, less efficient but more portable, adopts a Publisher/Subscriber communication model.[27] Whenever the Publisher (sensor) changes state, it sends a notification to all its Subscribers. Subscribers in turn retrieve the changed data at their discretion. OMG introduced two variants of the Publisher/Subscriber communication model in the CORBA standard, the *Event* and *Notification Services*, that strongly decouple Publisher and Subscribers by means of an "Event Channel." Exploitation of these services for data distribution is investigated next.

**Event Service**. We implemented a first version of the data distribution subsystem based on the CORBA Event Service.[28] This component allows Suppliers and Consumers to exchange data without requiring the peers to know each other explicitly. The general idea of the Event Service is to decouple Suppliers and Consumers using an *Event Channel* that acts as a Proxy Consumer for the real Suppliers and as a Proxy Supplier towards the real Consumers. Therefore, the Supplier can perform a non blocking send of sensory data in the Event Channel, while the interested Consumers can connect to that channel to get the "event" (Figure 7). This implementation also allows a transparent implementation of the broadcast of sensory data to multiple Consumers.

The CORBA standard proposes four different models interleaving active and passive roles of Suppliers and Consumers. We discarded models with active Consumers as they can produce blocking communications when new data are not available at Sensor Proxy. For telerobotic applications, the only reasonable model seems to be the Canonical Push Model, where an active Supplier pushes an event towards passive Consumers registered with the Event Channel.

Despite the benefits introduced by an Event Channel, experimenting with this Service brings in several matters of discussion. Firstly, to avoid compile-time knowledge of the actual type of the "event," sensor data must be communicated as an OMG Interface Definition Language (IDL) any type, that can contain any OMG IDL data type. The communication is therefore type-unsafe and Consumers are charged with the duty of converting the any type toward the data type they need. Secondly, the Event Service specification lacks event filtering features: everything is conveyed through the Event Channel, that in turn sends everything to any connected Consumer. Therefore, the load of a missing property is laid on Consumers, that are forced to filter the whole data looking for the ones they really need. Moreover, the flow of unrequested data can again introduce the problem of network saturation. Finally, the Event Service specification does not consider QoS properties related to priority, reliability, and ordering. Attempting to ensure these properties in an application results in proprietary solutions that prevent ORB interoperability.

**Notification Service**. A second solution investigated for the data distribution subsystem is based on the CORBA Notification Service,[29] recently introduced in the CORBA Standard to overcome the limitations of the CORBA Event Service.

The Notification Service is essentially a superset of the Event Service; most components of the Event Service architecture (Figure 7) have been enhanced in the Notification Service. Notable improvements with respect to the Event Service include filtering and QoS management. In the Notification Service each Client subscribes to the precise set of events it is interested in receiving through the use of filter objects, encapsulating one or more constraints. Two filter types are defined: a *forwarding filter*, that decides whether the event can continue toward the next component, and a *mapping filter*, that defines event priority and lifetime. Moreover, QoS properties for reliability, priority, ordering, and timeliness can be associated to a Channel, to a Proxy, or to a single event.

### 4.3.1. Experimental Evaluation

The communication models described in the previous section have been evaluated in a CORBA-based telerobotic application requiring to distribute sensory data to a number of clients. The experiments reported in the following only assess the relative performance in terms of latency and scalability of the two proposed data distribution mechanisms. All experiments reported in this section follow the Push Model: a Supplier generates data and sends them to the Event Channel, when available, or directly to Consumer processes. A single Supplier and one or more Consumers, all requiring the same sensory data, are considered. Both Supplier and Consumer(s) are located on the same host, whereas the Event Channel can be on a different host.

Two host machines exploited in the experiments are listed in Table I along with their features. The hosts are connected via a Fast Ethernet switch. Unless stated otherwise, the network had no significant traffic, nor was any other processing taking place on these hosts.

In the first three sets of experiments, a 64 Byte packet is pushed by the Supplier to a single Consumer and minimum, average, and standard deviation (jitter) values of interarrival times are evaluate on a set of 50,000 samples. Consumer activity is limited to the update of the latency value so far. Figure 8 shows the average time interval between two successive 64 Byte packet receptions (interarrival time) increasing the number of Consumers from 1 to 100. Event and Notification Services results are compared with those of a Distributed Callback implementation based on the Observer patter,[30] i.e., a pattern where new sensor data, when ready, are sent by the Supplier application to all Consumers previously registered as Observers.

At the beginning of the investigated range (inset graph in Figure 8), the Callback implementation has slightly better performance than Event Channel-based ones, because it requires data to cross a lower number of hops. However, Event Service and Notification Service implementations have better scalability: they achieve a performance comparable to the Callback implementation starting from 10 Consumers, and significantly better performance starting from 70 Consumers.

