4

Designing Distributed, Component-Based Systems for Industrial Robotic Applications

Michele Amoretti[1], Stefano Caselli[1], Monica Reggiani[2]
[1]*University of Parma*
Italy
[2]*University of Verona*
Italy

1. Introduction

The design of industrial robotic applications is dramatically changing as the result of rapid developments in underlying technologies and novel user requirements. Communication and computing technologies provide new standards and higher levels of performance. Meanwhile, novel and more capable robotic platforms and sensors have become available. In industrial automation, this technology push has opened up new application areas, while raising the expectations casted upon robotic systems in terms of overall performance and functionality.

In next generation robotic applications, robots will turn from traditional preprogrammed, stationary systems into machines capable of modifying their behaviour based on the interaction with the environment and other robots. Changes are also induced by user requirements demanding higher efficiency, functionality and performance in the control of industrial plants. In a modern manufacturing scenario, interaction with customers and suppliers must increasingly occur remotely and on-line, *e.g.* for just-in-time manufacturing, remote maintenance, and quality monitoring. For a number of reasons, transparent and widespread access to run-time production data and internal control variables of robotic systems must be provided. This requirement highlights the distinct transition beyond the conventional idea of isolated robotic workcell. As an added element of complexity, integration between different technologies in the shop floor and in the design and production management departments is required to achieve the full potential of computer-integrated manufacturing.

The growth in complexity in industrial robotic applications demands higher levels of abstraction. A first step toward the improvement of software quality and cost reduction has been the increasing adoption of object-oriented reuse techniques, based on components and frameworks. The next step is to gain a comparable acceptance of reusability and transparency concepts as far as the distributed nature of industrial automation systems is concerned. Indeed, industrial automation applications fall naturally into distributed architectures, since they require the ability to connect multiple robotic and sensory devices, which must be controlled in a coordinated manner. As discussed in this chapter, communication middlewares are the key technology to achieve these goals.

Source: Industrial Robotics: Programming, Simulation and Applicationl, ISBN 3-86611-286-6, pp. 702, ARS/plV, Germany, December 2006, Edited by: Low Kin Huat

The exploitation of standard components and distributed systems technology as key interconnection infrastructure can help to alleviate many of the complexity issues associated with the requirements of flexibility, fast implementation, maintainability, and portability. However, requirements arising in industrial automation provide additional challenges to application developers, beyond those faced when middleware technology is deployed in standard Internet applications. Control of manufacturing lines and data acquisition systems requires reliable and high-performance architectures operating in real-time. Resorting to custom protocols to satisfy such requirements in the context of complex systems is no longer viable. The need to reconfigure systems quickly and frequently for competitiveness 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. Another challenge confronting developers is the very size and complexity of industrial systems that often prevent a full understanding of the overall system and libraries, making hard for developers to predict the side-effects of future changes.

The purpose of this chapter is to describe how mainstream and advanced features of an object-oriented middleware can be exploited to meet the requirements of novel industrial robotic applications. We identify CORBA as the middleware best suited for such applications. More specifically, we review a number of CORBA services and show how Real-Time CORBA extensions and asynchronous method invocation meet performance and functional requirements. CORBA services for concurrency control and large-scale data distribution provide an effective infrastructure to meet common requirements arising in the control of distributed devices. Limitations in currently available implementations of the CORBA standards (*e.g.* for fault-tolerance and security) are also discussed.

The remaining of the chapter is organized as follows. Section 2 describes the fundamental requirements to be faced in the development of distributed industrial applications. Section 3 provides a brief overview of the two most prominent actors of the distributed middleware scenario, Web Services and CORBA. Section 4 illustrates and evaluates the most advanced CORBA features and other CORBA services relevant to distributed industrial robotic applications. Section 5 presents a sample CORBA-based application including a robotic manipulator platform and several sensor devices. Finally, Section 6 concludes the chapter, discussing its main results.

2. Requirements of Distributed Industrial Robotic Applications

Developing a suitable system architecture meeting requirements for distributed industrial robotic applications is 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 distributed robotic applications, we identify the following set of requirements for the system architecture.

The system must ensure **interoperability** and **location transparency**. In modern industrial plants many embedded processors and servers are networked together requiring their software to handle communications and interoperability issues while ensuring reliability and performance. To achieve portability, the active software components of the architecture (or *peers*) should be light-weight 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 industrial systems need to be reconfigurable, able to produce different parts, depending on which machines are used and in which order. Control software and physical resources, like sensor and robot controllers, need to be easily connected/disconnected to the systems. Hence, **support for multiple clients** is required to system devices, along with suitable **authentication and concurrency protocols** to ensure safe access according to prescribed policies and priorities. Control actions, in fact, may require or be entrusted with different access levels: from simple sensor monitoring, to robot control, to system supervision.
Servers should ensure **real-time operation** to allow implementation of the appropriate control laws with guaranteed operation. Moreover, they should accept and manage requests preserving their ordering, and exhibit differentiated reactions depending on their urgency. An industrial robotic system should also provide 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 devices and sensors in a networked environment requires **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, 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 controlling networked devices. This feature is not always required for simple applications consisting of a limited number of distributed requests, possibly operated in stop-and-go mode. However, it becomes necessary for advanced scenarios where multiple concurrent activities can be invoked at the server. Examples of such tasks are coordinated operation of multiple arms or concurrent sensing and manipulation.
In industrial applications, the timely availability of adequate sensory data is required for plant control, production monitoring, remote maintenance, data logging, and post-production analysis. **Efficient and scalable data distribution** techniques must be available to return sensory information to a dynamic and possibly large set of data consumers.
Finally, while early approaches to distributed robotic systems were based on development of in-house solutions, new industrial applications take advantage of the availability of standard infrastructures and technologies. In this way, the total cost of ownership and management of the architecture controlling the robotic system can be reduced thanks to the adoption of **component-off-the-shelf (COTS)** and **component-based design**. Approaches promoting **software reuse**, such as developing a common framework and exploiting COTS, have the potential of drastically reducing maintenance and integration costs.

