Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada · October 2003

Evaluation of Data Distribution Techniques in a CORBA-based Telerobotic System

Michele Amoretti, Stefano Bottazzi, Monica Reggiani, Stefano Caselli
RIMLab - Robotics and Intelligent Machines Laboratory
Dipartimento di Ingegneria dell'Informazione,
University of Parma,
Email: {amoretti,bottazzi,reggiani,caselli}@ce.unipr.it

Abstract— **Distributed telerobotic applications exploiting Internet-related technologies, such as virtual laboratories and on-line robots, require effective techniques for timely delivery of sensory data to remote clients. In these systems, there is a need to distribute increasing quantities of sensory data to a potentially large number of clients during system operation. In this paper, we describe and evaluate three implementations of a sensory data distribution subsystem in the context of a CORBA-based framework for telerobotic applications. Experimental results show that solutions exploiting CORBA Services and based on the Event Channel paradigm represent a viable alternative to ad-hoc solutions. The overhead associated to CORBA Services becomes less significant with larger message size. Moreover, these services ensure portability, extensibility, reduction in programming complexity, and improved scalability when the number of clients increases.**

I. Introduction

Distributed computing systems and Internet-related technologies have opened new application perspectives to robot teleoperation systems. Examples of novel applications, often broadly termed as "networked" or "on-line" robot systems [1], are tele-teaching/tele-learning, virtual laboratories, remote and on-line equipment maintenance, and projects requiring collaboration among remote users, experts, and devices [1]–[3]. These applications, while providing new opportunities, propose alternative challenges with respect to the traditional approach to robot teleoperation [4], [5], which is based on a dedicated system architecture ensuring tight coupling of a master device and a slave robot by means of a dedicated connection.

Indeed, many networked telerobotic applications 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, based on hardware acquired from multiple vendors, running different operating systems and programmed in a variety of languages. New telerobotic applications tend also to be very dynamic, with sensor and robot controllers dynamically connected to the network (e.g., an equipment available only part-time in a virtual laboratory), resulting in changes in the number and location of peers, in variable service roles, and in the need of load reconfiguration to improve distributed system performance. Clients can also register or disconnect at runtime and bid for commanding or supervisory roles, leading to a variety of dynamic interaction patterns.

In spite of their specific features, telerobotic applications often share common requirements about the Server, Client, and communication structures. The costly development of a new application from scratch can thus be avoided relying on previous experience or, even better, on a common framework. To this end, telerobotic systems can take advantage from the developments in distributed systems research. In modern distributed systems design, open, reconfigurable, and scalable architectures can be built using commercial off-the-shelf (COTS) components. Moreover, the object-oriented (OO) design methodology provides fundamental concepts such as inheritance, polymorphism, and hiding, useful in the development of complex distributed COTS-based applications [6]. In this area, exploitation of middleware software between low-level APIs and Client/Server applications is a major approach pursued to cope with dynamic and flexible applications. Several established middleware implementations are based on the Common Object Request Broker Architecture (CORBA) (`http://www.corba.org`), a vendor-independent specification promoted by the Object Management Group (OMG) (`http://www.omg.org`). CORBA overcomes the heterogeneity problem arising also in networked robotic systems, since it allows interoperability of systems built using different software technologies, programming languages, operating systems, and hardware.

A software framework for distributed telerobotic systems exploiting advanced CORBA features was presented in [7] and shown to improve flexibility, portability, openness, extensibility, and reusability of the application. According to our experience, the current CORBA specification and its extensions reduce the programming effort of complex applications, hiding to developers details regarding communication among distributed objects. Two extensions of the CORBA standard particularly relevant for telerobotic applications are Asynchronous Method Invocation (AMI) and Real-Time CORBA (RT CORBA). They support, in a portable way, functionalities which are mandatory in Servers controlling robots and sensory systems, such as proper management of request priorities and preemptable Server actions.

A distinct trend in networked robot systems is the need to distribute increasing quantities of sensory data to a potentially large number of Clients during system operation. The very

0-7803-7860-1/03/$17.00 © 2003 IEEE

concept of networked robot aims at enabling a larger and dynamic set of users, which nonetheless must be provided with sufficient data for their real-time interaction with remote systems. In a telerobotic task, the timely availability of adequate sensory data to emulate the operator's physical presence at the remote site is particularly crucial [8], [9]. The issue of sensory data distribution was not specifically dealt with in the proposed framework [7], but clearly manifested itself as a problem in later experimentation involving more data-intensive tasks [10].