**Table I.** Host features for data distribution subsystem evaluation.

| Hardware configuration | Operating system |
|---|---|
| PIV 2.4 GHz, 512 MB RAM | SuSE Linux 8.0 |
| Athlon 800 MHz, 256 MB RAM | Mandrake Linux 9.2 |

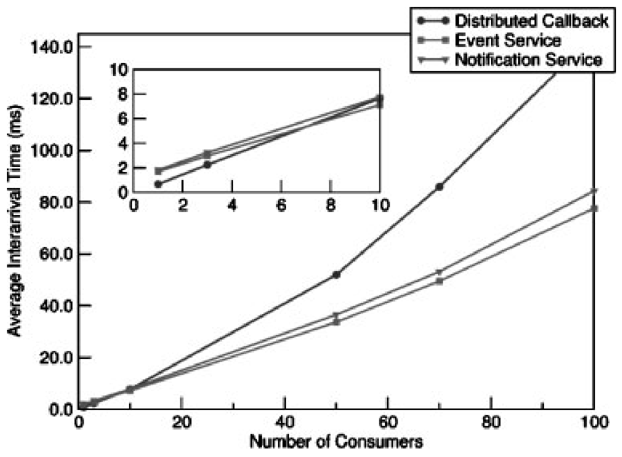

**Figure 8.** Average Interarrival Time (in ms) with a 64 Byte packet, increasing the number of Consumers. Suppliers and Consumers are on the slower machine, and Event Channel is on the faster machine.

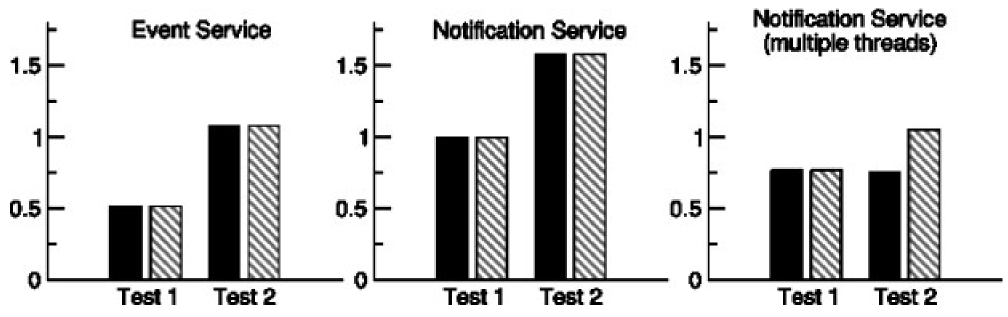

**Figure 9.** Interarrival time exploiting three different Publisher/Subscriber models. Test 1 involves two Consumers with similar characteristics; in Test 2 one of the Consumers (dashed bar) is on a slower machine.

The second set of experiments investigates synchronization among Consumers on the reception of data packets. Figure 9 shows the interarrival time for two Consumers when they are on two machines with similar characteristics (Test 1) and when one of them is on a slower machine (Test 2). In TAO standard configuration, Event and Notification Services pair together the two Consumers, forcing the packet reception rate to the one permitted by the slower one. This limitation, however, can be overcome thanks to the flexibility of the Notification Service in TAO. With proper configuration, in fact, the Notification Service deploys a thread at each proxy supplier, and Consumers are fully decoupled, as shown in the third diagram of Figure 9.

Despite the benefits achieved by decoupling Consumers, the slower one can still experience a higher delay with respect to the fast Consumer due to packet accumulation. Figure 10 shows the arrival time of 30,000 packets for two Consumers. The slow Consumer lags behind in received packets because it cannot consume them at the rate at which they are supplied.

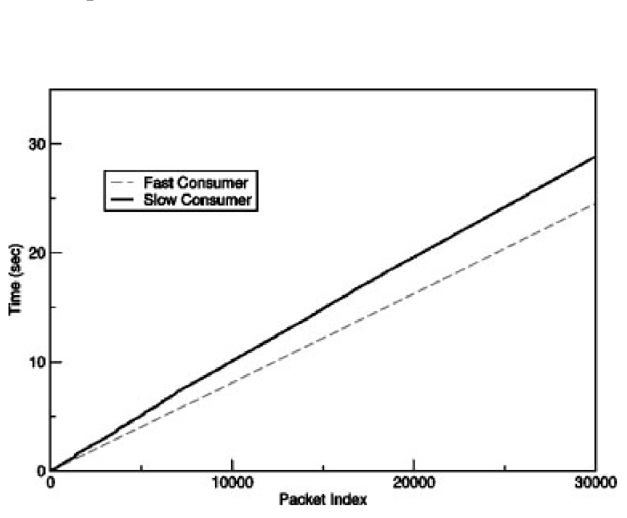

**Figure 10.** Cumulative packet arrival times for two decoupled Consumers with different performance.

The problem of the increased delay for slow Consumers can be overcome using QoS properties of the Notification Service. When the size of the buffer queue is set to one and a policy always discarding the older packet is applied, packets received by the two Consumers show an almost constant, small delay, as shown by the two parallel lines in Figure 11. In the experiment in Figure 11, the slow Consumer is further hampered by a heavy network load, purposely introduced to stress the features of the Notification Service. The drawback in the use of this setting is the reduction in the number of packets received by the slower Consumer. As shown in the inset graph in Figure 11, due also to the high network load only one packet on 500 is received by the slow Consumer.

To summarize, for robot servers performing several complex tasks (e.g., sensor data acquisition and distribution, motion planning, actuator control) and dealing with a large number of attached clients, the Event Channel represents an effective tool to maintain the overall workload under control. When Clients have different QOS needs and at the expense of a slight overhead, the Notification Service is the most appropriate solution, thanks to its configurability options not available in Callback and Event Service implementations.