3. Distributed Middlewares

A distributed middleware is a connectivity software designed for the interaction of software components in complex, distributed applications. During the 90's, this technology has greatly evolved to provide interoperability and application portability in client/server architectures. A distributed middleware, however, has emerged as an enabling technology

also for distributed industrial robotic applications, requiring the ability to interconnect multiple and possible heterogeneous platforms while ensuring location transparency and extensibility. Among the most prominent actors in the distributed middleware arena are Web Services and CORBA technologies.

3.1 Web Services

Service-oriented computing [Bichier et al., 2006] defines an architectural style whose goal is to achieve loose coupling among interacting software entities. In this context, Web Services [W3C, 2004][Motahari Nezad et al., 2006] provide a standard, simple and lightweight mechanisms for exchanging structured and typed information between services in a decentralized and distributed environment. The main goal of Web Services is to allow machine-to-machine interaction, whereas traditional Web applications are human-to-human oriented. Relevant specifications and standards for Web Services include *eXtensible Markup Language* (XML), *Simple Object Access Protocol* (SOAP), the communication protocol, *Web Service Description Language* (WSDL), a machine-processable format to describe service interfaces, and *Universal Description, Discovery and Integration* (UDDI), centralized or distributed repositories listing service interfaces. We refer to the cited literature for the full development of these concepts.

A Web Service contract describes provided functionalities in term of messages. By focusing solely on messages, the Web Service model is completely language, platform, and object model-agnostic. As long as the contract that explains service capabilities and message sequences and protocols it expects is honoured, the implementations of Web Services and service consumers can vary independently without affecting the application at the other end of the conversation.

The use of Web Services is rapidly expanding driven by the need for application-to-application communication and interoperability. The question thus is whether Web Services are the right middleware for robotics as well. In our view, while the generality and reuse capabilities of Web Services are appealing for robotic applications as well, they fail to provide the performance guarantees for end-to-end system operation required in robotic applications.

3.2 CORBA and Other DOC Middlewares

Several Distributed Object Computing (DOC) middleware standards are available, notably the Common Object Request Broker Architecture (CORBA) [OMG A, 2004] promoted by the Object Management Group (OMG), and Microsoft's Distributed Component Object Model (DCOM). The distributed objects concept is an important paradigm, because it is relatively easy to hide distribution aspects behind an object's interface. Furthermore, since an object can be virtually anything, it is also a powerful paradigm for building systems.

DCOM's major design goal appears to be providing improved functionality while staying compatible with previous versions incorporated in the early Windows systems. Unfortunately, DCOM has proven exceedingly complex, especially considering its proprietary nature. Today, DCOM has been largely superseded by its successor .NET, which however remains a proprietary solution mainly restricted to Microsoft operating systems.

When language, vendor, and operating system independence is a goal, CORBA is a mature solution that provides similar mechanisms for transparently accessing remote distributed objects while overcoming the interoperability problems of .NET. CORBA is the result of an effort to provide a standard middleware platform to allow applications from many different software manufacturers to interoperate. Implementing a distributed architecture with

CORBA allows smooth integration of heterogeneous software components. To ensure portability, reusability, and interoperability, 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 1).
To achieve language independence, CORBA requires 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.

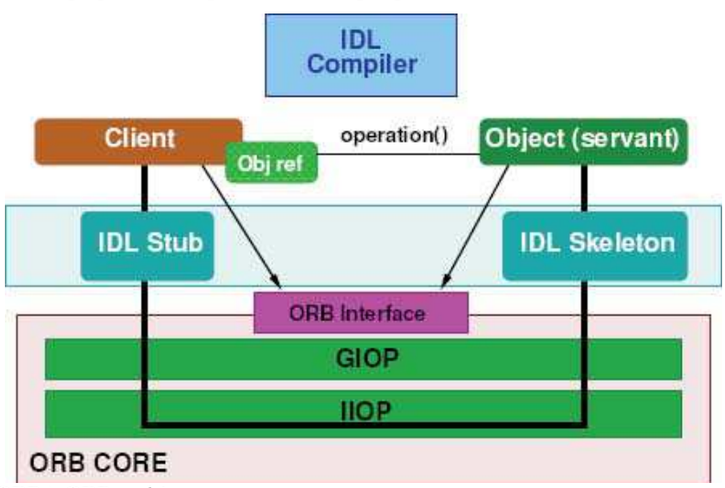


Fig. 1. CORBA ORB architecture.

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 cited literature [OMG A, 2004] for additional information on general features of the CORBA Standard. The most advanced CORBA implementation, in terms of compliancy, is The ACE ORB (TAO) [Schmidt et al., 1998]. TAO implements all the QoS-oriented parts of the specification. For example, the Real-time CORBA 1.0 [OMG C, 2002] standard supports statically scheduled Distributed Real-time and Embedded (DRE) systems (such as avionics mission computers and industrial process controllers) in which task eligibility can be mapped to a fixed set of priorities. Starting from version 1.5, TAO supports the Real-time CORBA 1.2 specification [OMG D, 2003] which addresses limitations with the fixed-priority mechanisms by introducing two new concepts in Real-time CORBA: *distributable threads,* that are used to map end-to-end QoS requirements to distributed computations across the endsystems they traverse, and a *scheduling service architecture,* that allows applications to enforce task eligibility.