The classic Remote Procedure Call [11] found in most Object Request Brokers (ORBs) is unsuitable for applications involving a large number of remote sites that exchange significant amounts of data, because it requires a polling operation that introduces saturation effects on both the network and the Server. Therefore, we identify the following guidelines in the development of a data distribution subsystem for networked robot applications:

- It should be able to cope with the heterogeneity of Clients. For example, Clients can choose to receive data from specific sensor subsets; also, their network connection can range from high speed LANs to slow modems.
- It should be scalable. When additional Clients connect to the system and submit their requests, the performance should degrade only gracefully for Clients at same priority, whereas higher priority Clients should maintain their current perceived performance.
- It should transparently distribute data to Clients without requiring the Server to be aware of the number, locations, and platforms of Clients.
- It should avoid polling and minimize the load of data distribution on both Servers and Clients.

In this paper, after a brief description of the CORBA-based framework for teleoperation (Section II), we describe three communication models that have been implemented by means of CORBA Services to support data distribution to dynamically connecting Clients (Section III). We next report the experimental results obtained by the communication models (Section IV), and finally summarize the contributions of the paper (Section V).

II. A FRAMEWORK FOR TELEOPERATION

A middleware such as CORBA offers a set of tools for connecting objects across heterogeneous processing nodes, thereby simplifying the development of distributed applications, but does not implement an application by itself. Development of a telerobotic application, which is not a standard Client/Server system, remains a fairly demanding task. To relieve this task, we implemented a software framework as a tool to face common requirements of telerobotic applications. The framework exploits a commercial off-the-shelf standard from distributed systems, namely Real-Time CORBA [7]. The main achievement of the framework is a multithreaded Server with concurrency mechanisms simplifying sharing of CPU among computation and communication services, dealing with Client requests preserving their ordering, and exhibiting different kinds of reaction depending on their urgency. The Server operates in real-time to allow implementation of the appropriate control laws with guaranteed operation. Finally, it provides synchronization mechanisms for exclusive allocation of non-sharable resources. To avoid dedicated solutions or proprietary features that prevent portability, the Server implementation takes advantage from recent extensions of Real-Time CORBA and CORBA Messaging [12] that provide standard APIs for multithreading, synchronization, and asynchronous calls.

The effectiveness of the framework has been assessed in the concurrent execution of simple telerobotic tasks with

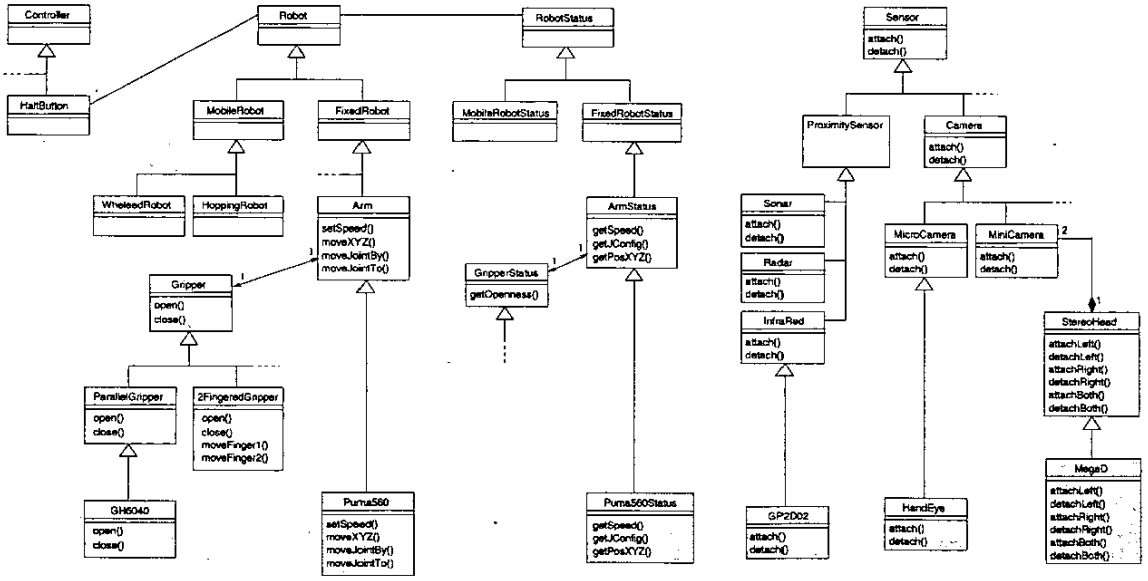

Fig. 1. Class Diagram of a peg-in-hole task.