### 4.4. Security

Security is a main problem in the development of distributed systems communicating through a shared channel, more vulnerable to intrusion than traditional systems. To guarantee multilateral security, OMG has defined a quite large and complex standard, the Security Service Specification,[31] trying to

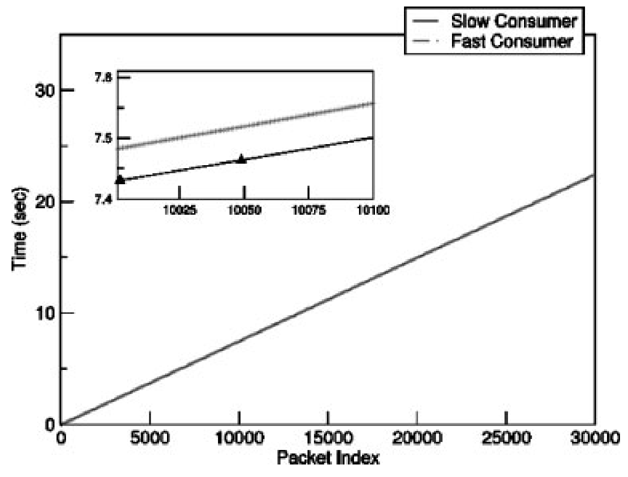

**Figure 11.** Cumulative packet arrival time for two decoupled Consumers with different performance when the buffer size is set to one and older packets are overwritten.

encompass several solutions proposed by its members. A goal of the standard is to allow the implementation of the application components independently from the concrete security mechanisms, encapsulated in a small trusted core that cannot be bypassed by *principals*, i.e., human users or objects operating in the system.

The TAO implementation of the security service focuses on enforcing the *security attributes* of the *principals* through the use of SSLIOP, an implementation of IIOP over SSL. Identification and authorization of principals are achieved using certificates issued by a Certification Authority on registration phase. Once that the secure connection from a recognized principal is accepted, the server controls its *privilege attribute* before allowing access to the target object. It is the server that defines the levels of privileges required to access its robotics components and, at runtime, grants authorizations on the base of the certification owned by principals.

#### 4.4.1. Experimental Evaluation

The main goal of this evaluation session is the analysis of the performance penalty introduced by the Security Service. All the experimental results refer to a client-server application with client authentication and authorization. Real-Time CORBA features could not be used because a bug in the current TAO release makes its real-time features incompatible with SSLIOP. This fact prevented the possibility to exploit both real-time and Security TAO features in our experiments. (The bug has been signalled to TAO developers and should be corrected in the next release.) Table II reports the overall wall clock time required to push 50,000 times a 64 Byte packet from the Supplier to a single Consumer. Consumer activity is negligible. Experimental results show that the use of SSLIOP greatly affects performance of the system, with an overhead that can reach 86%. Therefore exploitation of the Security service should be limited at the invocation of sensitive actions on target objects, such as for commands affecting robot movements.

## 5. A CORBA-BASED TELEROBOTIC SYSTEM

In the previous section, CORBA Standard and Services relevant to telerobotic applications have been introduced and evaluated using microbenchmarks. The aim of this section is, instead, to describe and assess a complete telerobotic application, based on the framework integrating various CORBA services.

### 5.1. The Application

The telerobotic system is illustrated in Figure 12. The remote site includes an Unimation PUMA560, a six d.o.f. robot arm, whose controller is interfaced to a Pentium-based workstation running the RCI/RCCL robot programming and controlling environment. Mounted on the arm is a parallel gripper, pneumatically actuated with a maximum opening of 80 mm; the gripper is provided with a binary sensor able to detect its closure. The same host controlling the arm performs as a supplier for sensory data generated by an eye-on-hand microcamera and an IR proximity sensor positioned near the gripper. The system features additional sensoriality, including a video cam-

**Table II.** Overhead introduced by the Security Service when a 64 Byte packet is pushed by a Supplier to a single Consumer for 50,000 times.

| Supplier | Consumer | without SSLIOP | with SSLIOP | (% Overhead) |
|---|---|---|---|---|
| *Faster Machine* | *Faster Machine* | 9.270 | 17.310 | 86% |
| *Faster Machine* | *Slower Machine* | 30.430 | 51.270 | 68% |
| *Slower Machine* | *Faster Machine* | 36.790 | 52.270 | 42% |