3.3 Choosing CORBA in Industrial Robotic Applications

The decoupling and interoperability features offered by the message-oriented approach characterizing Web Services are of course interesting for the industrial robotics scenario, where mobile robots and server applications often run on different platforms. However, XML processing is too "heavy" to meet the real-time constraints arising in industrial robotic applications. Several researchers have experimented with performance improvements to Web Services, such as faster parsers [Bayardo et al., 2004], data compression [Liefke, 2000], protocol optimizations [Chiu et al, 2002], or binary encodings [Tian et al., 2004], but the outcome generally depends on the structure of the data used and the application itself.

Today, CORBA is in wide use as a well-proven architecture for building and deploying significant enterprise-scale heterogeneous systems, as well as DRE systems. In these sectors CORBA is growing because it can harness the increasing number of operating systems, networks, and protocols supporting real-time scheduling into an integrated solution that provides end-to-end QoS guarantees to distributed object applications. Research over the past several years has identified key architectural patterns and performance optimizations required to build high-performance, real-time ORBs. This effort resulted in the development of prototype real-time ORB endsystems that can deliver end-to-end QoS guarantees to applications [Gill et al., 2004], [Schantz et al., 2006], [Zhang et al., 2005], [DiPippo et al., 2003].

CORBA's most advanced and recent features (Real-Time CORBA, AMI) provide functionalities almost essential in application domains sharing critical concerns such as industrial automation, avionics, and telecommunications. Based on the features discussed in the next section, CORBA seems the logical choice for building complex distributed robotic applications, thereby satisfying the interoperability, transparency, COTS availability, component-based design, and software reusability requirements previously highlighted.

4. CORBA Features for Distributed Robotic Applications

This section discusses CORBA features introduced with version 2.4 of the Standard, along with other CORBA Services relevant to distributed robotic systems. Real-Time CORBA 1.0, Messaging and several consolidated services are evaluated by means of simple benchmarks designed to assess their suitability for distributed robotic applications. Real-Time CORBA 1.2 is not considered here because of its draft status.

4.1 Concurrency Among Clients

A requirement of advanced distributed robotic systems is to manage input from all controlling devices while generating a single and coherent control sequence for each robot server, allowing collaborative and coordinated operation [Chong et al., 2000].

As a basic functionality, server applications 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** [OMG, 2000]. This service allows several clients to coordinate their concurrent accesses to a shared resource so that the resource consistent state is not compromised. Of course, it is up to the developer to define resources and identify situations where concurrent accesses to resources conflict.

For these purposes, the Concurrency Service provides the *lock* as the basic coordination mechanism. 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 are defined, allowing different conflict resolution policies among concurrent clients. We show how the CORBA Concurrency Service can be exploited by means of an example. This pattern can be adapted to several situations involving conflicting reading and writing access rights to one or more devices across the distributed system.

In a scenario where several clients compete to control a single robot and/or access data from multiple sensors, exclusive control of the robot must be granted only to one client in a given interval of time, while simultaneous read of sensor data should be allowed to other consumers as well. 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 lock object is created and registered in the Naming Service.

As shown in Figure 2 (scenario 1), the client invoking commands on the Robot object holds an exclusive lock ensuring exclusive control. Indeed, as the exclusive 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.

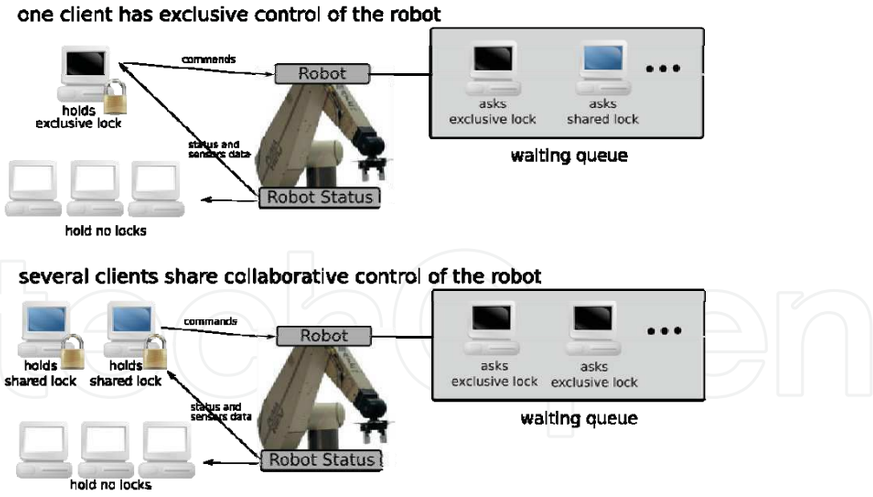


Fig. 2. Two scenarios of concurrent access to a single robot device from several clients.

To preserve generality and cover a wider domain of applications [Chong et al, 2000], an alternative scenario can be outlined, where a group of clients wants to control a single robot

in a collaborative way (*e.g.*, in a telerobotic application, a ''primary'' operator with some ''backup'' operators), while preventing further clients from obtaining exclusive control of the robot. In this scenario (Figure 2, scenario 2), a collaborating client holds a shared lock. Since the shared lock conflicts only with exclusive locks, it allows sharing of robot control with other clients holding shared locks, whereas clients requesting exclusive control through an exclusive lock are suspended in the exclusive access queue.
An experimental evaluation of the access control mechanisms outlined above for multiple concurrent clients is reported in Section 5.