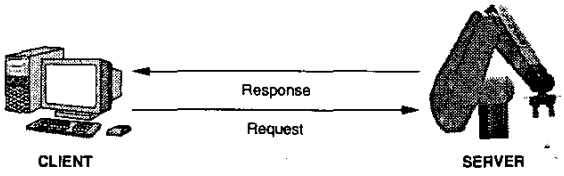

Fig. 2. Client/Server model for data exchange.

specific priority levels [7]. The Server was able to provide differentiated service to these requests based on their portable priority attributes. Additional experiments investigated the completeness, modularity and flexibility of the framework by implementing a peg-in-hole task in the context of a complete telerobotic application with rich and heterogeneous sensory data returned to multiple Clients [10]. Location transparency provided by CORBA and portability of the framework to a number of operating systems allowed exploitation of the same code in several Client stations and reallocation of sensoriality among heterogeneous Server stations. The class diagram in Figure 1 shows the use of the framework in the development of two Server applications required in the peg-in-hole task: one for manipulator control and local sensoriality, and the other for distribution of video images from a stereo camera system.

As previously mentioned, our teleoperation framework did not include specific mechanisms to support efficient distribution of sensory data to a large number of Clients in a scalable way. The development and implemention of these mechanisms is the contribution of this paper.

III. COMMUNICATION MODELS

The Client/Server model is the most common communication method in distributed object computing: a Client invokes an operation on a target object implemented at the Server side and then synchronously awaits to receive the response (Figure 2). Drawbacks of this model are the inactivity of the Client while waiting for a response, and the point-to-point synchronous communication, with a strong Client/Server relationship that requires the Server to be always available. Partial solutions, such as implementing multithreading to avoid blocking of the Client, usually lead to a higher programming complexity.

In data distribution applications, the Client/Server model introduces even more difficulties. Indeed, as the server is not able to asynchronously reply to Clients and no group communication is supported, Clients must continuously poll the remote resource for new data. Polling operations introduce well known saturation effects on both the network, due to the useless flow of requests and responses, and the Server, unable to answer to a large number of simultaneous Client requests.

More suitable for interactions among peers is the Publisher/Subscriber communication model [13]. 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

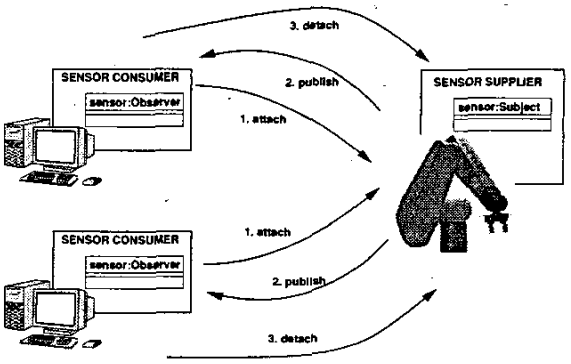

Fig. 3. Distributed callback for data distribution.

standard, the *Event* and *Notificaton Services*, that strongly decouple Publisher and Subscribers by means of an "Event Channel". Exploitation of these services for data distribution is investigated next, along with a Callback-based technique.

A. Distributed Callbacks

To avoid polling operations and minimize network saturation, we implemented a sensor data distribution subsystem based on Distributed Callbacks [11]. Following to the Observer pattern, we defined two CORBA classes for each available sensor: the *subject* at the Server side and the *observer* at the Client side (Figure 3). To receive data from a sensor, the Client application calls a method *attach* on the remote *subject* object, passing a reference to an *observer* object. Each sensor holds a list of all *observer* objects that have been attached. When new sensor data are ready, they are sent by the Server application to all the "attached" *observer* objects, through the invocation of the appropriate method.

In this solution, and in the following ones, the peers involved in the communication do not exhibit the Client/Server relationship anymore, therefore a more suitable terminology defines *Supplier* the peer producing the sensor data, and *Consumer* anyone who receives them.