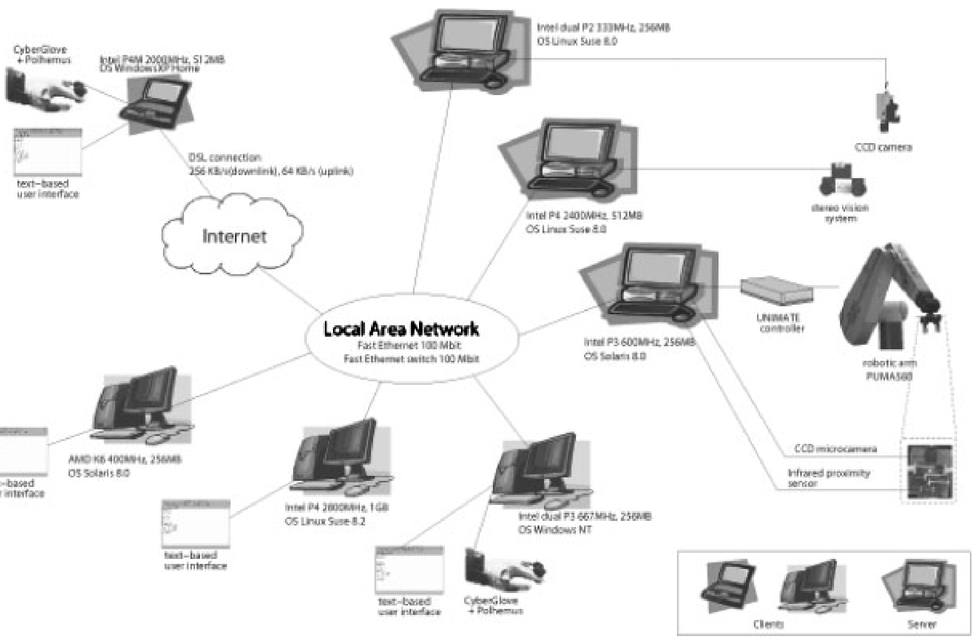

**Figure 12.** The experimental testbed.

era mounted on the ceiling and shooting the testbed area and a stereo vision system in front of the robot workspace.

Clients are connected to the remote robot either with a local (Ethernet LAN) or a geographical network (DSL connection with 256KB/s downlink and 64KB/s uplink).

To allow the clients to interact with the remote environment, program and monitor the task, each client is provided with a set of applications. Using a Graphical User Interface (GUI) the operator can dynamically configure his environment, adding new sensoriality and moving available hardware. The inset graph in Figure 13 shows the GUI with windows displaying sensory information received from the IR Proximity Sensor, the eye-in-hand camera, and one image of the stereo camera.

Authorized clients can submit motions to the remote robot either defining a sequence of single commands or writing a program in a C-like language. More advanced Client stations also support the control of the robot using an 18-sensor CyberTouch (a virtual reality glove with tactile feedback from Immersion Corp., Inc.) and a six degree of freedom Polhemus tracker (Figure 13).

Several telerobotic manipulation tasks, including pick and place, stacking of objects, and simple peg-in-hole operations, have been implemented using the system.

## 5.2. Performance Analysis

The experimental results in this section refer to a manipulation task requiring the operator to stack three blocks (Figure 14). The clients can move the robot either using a textual interface in a step-by-step control or using the virtual glove and tracker. While performing the task, the system returns to connected users the visual information generated by a camera shooting the workspace and by the eye-in-hand camera.

In the first test four clients participate to the execution of the stacking manipulation task. Clients are connected to the robotic server through a Fast Ethernet switch with negligible latency. The main goal of the experiment is to investigate the reliability of the mechanisms for concurrency among clients described in Section 4.1.

Figure 15 traces the main environmental values during a task section. The first row shows the evolution of the distance between the robotic gripper and the table (Z), sampled at 1 Hz, during the execution of the manipulation task. Management of concurrent client access is described by the next four rows of Figure 15. Each client can go across four states: exclusive control, collaborative control, waiting for control, observing. Waiting slots are experienced by clients asking for control when the robot is already controlled in an incompatible mode by other client(s). The last row of Figure 15 shows the total number of clients cur-

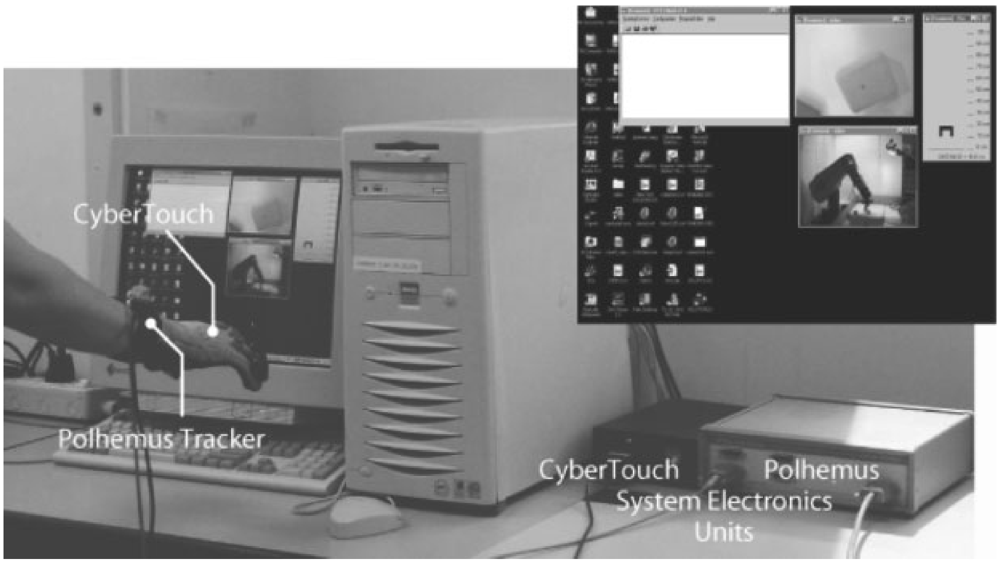

**Figure 13.** An example of the Client Setup and a snapshot of the output with windows displaying sensory information received from the remote site.

rently connected to the system including slot time when they are connected as "observer."

Several task sessions demonstrated the reliability of the concurrency mechanisms. The task was executed multiple times, and the presence of multiple clients controlling the robot never prevented their successful completion.

In the second test, a single client is connected to the robot server through a wide area network (WAN). When the robot is located at a remote site accessed through the Internet, its control becomes problematic due to communication delays. The higher latency prevents real-time user interaction, and the variable network load changes the time required to update sensory data at the clients.