4.2 Server-side Concurrency 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 (low-priority tasks blocking high-priority ones).
With the Real-Time CORBA specification, 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 pre-allocation 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 maximum number of dynamic threads is available. These threads 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 an industrial robotic 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 distributed robotic applications should also provide clients with the guarantee of correct execution priorities of application tasks. However, heterogeneity of nodes, in distributed applications, precludes the use of a common priority scale. In early CORBA versions and other middleware approaches, this problem forced users to concern about low-level details of threads on different OSes. Real-Time CORBA has solved the problem by providing a **priority mapping** mechanism converting CORBA priority levels, assigned to CORBA operations, to OS native priority levels (and vice versa).
Critical distributed robotic 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 honoured 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. 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 [Arulanthu et al., 2000].
The **Asynchronous Method Invocation (AMI)** model [OMG A, 2004] provides a more efficient way to perform non-blocking 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, whereas 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 behaviour is not satisfactory for many robotic applications, where a set of parallel actions must often begin at the ''same'' time and coordination of their execution is required to ensure logical correctness or safety. This problem can be solved by implementing a *waiting rendezvous* strategy [Douglass, 1999] based on the available CORBA mechanisms. We have developed an implementation where 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.

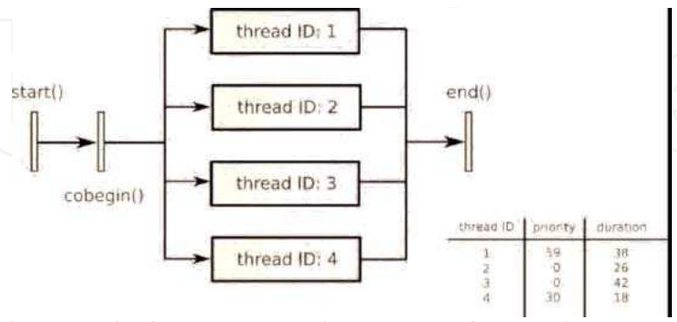


Fig. 3. Precedence graph of a concurrent task, consisting of an initial action start(), followed by four concurrent actions with different priority.

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. We have also experimented the effectiveness of cobegin(n) in avoiding priority inversion in the execution of parallel actions. One of the experiments is described in Figure 3, showing 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 4 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 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.

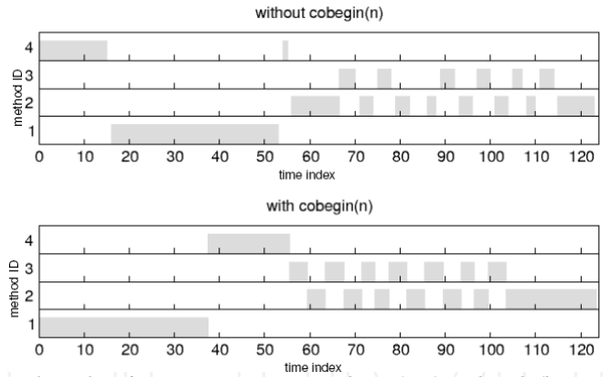


Fig.. 4. Experimental results of concurrent actions without (top) and with (bottom) cobegin(n).

Solutions currently available in CORBA for transmission of time-critical streaming data are still quite limited. A standard for Audio/Video Streaming [OMG B, 2000] 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 has recently defined a new specification to facilitate the exchange of continuous data in the CORBA Component Model [OMG, 2005].

A data distribution technique alternative to Audio/Video Streaming, less efficient but more portable, adopts a Publisher/Subscriber communication model [Buschmann et al., 1996]. 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, namely 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. The CORBA Event Service [OMG B, 2004] 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 5). 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 distributed robotic applications, the most appropriate 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 highlights several issues. Firstly, to avoid compile-time knowledge of the actual type of the ''event,'' sensor data must be communicated as an OMG IDL "any type" data type. The communication is therefore type-unsafe and consumers are charged with the duty of converting between the "any type" and 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 set 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.

Fig. 5. CORBA Event Service architecture.

Notification Service. A second solution investigated for the data distribution subsystem is based on the CORBA Notification Service [OMG C, 2004], which has been 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 5) 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 robotic application requiring to distribute sensory data to a number of clients. The experiments reported in the following 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. Two hosts have been used, one for the supplier, and one for the consumer(s). The experiments have been performed with the Event Channel located either on the machine which hosts the supplier (low workload), or on the other machine (high workload). The main features of the exploited machines are listed in Table I. The hosts are connected via Fast Ethernet switches, with different bitrates. Unless stated otherwise, the network had no significant traffic, nor was any other processing taking place on these hosts.

Host name	Hardware configuration	Operating system	Network connection
Trovatore	Intel P4 2.4 GHz, 512 MB RAM	Gentoo 1.6.14	1Gbps
Malcom	AMD Athlon 64 3500+, 1 GB RAM	Gentoo 1.6.13	1Gbps
Tromba	Intel P4 2.0GHz, 512 MB RAM	Kubuntu 5.10	100 Mbps

Table I. Features of the machines hosting the data distribution tests.

In the first set of experiments, the supplier is hosted by Trovatore and pushes 20000 packets (64 Bytes per packet). Minimum, average, and standard deviation (jitter) values of interarrival times are evaluated. Consumer activity is limited to the update of the latency value so far. We measured the average time interval between two successive 64 Byte packet receptions (interarrival time) increasing the number of Consumers from 1 to 150 (hosted by Malcom). Event and Notification Services results are compared with those of a Distributed Callback implementation based on the Observer patter [Gamma et al., 1995], *i.e.*, a pattern where new sensor data, when ready, are sent by the supplier application to all consumers previously registered as Observers (see Figure 6).