Though the Supplier/Consumer approach avoids Client active waiting and network saturation, new problems spawn at the Supplier side. When thousands of Consumers are attached, the Supplier is supposed to persistently store thousands of references and send a separate message to each in turn according to their preferences (each Consumer should be able to define the desired data receiving rate to avoid unnecessary distribution of data). In this scenario the efficiency of the Supplier is greatly affected by the number of Consumers. Therefore, due to the high memory and computation requirements, scalability is bounded to a relatively small number of Consumers.

B. Event Service

To relieve the Supplier of administrative duties related to Consumers management, we implemented a second version of the data distribution subsystem based on the CORBA Event Service [14]. This component allows Suppliers and Consumers to exchange data without requiring the peers to know each

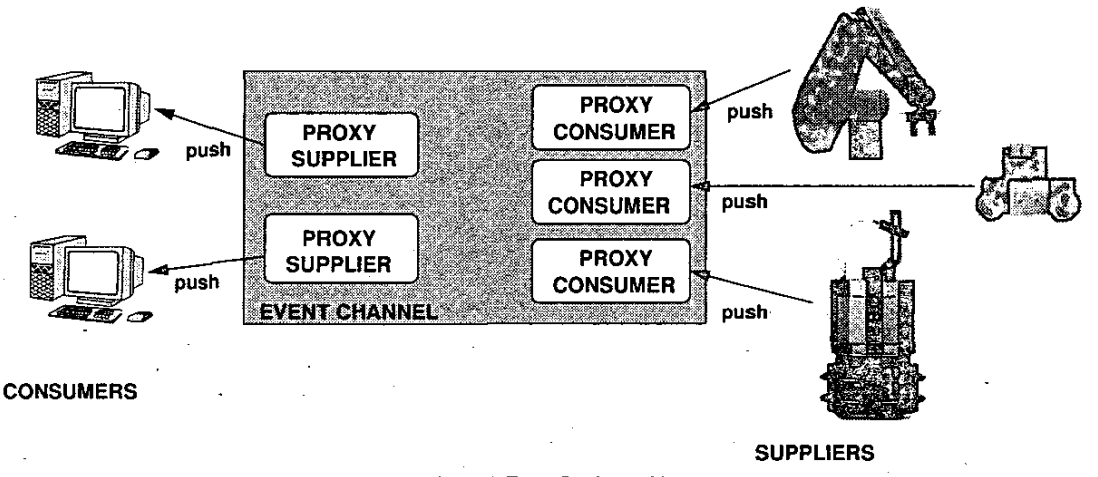

Fig. 4. CORBA Event Service architecture.

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 the sensor data in the Event Channel, while the interested Consumers can connect to that channel to get the "event" (Figure 4). This implementation also allows a transparent implementation of the broadcast of sensor 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 robotic 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. Once again, the load of a missing property is laid on the 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.

C. Notification Service

Our third solution for the distributed data subsystem is based on the CORBA Notification Service [15], recently introduced in the CORBA Standard to overcome the previously listed deficiencies of the CORBA Event Service.

The Notification Service is a superset of the Event Service; most components of the Event Service architecture (Figure 4) 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.

IV. Empirical Performance Evaluation

The communication models described in Section III have been integrated in our CORBA-based framework for telerobotic systems. The implementation of the framework is written in C++ and based on "The ACE ORB" (TAO) [16], a freely available, open-source, and standard-compliant Real-Time CORBA implementation. We remark that the fundamental advantages brought by a CORBA-based framework for teleoperation are portability, flexibility, openness and reusability of code. These advantages, which cannot be quantified in an experimental section, are now retained also for the sensory data distribution subsystem. The experiments reported in this section only attempt to assess the relative performance in terms of latency and scalability of the three proposed data distribution mechanisms. It should be obvious that the Event and Notification Services provide additional features not exploited in the experimental assessment but very useful with many Consumers and general patterns of Consumer requests.

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. Aside from our tests, the network had no significant traffic nor was any other processing taking place on these hosts.

Host name	Hardware configuration	Operating system
Trovatore	PIV 2.4GHz, 512MB RAM	SuSE Linux 8
Puma	PIII 600MHz, 256MB RAM	Solaris 8

TABLE I

EXPERIMENTAL SETUP: HOST FEATURES.

In the remaining of this section we will identify the implementations of the data distribution subsystem with the following abbreviation:

A based on Distributed Callbacks;
B based on CORBA Event Service;
C based on CORBA Notification Service.