The experimental performance observed in this test was mainly influenced by the time delay and frame rate of the visual information received from the two cameras using the CORBA-based data distribution mechanisms integrated in the application (latency ranging from 2 to 5 seconds, frame rate below 0.5 fps). To improve the visual feedback, a data channel based on RTP on top of a UDP/IP connection was set up outside the CORBA infrastructure. While this setting requires that client and server agree on the use of non-CORBA communication, it is currently the only solution for efficient streaming of video images over the Internet, since current CORBA support for video streaming is too limited. The specific RTP data channel allowed a time delay around 2 seconds and a frame rate around 10 fps with a compressed video stream at a resolution of 160×120. The drawback of this solution was of course the lower quality of the image due to the lossy compression.

As reported by Hashimoto et al.,[32] the limit for psychological presence, i.e., the time range which is perceived as present when phenomena happen continuously, is around two to three seconds. When the time is longer human captures the period internally by evaluating or estimating the phenomena that occur.

Our experience confirms that delays larger than two seconds do not allow the operator to move continuously the objects. Instead, objects are moved in a stop-and-go way: after each movement the operator controls the results of the command and the state of the objects in the workspace before issuing the next command. Depending on the task phase, the operator uses alternatively the information received from the sensors. While approaching the grasping or releasing of objects, the operator prefers to use the eye-in-hand camera, which returns more accurate information about the gripper-object alignment. When ample movements are required, the position of the robot arm is more important and the operator prefers snapshots received from the RTP streaming.

Figure 16 shows the distance between the gripper and the table during two teleoperated assembly tasks. The top diagram shows the gripper height during a task performed using the virtual reality glove and 3D tracker. The bottom diagram shows the same data obtained using the textual interface. Comparing

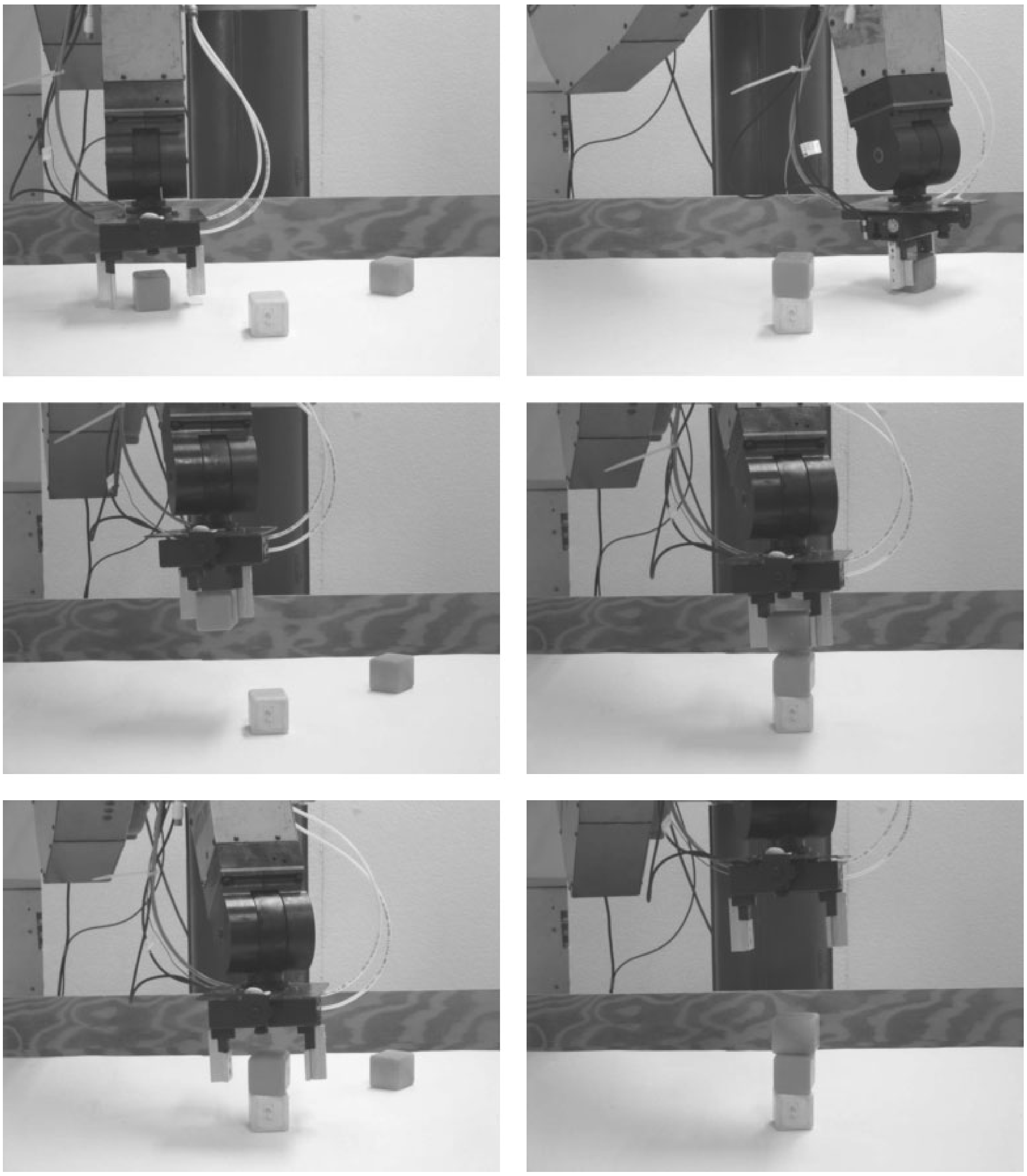

**Figure 14.** A stacking manipulation task (left to right, top to bottom).

the two diagrams, we can observe that using the textual interface the task was performed in a few steps interleaved by long time periods; instead, using the virtual reality glove the task was performed more quickly, although control was less precise.