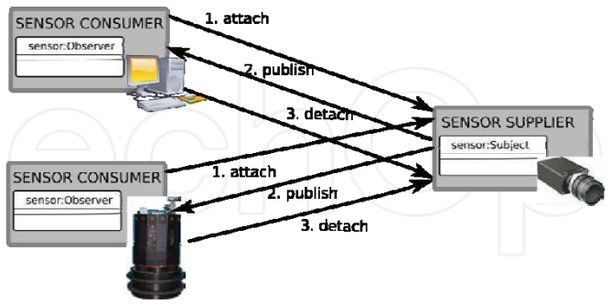


Fig. 6. The Distributed Callback mechanism.

Figure 7 shows the results for five solutions: Distributed Callback, Event Service with channel on Malcom with the consumers (A) , Event Service with channel on Trovatore with the supplier (B), Notification Service with channel on Malcom (A), Notification service with channel on Trovatore (B). With less than 10 consumers, the five solutions are equivalent. With more than 10 clients, ES(A) and NS(A) perform better than Distributed Callback, ES(B), and NS(B). That is, the Event and Notification Services show better performance if they are located on the same machine which hosts the consumers, because this setting reduces network congestion since the only remote connection is the one between the supplier and the event channel, while the connections between the channel and the consumers are local to the machine which hosts them. Focusing on this solution, for more than 100 consumers the Notification Service seems to be more efficient than the Event Service.

The second set of experiments investigates synchronization among consumers on the reception of data packets. Figure 8 shows the average interarrival time for two consumers when they are on the same machine (Malcom, Test 1), and when one of them is on a machine slower than the one hosting the other consumer (Tromba and Malcom, Test 2). In TAO default configuration, the Event Service pairs together the two consumers, forcing the packet reception rate to the one permitted by the slower one (compare Test 1 ES with Test 2 ES). The same result is obtained with the Notification Service in the default single-threaded configuration (compare Test 1 NS with Test 2 NS). Not surprisingly, if the Notification Service is configured to deploy a thread at each proxy supplier, the overall performance is improved, and Test 2 gives better results than Test 1 (label MT stands for MultiThreaded).

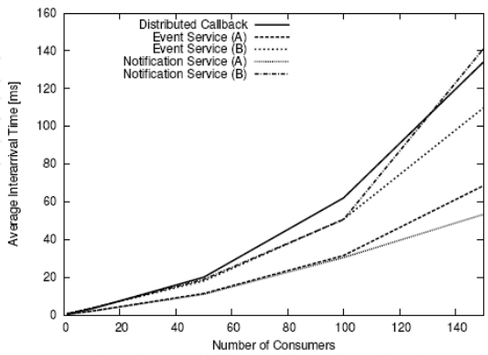


Fig. 7. Interarrival times resulting from different data distribution architectures.

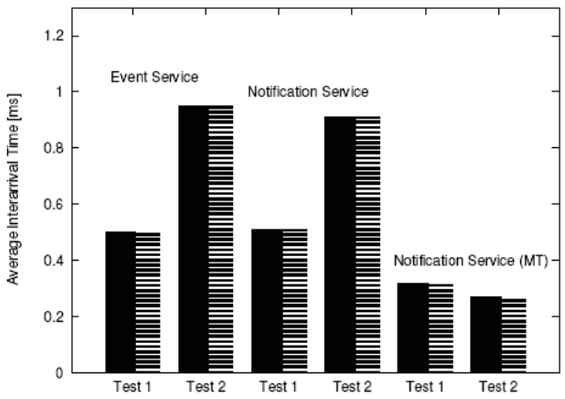


Fig. 8. Interarrival times resulting by three different Publisher/Subscriber models. Test 1 involves two consumers with similar characteristics; in Test 2 one of the consumers (dashed bar) runs on a slower machine.

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 (offered by the Event and Notification Services) represents an effective tool to maintain the overall workload under control. When consumers 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 Distributed Callback and Event Service implementations.

5. A CORBA-based Telerobotic System

In the previous section, CORBA services relevant to distributed robotic applications have been introduced and evaluated using "microbenchmarks", *i.e.* test suites conceived for the specific service under investigation. The aim of this section is to describe and assess a complete telerobotic application which encompasses many features typical of distributed industrial robotic applications. The telerobotic application has been developed exploiting a framework which integrates various CORBA services.

5.1 The Application

The telerobotic system is illustrated in Figure 9. The remote site includes an Unimation PUMA 560, a six d.o.f. arm whose controller is interfaced to a Pentium-based workstation running the RCCL/RCI 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 system features additional sensoriality, including 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.

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 its environment, adding new sensoriality and moving available hardware. The inset graph in Figure 10 shows the GUI with windows displaying sensory information received from the stereo camera.
An advanced client station also supports the control of the robot using an 18-sensor CyberTouch (a virtual reality glove with tactile feedback form Immersion Corp, Inc.) and a six degree of freedom Polhemus tracker (Figure 10).