Implementation	t_{min} (ms)	t_{avg} (ms)	*jitter* (ms)
A	0.12	0.38	0.58
B	0.33	0.62	0.54
C	0.40	0.72	0.58

TABLE II

LATENCY WITH A 64 BYTE PACKET WHEN EVENT CHANNEL, SUPPLIER AND CONSUMER ARE ON THE FASTER MACHINE.

Implementation	t_{min} (ms)	t_{avg} (ms)	*jitter* (ms)
A	0.12	0.38	0.58
B	0.67	1.44	0.78
C	0.79	1.49	0.74

TABLE III

LATENCY WITH A 64 BYTE PACKET WHEN EVENT CHANNEL IS ON THE SLOWER MACHINE AND SUPPLIER AND CONSUMER ARE ON THE FASTER MACHINE.

Implementation	t_{min} (ms)	t_{avg} (ms)	*jitter* (ms)
A	0.34	0.52	0.57
B	0.68	1.28	0.98
C	0.76	1.35	0.79

TABLE IV

LATENCY WITH A 64 BYTE PACKET WHEN EVENT CHANNEL IS ON THE FASTER MACHINE, AND SUPPLIER AND CONSUMER ARE ON THE SLOWER MACHINE.

In the first set of experiments a 64 Byte packet is pushed by the Supplier to a single Consumer. Consumer activity is limited to the update of the latency value so far. Tables II, III, and IV report latency's minimum, average, and standard deviation (jitter) values on a set of 50,000 samples considering alternative allocations of Event Channel, Supplier, and Consumer. Allocation of the Event Channel only affects implementations B and C.

Of course, due to the single Consumer type of experiment, the Distributed Callback approach exhibits a lower latency than Event and Notification Services implementations (whose additional features are not utilized). The added latency of Event and Notification Services is small when the Event Channel is located in the same host as the Supplier and Consumer. When the Channel is located on another machine the performance of CORBA services decreases (Tables III, IV), since data are sent forth and back to the Channel host through the network. In the latter arrangement, which is recommended for scalability to multiple Consumers, latency decreases when the Event Channel is on the faster machine even if Supplier and Consumer are located on a slower machine.

Similar results are obtained by considering a more realistic experiment where a 76817 Byte frame (a 320×240 bitmap with 256 grey levels, plus the PGM header) acquired from a video camera is pushed by the Supplier to the Consumer, that in turn displays the received frame on a screen. The latency of the three implementations is reported in Table V. Comparing Tables V and II, performance of Channel solutions and Distributed Callbacks is even closer on larger data size. Moreover, a three order of magnitude message size increase (from 64 to 76817 Bytes) determines a one order of magnitude increase in latency. With any data distribution technique, delivery of information is more efficient with larger chunks of data.

Implementation	t_{min} (ms)	t_{avg} (ms)	*jitter* (ms)
A	1.97	2.8	1.5
B	4.3	5.3	1.6
C	3.9	4.9	1.9

TABLE V

LATENCY WITH A 76817 BYTE FRAME WHEN EVENT CHANNEL, SUPPLIER AND CONSUMER ARE ON THE FASTER MACHINE.

Implementation	t_{min} (ms)	t_{avg} (ms)	*jitter* (ms)
A	1.97	2.8	1.5
B	22	25	4.5
C	20.8	23	2.5

TABLE VI

LATENCY WITH A 76817 BYTE FRAME WHEN SUPPLIER AND CONSUMER ARE ON THE FASTER MACHINE, AND EVENT CHANNEL IS ON THE SLOWER MACHINE.

Table VI confirms that placing the Event Channel on a separate but slow host has a detrimental effect on data distribution performance.

The next set of experiments measures the average time interval between two successive 64 Byte packet receptions

(interarrival time) increasing the number of Consumers from 1 to 100. Results (in ms) for selected Consumer configurations are reported in Tables VII and VIII.

Implem.	N=1	N=3	N=10	N=50	N=70	N=100
A	0.73	0.98	3.76	37.76	45.53	82.77
B	1.25	1.48	4.52	38.00	41.94	71.46
C	1.38	1.67	5.15	40.60	45.18	76.34

TABLE VII

AVERAGE INTERARRIVAL TIME (IN MSECS) WITH A 64 BYTE PACKET, INCREASING THE NUMBER OF CONSUMERS (N). SUPPLIER, CONSUMERS AND EVENT CHANNEL ARE ON THE FASTER MACHINE.