Comparison between these graphs and the first graph in Figure 15 shows that manipulation task executed upon a WAN drastically increases the time required to stack the cubes. This can lead to operator fatigue, an important factor as a major source of er-

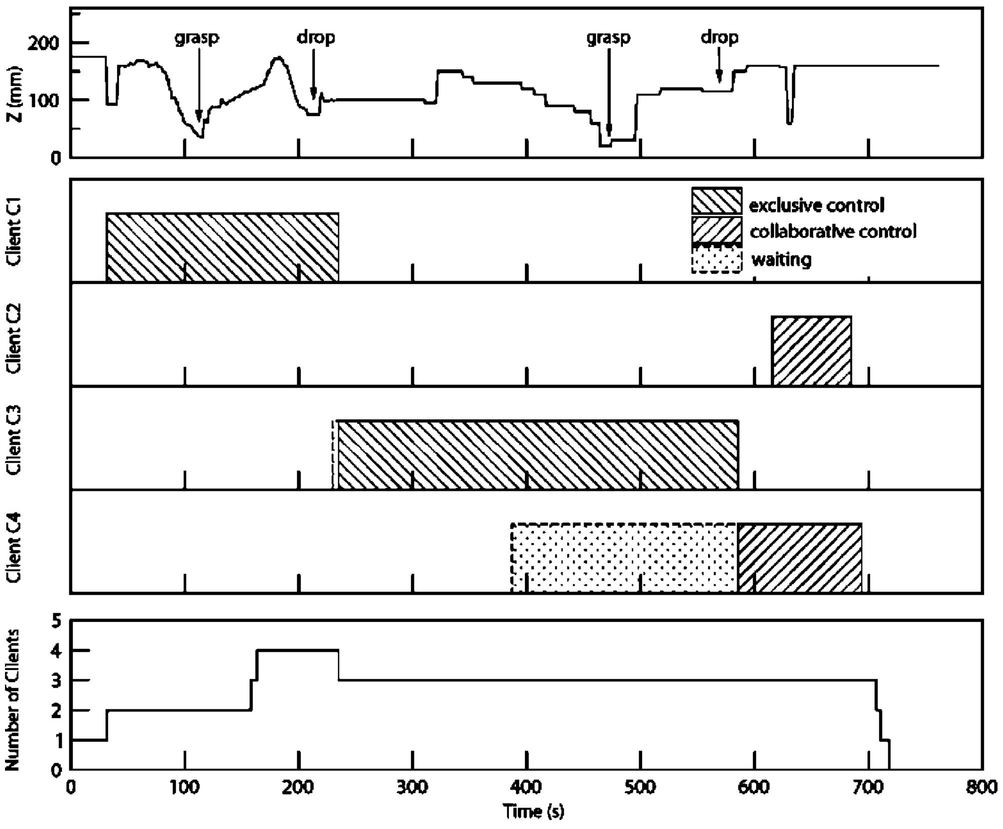

Figure 15. Tracing of a teleoperation experiment involving four clients that concurrently access the robot arm. Top diagram: vertical height of the gripper. Remaining diagrams: client access policy information.

rors. Nevertheless, several task sessions successfully executed demonstrate the possibility to manipulate objects remotely on a WAN using the proposed telerobotic architecture.

## 6. RELATED WORK

Our work relates to the area of Internet-based telerobotics, whose aim is to build flexible, cheap, dynamic, heterogeneous distributed telerobotic systems and applications. A broad perspective on these applications is given in the collection.[1] The main issue in many of these projects is the interaction with web users who, lacking technical skills, require easy-to-use command interfaces.

Other research views Internet-based telerobotics as *distributed robotic systems*,[5] addressing the issues arising in the implementation of Client/Server systems.

A few recent papers are concerned with teleoperation of multiple robotic devices by one or multiple users.[34–36] A taxonomy for classifying various kinds of teleoperation systems based on number of robots and operators is proposed in ref. 19. In ref. 37 a modeling and control method for Internet based cooperative teleoperation is proposed. This method combines Petri Net model and event-based planning and control theory to face the complexity of cooperative te-

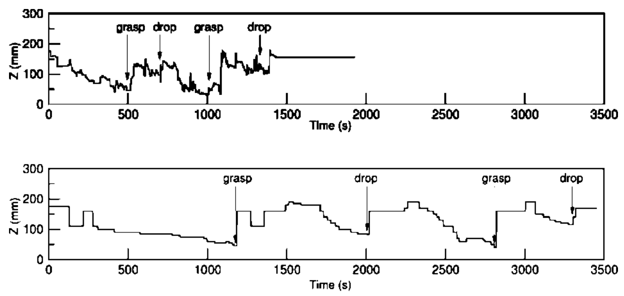

Figure 16. Distance between the gripper and the table during assembly tasks teleoperated over the Internet. Continuous control with virtual reality glove and 3D tracker (top) and step-by-step control with textual interface (bottom).

leoperation involving geographically distributed sites, with multiple operators and robots, connected through the Internet.