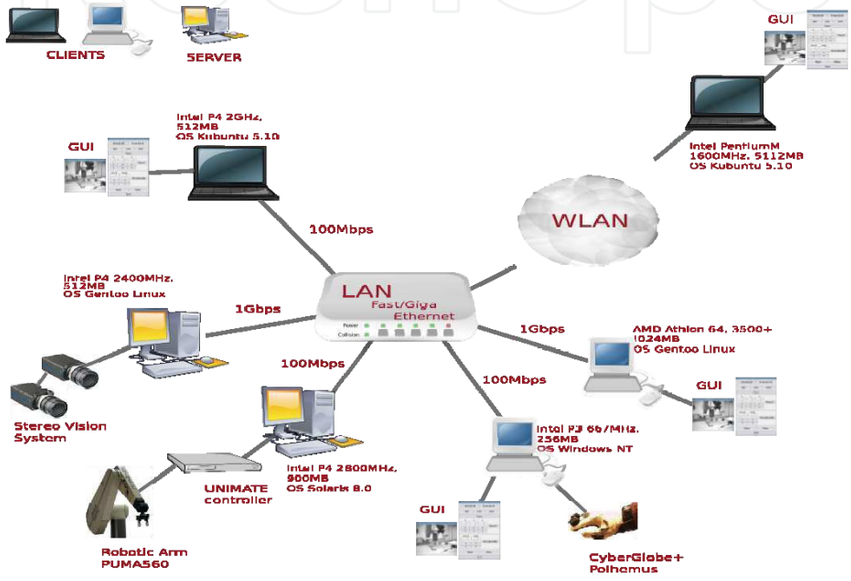


Fig. 9. The experimental testbed for a CORBA-based telerobotic application.

5.2 Performance Analysis

The experimental results in this section refer to a peg-in-hole manipulation task, and to a concurrent task requiring four clients to perform a stacking manipulation task. While performing the tasks, the system returned to connected users the visual information generated by a camera shooting the workspace.
The first test was performed both over an Ethernet LAN (with average latency of 0.1 ms), and over a WAN (with average latency of 100 ms). The performance observed in the second case was mainly influenced by the time delay and frame rate of the visual information received from the camera using the CORBA-based data distribution mechanisms integrated in the application. 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 drawback of this solution was of course the lower quality of the image due to the lossy compression.

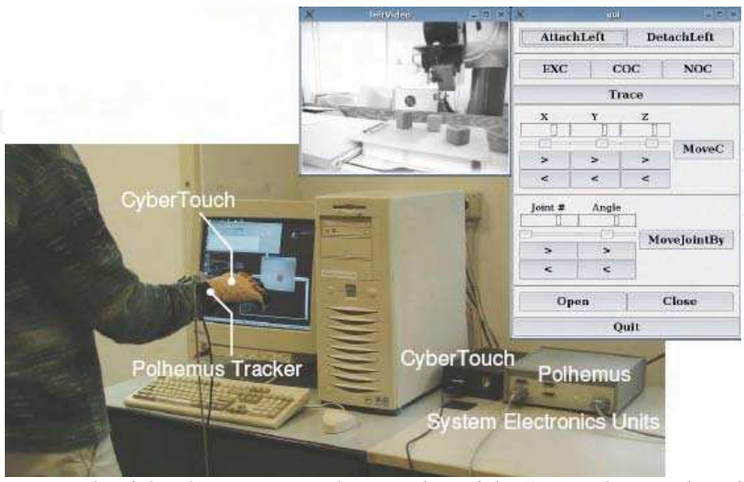


Fig. 10. An example of the client setup, and a snapshot of the GUI with a window displaying sensory information received from the remote site.

Figure 11 shows the Z trajectory of the gripper during two teleoperated peg-in-hole tasks. The solid line shows the gripper height during a task performed in the Ethernet LAN environment. The dashed line shows the same data obtained with the same operator remotely connected to the servers on a high-speed DSL link. In both cases, the GUI was used and proved very useful to ensure effective interaction with the remote system. It can be observed that the task duration was essentially the same for both experiments, even though in the case of teleoperation over LAN the operator was allowed to send more motion commands (65 against 44 of the other case) thanks to the better response time of the system, *i.e.* motion commands (uplink) and sensor data (downlink) were delivered more quickly.

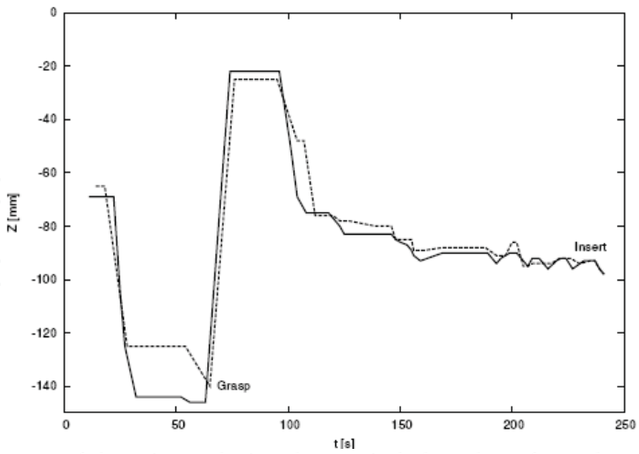


Fig. 11. Z trajectory of the gripper during the peg-in-hole task performed over an Ethernet LAN (continuous line), and over a WAN (dashed line).

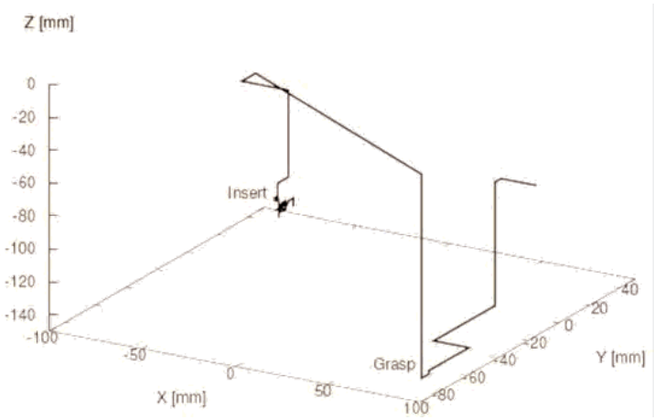


Fig. 12. Trajectory of the gripper in the peg-in-hole experiment (over an Ethernet LAN).