Implem.	N=1	N=3	N=10	N=50
A	0.73	0.98	3.76	37.76
B	1.99	3.58	9.88	59.40
C	2.12	3.83	10.76	65.80

TABLE VIII

AVERAGE INTERARRIVAL TIME (IN MSECS) WITH A 64 BYTE PACKET, INCREASING THE NUMBER OF CONSUMERS (N). SUPPLIER AND CONSUMERS ARE ON THE FASTER MACHINE, AND EVENT CHANNEL IS ON THE SLOWER MACHINE.

At the beginning of the range investigated, 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, and they show better performance than the Callback implementation when the number of Consumers increases.

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 implementions. Exploitation of CORBA services for data distribution preserves portability of the application.

V. CONCLUSIONS

Distributed telerobotic applications require effective techniques for timely delivery of sensory data to remote clients. In this paper, we have described and evaluated three implementations of a sensor data distribution subsystem in the context of a CORBA-based framework for telerobotic applications.

Experimental results show that solutions exploiting CORBA Services and based on the Event Channel paradigm represent a viable alternative to ad-hoc solutions. The overhead associated to CORBA Services becomes less significant with larger message size, and, moreover, these services achieve better robustness, extensibility, reduction in programming complexity, and scalability when the number of clients increases.

We now plan to investigate additional techniques for distributing data with minimal overhead. Relevant ongoing research includes implementation of the publisher/subscriber model in the real-time domain [17]–[19] and the OMG initiative toward a new CORBA Service for data distribution in real-time systems [20].

ACKNOWLEDGMENT

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] K. Goldberg and R. Siegwart, Eds., *Beyond Webcams: an Introduction to Online Robots*. The MIT Press, 2001.
[2] *WS2001: International Workshop on Tele-Education in Mechatronics Based on Virtual Laboratories*, Weingarten, Germany, July 2001.
[3] P. Backes, K. Tso, and G. Tharp, "Mars Pathfinder Mission Internet-Based Operations using WITS," in *IEEE Int. Conf. Robotics and Automation*, May 1998, pp. 284–291.
[4] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory control*. Cambridge, MA: MIT Press, 1992.
[5] R. J. Anderson and M. W. Spong, "Bilateral Control of Teleoperators with Time Delay," *IEEE Trans. on Automatic Control*, vol. 34, no. 5, 1989.
[6] B. Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, 1999.
[7] S. Bottazzi, S. Caselli, M. Reggiani, and M. Amoretti, "A Software Framework based on Real-Time CORBA for Telerobotic Systems," in *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, 2002.
[8] C. Sayers, *Remote Control Robotics*. Springer Verlag, 1999.
[9] 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, pp. 1167–1172.
[10] M. Amoretti, S. Bottazzi, M. Reggiani, and S. Caselli, "Experience in teleoperation system design based on real-time CORBA," in *Int'l Conf. on Advanced Robotics*, 2003.
[11] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.
[12] *The Common Object Request Broker: Architecture and Specification Revision 2.4*, Object Management Group, Oct. 2000.
[13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *A System of Patterns*. Wiley and Sons, 1996.
[14] Object Management Group, "Event service specification, v. 1.1," http://www.omg.org/technology/documents/formal/event_service.htm, Mar. 2001.
[15] ——, "Notification service specification, v. 1.0.1," http://www.omg.org/technology/documents/formal/notification_service.htm, Aug. 2002.
[16] Distributed Object Computing (DOC) Group, "Real-time CORBA with TAO (The ACE ORB)," http://www.ece.uci.edu/ schmidt/TAO.html.
[17] R. Rajkumar, M. Gagliardi, and L. Sha, "The real-time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation," in *IEEE Real-time Technology and Applications Symposium*, 1995.
[18] T. Harrison, D. Levine, and D. Schmidt, "The design and performance of a real-time CORBA Event Service," in *ACM SIGPLAN Conferences on Object-Oriented Programming, Systems, Languages, and Applications*.
[19] M. Mock and E. Nett, "Real-time communication in autonomous robot systems," in *Int. Conf. on Autonomous Decentralized Systems*, 1999.
[20] Object Management Group, "Data distribution service for real-time systems, request for proposal," Apr. 2002.