A fundamental problem arising teleoperating a robot over the Internet, but also over any TCP/IP based network, is the unpredictable time delay of communications.[38,39]

Internet based teleoperation enhanced by a collection of sensorial feedback (video, audio, haptic information, temperature, and other) and experimented with mobile robots and mobile manipulators is presented in ref. 40.

A few papers exploit the interoperability and location transparency provided by CORBA to ease system implementation in applications such as a distributed laboratory,[41] a supervisory control scheme,[42] or an Internet telerobotic system conceived to provide assistance to aged and disabled people.[43,44]

Two papers are more directly concerned with the implementation of systems supporting distributed telerobotic applications. Hirukawa and Hara[2] propose a framework based on OO programming for robot control, whereas Dalton and Taylor[5] advocate nonblocking asynchronous communications, viewed as essential to build a distributed robotic systems. Since this feature was not available in the CORBA implementation they relied upon, the architectural framework in ref. 5 exploited nonstandard middleware. A recent work describes a virtual laboratory accessible through the Internet[33] developed upon an architecture based on the recent CORBA Components Model (CCM).

Our research departs from this prior art in several respects. Our telerobotic framework exploits COTS middleware not merely for interoperability or location transparency, but taking full advantage of its multithreading and real-time features. No previous work in the area has used the Asynchronous Method Invocation model, even though an asynchronous interface is deemed an essential feature.[5] Now that RT CORBA technology has matured, it can be leveraged upon to develop reliable COTS-based telerobotic systems with strict control over computational resources.

## 7. CONCLUSIONS AND DISCUSSION

The viability and cost effectiveness of new telerobotic applications such as virtual laboratories, networked and on-line robots can be widely enhanced exploiting COTS-based software components. Moreover, systems implementing those applications pose also demanding challenges: they should be dynamically reconfigurable and highly scalable to deal with a potentially large number of peers, they should provide real-time features, guaranteed performance and efficient concurrency mechanisms, both locally and in a distributed environment. The CORBA middleware, especially with its recent extensions, has proven well suited for the needs of many distributed telerobotic systems. In this paper we have summarized our experience resulting from development and application of a software framework for telerobotics based on Real-Time CORBA, ensuring proper real-time operation of the server and managing concurrent control and data distribution with multiple Clients.

The results obtained show that exploitation of CORBA brings a number of remarkable advantages in the telerobotic domain, enabling portable, highly reconfigurable applications with support for concurrency and real-time features. Furthermore, CORBA standard services for naming resolution, data distribution and concurrency control avoid the need for *ad hoc* solutions, which are often error prone, require significant development effort, and prevent portability.

Of course, providing transparency and high level services in CORBA is not without cost. The overhead in communication and in services such as the security service, along with the limited performance and reliability of available network infrastructures, may prevent exploitation of a CORBA-based architecture in critical telerobotic applications, unless suitable control and predictive simulation techniques are also put into operation. We expect that with the scaling of network and Internet infrastructures to higher levels of performance and QoS guarantees, such performance limitations will become less relevant. Additional drawbacks encountered in our experience stem from the incompleteness in the implementation of the CORBA standard suite. None of the available CORBA ORBs offers a full implementation of the CORBA standard, i.e., covering aspects such as dynamic scheduling, fault-tolerance, fully compliant CORBA services. Furthermore, some extensions of the standard would be useful, e.g., higher-level synchronization mechanisms (condition variables, barriers) and more effective and useable services for streaming and continuous media management. These deficiencies, however, are widely acknowledged by OMG and in the CORBA standard community, and for some of them solutions are in the making.

## ACKNOWLEDGMENTS

Work on this paper has been supported by CNR and MURST, Italy, project "Sistema multiagente con componenti fisse e robotiche mobili," under the program funded by L. 449/97.

## REFERENCES

1. K. Goldberg and R. Siegwart (Editors), Beyond Webcams: an Introduction to Online Robots, MIT Press, Cambridge, 2001.
2. H. Hirukawa and I. Hara, Web-Top Robotics, IEEE Rob Autom Mag 7(2) (2000), 40–45.
3. T. Hori, H. Hirukawa, T. Suehiro, and S. Hirai, Networked Robots as Distributed Objects, in IEEE/ASME Int Conf on Advanced Intelligent Mechatronics, 1999.
4. K. Goldberg, S. Gentner, C. Sutter, and J. Wiegley, The Mercury Project: A Feasibility Study for Internet Robots, IEEE Rob Autom Mag 7(1) (2000), 35–39.
5. B. Dalton and K. Taylor, Distributed Robotics over the Internet, IEEE Rob Autom Mag 7(2) (2000), 22–27.
6. H. Hirukawa, I. Hara, and T. Hori, Online robots, in Beyond Webcams: an introduction to online robots, K. Goldberg and R. Siegwart (Editors), MIT Press, Cambridge, 2001.
7. A. Bicchi, A. Coppelli, F. Quarto, L. Rizzo, F. Turchi, and A. Balestrino, Breaking the Lab's Walls: Tele-Laboratories at the University of Pisa, in IEEE Int Conf on Robotics and Automation, 2001.
8. WS2001: International Workshop on Tele-Education in Mechatronics Based on Virtual Laboratories, Weingarten, Germany, 2001.
9. P. Backes, K. Tso, and G. Tharp, Mars Pathfinder Mission Internet-Based Operations using WITS, in IEEE Int Conf Robotics and Automation, 1998.
10. R. Kress, W. Hamel, P. Murray, and K. Bills, Control Strategies for Teleoperated Internet Assembly, IEEE/ASME Trans Mechatron 6(4) (2001), 410–416.
11. T.B. Sheridan, Telerobotics, Automation, and Human Supervisory control, MIT Press, Cambridge, MA, 1992.
12. C. Zielinski, Object-Oriented Robot Programming, Robotica 15(1) (1997).
13. 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.
14. M. Amoretti, S. Bottazzi, M. Reggiani, and S. Caselli, Evaluation of Data Distribution Techniques in a CORBA-based Telerobotic System, in IEEE Int'l Conf Intelligent Robots and Systems, 2003.
15. C. Sayers, Remote Control Robotics, Springer, Berlin, 1999.
16. Y. Tsumaki, T. Goshozono, K. Abe, M. Uchiyama, R. Koeppe, and G. Hirzinger, Verification of an Advanced Space Teleoperation System using Internet, in IEEE/RSJ Int'l Conf on Intelligent Robots and Systems, 2000.
17. The Common Object Request Broker: Architecture and Specification Revision 2.4, Object Management Group, 2000.
18. Distributed Object Computing (DOC) Group, Real-time CORBA with TAO (The ACE ORB), http://www.ece.uci.edu/~schmidt/TAO.html.
19. N. Chong, T. Kotoku, K. Ohba, K. Komoriya, N. Matsuhira, and K. Tanie, Remote Coordinated Controls in Multiple Telerobot Cooperation, in IEEE International Conference on Robotics and Automation, 2000.
20. Object Management Group, Concurrency Service Specification, http://www.omg.org/technology/documents/formal/concurrency_service.htm, 2000.
21. Real-Time CORBA Revision 1.1, Object Management Group, 2002.
22. A. Arulanthu, C. O'Ryan, D. Schmidt, M. Kircher, and J. Parsons, The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging, in Proc of the Middleware 2001 Conference ACM/IFIP, 2000.
23. The Common Object Request Broker: Architecture and Specification Revision 3.0, Object Management Group, 2002.
24. B. Douglass, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison-Wesley, Reading, MA, 1999.
25. Audio/Video Stream Specification, Object Management Group, 2000.
26. O.M. Group, Streams for CORBA components, Request for Proposal.
27. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, A System of Patterns, Wiley and Sons, New York, 1996.
28. Object Management Group, Event service specification, v. 1.1, http://www.omg.org/technology/documents/formal/event_service.htm, 2001.
29. Notification service specification, v. 1.0.1, http://www.omg.org/technology/documents/formal/notification_service.htm, 2002.
30. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns—Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
31. CORBA Security Service, Version 1.8, Object Management Group, 2002.
32. H. Hashimoto, N. Ando, and J.-H. Lee, The Performance of Mobile Robots Controlled through the Web, in Beyond Webcams: an introduction to online robots, K. Goldberg and R. Siegwart (Editors), MIT Press, Cambridge, MA, 2001.
33. E. Guimares, A. Maffeis, R. Pinto, C. Miglinski, E. Cardozo, M. Bergerman, and M. Magalhaes, REALA Virtual Laboratory Built from Software Components, in Proc IEEE 91(3) (2003).
34. K. Goldberg, D. Song, Y. Khor, D. Pescovitz, A. Levandowski, J. Himmelstein, J. Shih, A. Ho, E. Paulos, and J. Donath, Collaborative Online Teleoperation with Spatial Dynamic Vonting and a Human Tele-Actor, in IEEE International Conference on Robotics and Automation, 1997.
35. K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, and G. Smith, Collaborative Teleoperation via the Internet, in IEEE International Conference on Robotics and Automation, 2000.
36. P. Rybski, S. Stoeter, N. Papanikolopoulos, I. Burt, T. Dahlin, M. Gini, D. Hougen, D. Krantz, and F. Nageotte, Sharing Control, Rob Autom Mag 9(4) (2002), 41–48.

37. I. Elhajj, N. Xi, W. Fung, Y. Liu, Y. Hasegawa, and T. Fukuda, Modeling and Control of Internet Based Cooperative Teleoperation, in IEEE International Conference on Robotics and Automation, 2001.
38. W. Hamel and P. Murray, Observations Concerning Internet-based Teleoperations for Hazardous Environments, in IEEE International Conference on Robotics and Automation, 2001.
39. W. Fung, N. Xi, W. Lo, and Y. Liu, Improving Efficiency of Internet Based Teleoperation Using Network QoS, in IEEE International Conference on Robotics and Automation, 2002.
40. I. Elhajj, N. Xi, W. Fung, Y. Liu, Y. Hasegawa, and T. Fukuda, Supermedia-Enhanced Internet-Based Telerobotics, in Proc IEEE 91(3) (2003).
41. C. Paolini and M. Vuskovic, Integration of a Robotics Laboratory using CORBA, in IEEE Int Conf Systems, Man, and Cybernetics, 1997.
42. R. Burchard and J. Feddema, Generic Robotic and Motion Control API Based on GISC-Kit Technology and CORBA Communications, in IEEE International Conference on Robotics and Automation, 1997.
43. S. Jia and K. Takase, An Internet Robotic System based Common Object Request Broker Architecture, in IEEE Int Conf Robotics and Automation, 2001.
44. 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.