In the second test four clients participated to the execution of a stacking manipulation task (Figure 13). Clients were connected to the robotic server through a Fast Ethernet switch with negligible latency. The main goal of the experiment was to investigate the reliability of the mechanisms for concurrency control and access arbitration among clients described in Section 4.1.

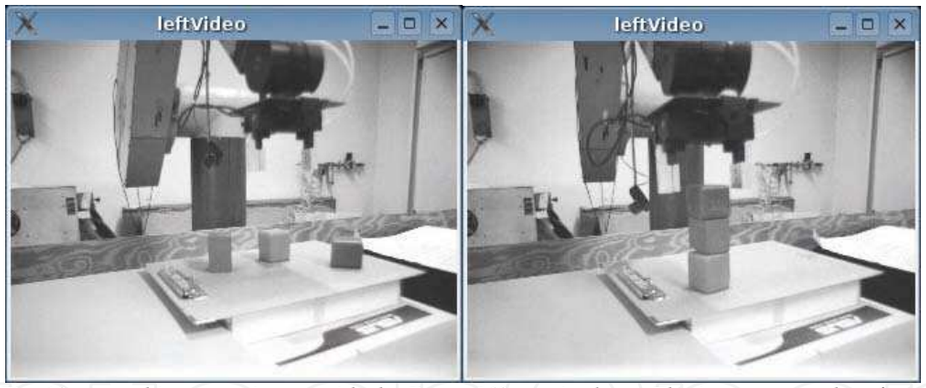


Fig. 13. A stacking experiment with the Puma 560 manipulator. The images are taken from the left eye of the stereo camera.

Figure 14 traces the main environmental values during a task segment. 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 14.
Each client can be in one of 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 bottom row of Figure 14 shows the total number of clients currently connected to the system including slot time when they are connected as ''observer.''

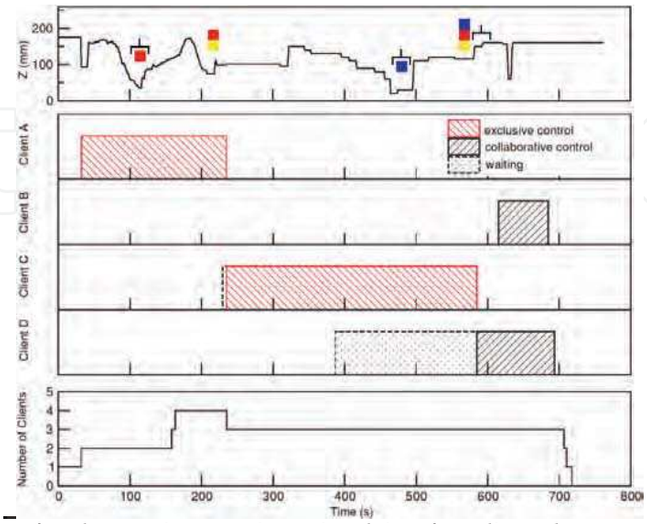


Fig. 14. Tracing of a teleoperation experiment involving four clients that concurrently access the robot arm.

Several task sessions demonstrated the reliability of the CORBA concurrency control mechanisms in providing safe arbitration to critical resources. Tasks similar to the one described were executed, and the presence of multiple clients controlling the robot never prevented their successful completion.

6. Conclusion

The viability and cost effectiveness of new distributed industrial robotic applications can be widely enhanced exploiting COTS-based software components. However, systems implementing those applications must face 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 robotic systems. In this paper, we have summarized our experience resulting from the development of distributed robotic applications based on Real-Time CORBA.

Results show that exploitation of CORBA brings a number of remarkable advantages in the distributed robotic 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. Nevertheless, the recent scaling of network and Internet infrastructures to higher levels of performance and QoS guarantees has made less relevant the overhead introduced by CORBA. 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.

7. Acknowledgements

The research described in this chapter has been supported by LARER (Laboratorio di Automazione of Regione Emilia-Romagna, Italy).

8. References

Arulanthu, A.; O'Ryan, C.; Schmidt, D. C.; Kircher, M. & Parsons, J. (2000). The Design and performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging. Proceedings of the ACM/IFIP Middleware 2000 Conference, pp. 208-230, ISBN 3-540-67352-0, New York, NY, USA, April 2000, Springer (Lecture Notes on Computer Science)

Bayardo, R.J.; Gruhl, D.; Josifovski, V. & Myllymaki, J. (2004). An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing. *Proceedings of 13th In'l Conf. World Wide Web,* pp. 345-354, ISBN 1581139128, New York, NY, USA, May 2004, ACM Press

Bichier, M. & Lin, K.-J. (2006). Service-Oriented Computing. *IEEE Computer*, Vol. 39, No. 3, pp. 99-101, ISSN 0018-9162

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P. & Stal, M. (1996). Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley and Sons, ISBN 0471958697, New York

Chiu, K.; Govindaraju, M. & Bramley, R. (2002). Investigating the Limits of SOAP Performance for Scientific Computing. *Proceedings of 11th IEEE In'l Symp. High-Performance Distributed Computing,* pp. 246-254, ISBN 0-7695-1686-6, Edimburgh, Scotland, July 2002, IEEE CS Press

Chong, N.; Kotoku, T.; Ohba, K.; Komoriya, K.; Matsuhira, N. & Tanie, K. (2000). Remote Coordinated Controls in Multiple Telerobot Cooperation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2000)*, pp. 3138-3143, ISBN 0-7803-5889-9, San Francisco, CA, USA, April 2000, IEEE Press.

DiPippo, L.; Fay-Wolfe, V.; Zhang, J.; Murphy, M. & Gupta, P. (2003). Patterns in Distributed Real-Time Scheduling, *Proceedings of the 10th Conference on Pattern Language of Programs 2003*, Monticello, IL, USA, September 2003.

Douglass, B. (1999). *Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, ISBN 0201498375, Reading, MA

Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1995). *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, ISBN 0201633612, Reading, MA

Gill, C.; Schmidt, D. C.; Krishnamurthy, Y.; Pyarali, I.; Mgeta, L.; Zhang, Y. & Torri, S. (2004). Enhancing Adaptivity Standard Dynamic Scheduling Middleware, *The Journal of the Brazilian Computer Society (JCBS) special issue on Adaptive Software Systems*, Vol. 10, No. 1, pp. 19-30, ISSN 0104-6500

Liefke, H. & Suciu, D. (2000). XMill: An Efficient Compressor for XML Data. *Proceedings of ACM SIGMOD Conference on Management of Data,* pp. 153-164, ISBN 1-58113-218-2, Dallas, TX, USA, June 2000, ACM Press

Motahari Nezad, H. R.; Benatallah, B.; Casati, F. & Toumani, F. (2006). Web Services Interoperability Specifications. *IEEE Computer*, Vol. 39, No. 5, pp. 24-32, ISSN 0018-9162

OMG A (2000). Concurrency Service Specification, Version 1.0, http://www.omg.org/technology/documents/formal/concurrency_service.htm

OMG B (2000). Audio/Video Streams, Version 1.0, http://www.omg.org/technology/documents/formal/audio.htm

OMG (2002). Real-time CORBA 1.0 Specification, http://www.omg.org/docs/formal/02-08-02.pdf

OMG (2003). Review draft of the 1.2 revision to the Real-time CORBA Specification (OMG document realtime/o3-08-01), previously designated Real-time CORBA 2.0, http://www.omg.org/docs/ptc/01-08-34.pdf

OMG A (2004). CORBA/IIOP 3.0.3 Specification, http://www.omg.org/docs/formal/04-03-01.pdf

OMG B (2004). Event Service Specification, Version 1.2, http://www.omg.org/technology/documents/formal/event_service.htm

OMG C (2004). Notification Service Specification, Version 1.1, http://www.omg.org/technology/documents/formal/notification_service.htm

OMG (2005). Streams for CCM Specification, http://www.omg.org/docs/ptc/05-07-01.pdf

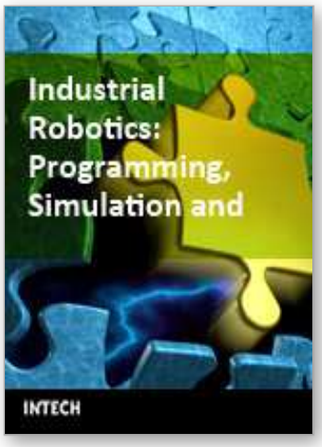
Schantz, R.E.; Schmidt, D.C.; Loyall, J. P. & Rodrigues, C. (2006). Controlling Quality-of-Service in Distributed Real-time and Embedded Systems via Adaptive Middleware. To appear in *The Wiley Software Practice and Experience Journal special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, ISSN 0038-0644

Schmidt, D. C.; Levine, D. & Mungee, S. (1998). The Design and Performance of the TAO Real-Time Object Request Broker. *Elsevier Computer Communications Journal, special issue on Building Quality of Service into Distributed Systems*, Vol. 21, No. 4, pp. 294-324, ISSN 0140-3664

Tian, M.; Voigt, T.; Naumowicz, T.; Ritter, H. & Schiller, J. (2004). Performance Considerations for Mobile Web Services. *Elsevier Computer Communications Journal,* Vol. 27, No. 11, July 2004, pp. 1097-1105, ISSN 0140-3664

W3C (2004). Web Services Architecture, *Working Group Note.*

Zhang, J.; DiPippo, L.; Fay-Wolfe, V.; Bryan, K. & Murphy, M. (2005). A Real-Time Distributed Scheduling Service For Middleware Systems. *Proceedings of the 10th International Workshop on Object-Oriented, Real-Time, Dependable Systems*, pp. 59-65, ISBN 0-7695-2347-1, Sedona, AZ, February 2005



Industrial Robotics: Programming, Simulation and Applications

Edited by Low Kin Huat

ISBN 3-86611-286-6

Hard cover, 702 pages

Publisher Pro Literatur Verlag, Germany / ARS, Austria

Published online 01, December, 2006

Published in print edition December, 2006

This book covers a wide range of topics relating to advanced industrial robotics, sensors and automation technologies. Although being highly technical and complex in nature, the papers presented in this book represent some of the latest cutting edge technologies and advancements in industrial robotics technology. This book covers topics such as networking, properties of manipulators, forward and inverse robot arm kinematics, motion path-planning, machine vision and many other practical topics too numerous to list here. The authors and editor of this book wish to inspire people, especially young ones, to get involved with robotic and mechatronic engineering technology and to develop new and exciting practical applications, perhaps using the ideas and concepts presented herein.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen