Support of multimedia services for distributed network training applications in CORBA-3

Fausto Rabitti
CNUCE-CNR, Via S. Maria, 36, Pisa, Italy

Abstract

In this paper, fundamental technological issues in the implementation of today and tomorrow distributed network training systems are outlined. Given their highly distributed nature, the problems inherent to interoperability among heterogeneous systems are discussed. In this respect, the role of CORBA, an emerging standard for application integration technology, is presented. The limitations of today CORBA definition are discussed, and the extension of CORBA-3, with respect to the management of multimedia data (fundamental in the development of the next-generation distributed network training systems), is examined in more detail.

1. Introduction

In order to provide a network-based training tool for geographically distributed user groups with different learning backgrounds, the following features are required [1]:

- self-instruction;
- interactive training (teacher-student and student-student);
- presentation through the network of various didactic material (texts, multimedia documents, audio/video material, etc.);
- creation and use of distributed databases for training course documents, books and other didactic material.

The range of users that can utilize networked training is broad: students and company workers, adult learning, middle school students, disabled, non-native language speakers, changing work-patterns, personnel training. There are two main implementation choices for this type of applications:

1. use of Internet only, which is wide-spread and low-cost;
2. use of a specialized network which allows a greater service quality, but at a higher cost.

While most today tools for NBT (Network Based Training) adopt the first choice, it seems likely that future system will enhance their functions adopting multimedia paradigms, therefore shifting to the second choice.

2. CORBA solution

Given the highly distributed nature of Network Based Training systems, the problems inherent to interoperability among heterogeneous systems are essential. In this respect, the role of CORBA, an emerging standard for application integration technology, can play an important role.

CORBA is an open standard for distributed object computing. It defines a set of components that allow client applications to invoke operations on remote object implementations. CORBA enhances application flexibility and portability by automating many common development tasks such as object registration, location, and activation; demultiplexing; framing and error_handling; parameter marshalling and demarshalling; and operation dispatching.

The objective of CORBA is to achieve portability and interoperability via object-orientation:

- Design portability is the ability to create applications which only rely on the knowledge of objects interfaces.
- Interoperability is the ability to invoke operations on objects regardless of where they are located, which platform they execute on, or what programming language they are implemented in.

CORBA is the specification of the functionality of the Object Request Broker (ORB), that is the key component intended to support location transparency, i.e., the ability to access and invoke operations on a CORBA object without needing to know where the object resides. The basic idea is that it should be equally easy to invoke an operation on an object residing on a remote machine as it is to invoke a method on an object in the same address space.

An object implementation is the part of a CORBA object that is provided by an application developer. It usually has an internal state and causes side effects on things that are not objects, such as databases, displays, network elements. An object reference is a handle to an object. An object reference will always denote a single object, but several distinct object references may denote the same object. Object references can be passed to clients either as values of their interface type, or as strings which can be turned into live object references. They contain enough information for the ORB to locate the correct implementation, but this information is inaccessible to their users. Unless an object has been destroyed or the underlying system is malfunctioning, the ORB should be able to convey an operation invocation to its target and return results. An object interface is a description of the operations that are offered by an object and can also contain structured type definitions used as parameters to those operations. Interfaces are specified in IDL (Interface Definition Language) and are related in an inheritance hierarchy.

The ORB structure (see Fig. 1) is composed of:

- IDL Stubs: pieces of codes needed to marshal client requests, generated by the IDL compiler and linked to the client
- IDL Skeleton: pieces of codes needed to unmarshal client requests, generated by the IDL compiler and linked to the object implementation
- Both these interface with the ORB run-time system to do their job for static invocations, i.e. the IDL is defined at compile time and only operations on known interface types can be invoked.
- Dynamic Invocation Interface: an interface that allows requests to be built dynamically for any operation by a client.
- Dynamic Invocation Skeleton: the symmetric interface for responding to arbitrary requests.
- ORB Interface: for communication from either client or server, mainly dealing with ORB initialization and object reference manipulation.
- Object Adapter: provides extra facilities for managing the interaction of object implementations with the ORB. The object implementation uses it to make itself available through an ORB, while the ORB uses it to manage the run-time environment of the object implementations.

The IDL interface definitions inform clients of an object what operations the object supports, the types of their parameters, and what return types to expect. A client programmer needs only the IDL to write client code that invokes remote object operations. The client uses the data types defined in IDL through a language mapping, which defines the programming language constructs that will be generated by the IDL compiler. The IDL compiler also generates stub code that the client links to. The stub code translates the programming language data types into a wire format for transmission as a request message to an object implementation. The implementation of the object has linked to it similar code, the skeleton, that translates the request into programming language data types.

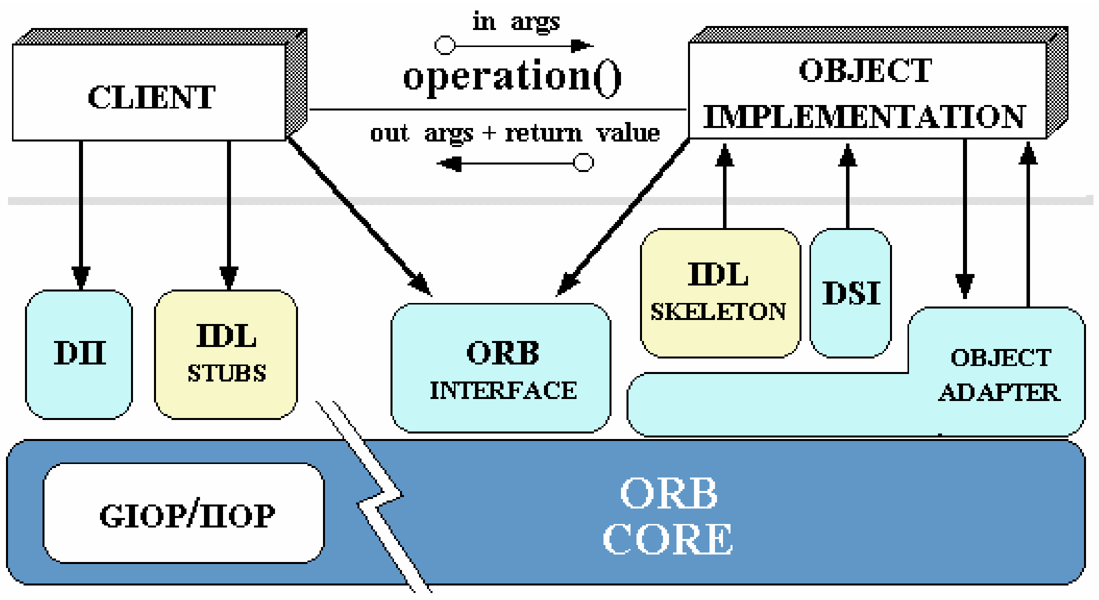

Fig. 1

The ORB acts as a message bus between objects which may be located on any machine on a network, implemented in any programming language, and executed on any hardware or operating system platform. The caller only needs an Object Reference and well-formed arguments in its language of choice to invoke an operation as if it were a local function. Different ORB's can interoperate according to the GIOP protocol. The GIOP defines a linear format for the transmission of CORBA requests and replies without requiring a particular network transport protocol. The Internet Inter-ORB protocol (IIOP) is a specialization of the GIOP which specifies the use of TCP/IP. It defines some primitives to assist in the establishment of TCP connections. Environment Specific Inter-ORB protocols (ESIOPs) allow the introduction of third-party protocols.

3. Multimedia Systems

Multimedia systems are bringing together efforts industries of computers and software, telecommunications, publishing, and consumer electronics. This convergence of the relevant technologies will allow information to be distributed on demand from very large information servers through a variety of high-capacity telecommunication media. This situation will have a major impact on the technology for network-based learning. The multimedia information servers will store very large amounts of multimedia data to support a large number of applications, such as movies (and other audio-visual data) on demand, tele-shopping, news on demand, tele-medicine etc. [4]

The main issues in the design and implementation of Multimedia Systems are:

- Large amount of data (e.g. Video)
- Unpredictable data distribution (burst data, non-Poisson arrivals)
- Delay sensitive & real-time data processing
- Flow control & buffering
- Admission control (dynamically guaranteed band-with)
- Quality of service - *QoS* (dynamic failure/recovery control)
- New cost/performance requirements

For example, supporting delay-sensitive data streams, implies the investigation of performance tradeoffs. One key question is how many disks of the array will be used for striping a multimedia object; as this number increases, the transfer time becomes smaller, which, in turn, reduces the user-observed latency . On the other hand, as the number of disks used for striping objects increases, the number of requests that the disk array can serve concurrently decreases, which in turn may lead to poor throughput.

The current state of CORBA does not support the function required by multimedia systems. However, since these perspective is considered of fundamental importance, CORBA is evolving towards this direction. This evolution is emerging in the latest evolution, i.e., CORBA3.

4. New features in CORBA3

CORBA has demonstrated to be is well suited for client/server applications running over conventional local area networks (such as Ethernet and Token Ring). However, building highly available applications with CORBA is much harder. Neither the CORBA standard nor conventional implementations of CORBA directly address complex problems related to distributed computing, such as real-time quality of service (*QoS*) or high-speed performance, group communication, partial failures, and causal ordering of events. This is also the case of multimedia systems.

There are three new main features in CORBA-3:

- Portable Object Adapter
- CORBA (asynchronous) Messaging
- Object passing by value

In the implementation of multimedia systems, the first two features are most important: the Portable Object Adapter because gives the application designer the possibility to program a suitable scheduler of multimedia object services, the CORBA Messaging because introduces the support of asynchronous data flows, typical of multimedia systems.

4.1 The Portable Object Adapter

The Basic Object Adapter (BOA) was introduced in CORBA 2.1. For the object implementer, the BOA is the interface to inform the ORB when objects come into existence and when running processes or tasks are ready to accept incoming requests on those objects. For the client, the BOA is the component of the ORB that ensures that an invocation on an object reference always reaches a running object that can respond to it. The BOA is capable of launching processes, waiting for them to initialize, and then dispatching requests to them. To do this, the BOA accesses the Implementation Repository, a component proprietary to each ORB, which stores information about:

- where the executable code that implements objects resides and
- how to run it correctly.

The semantics of the BOA specification were left intentionally vague because it was not clear which features would be required on various platforms. As a result, different vendors implemented different parts of the BOA with differences in their semantics. This implementation experience was used as the basis for the specification of the Portable Object Adapter (POA).

The POA (see Fig.2) aims at providing a comprehensive set of interfaces for managing object references and their implementations, now called servants. The code written using the POA interfaces should now be portable across ORB implementations and have the same semantics in every ORB. The POA provides standard interfaces to four kinds of tasks, very similar to the tasks that a TP monitor typically performs:

- Map an object reference to the servant that implements that object
- Allow transparent activation of objects
- Associate policy information with objects
- Make a CORBA object persistent over several server process lifetimes

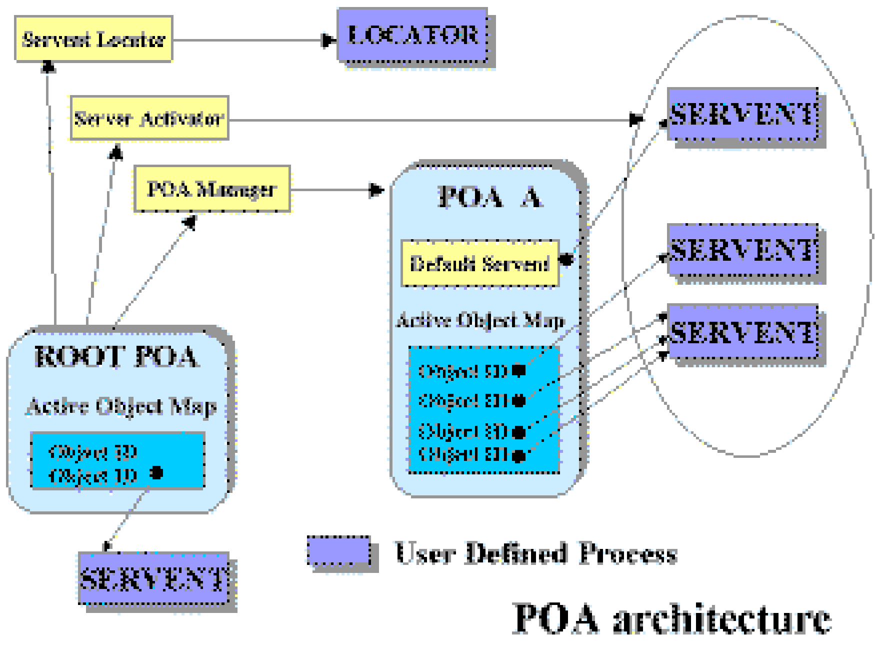

Fig. 2

The purpose of a POA is to dispatch incoming invocation requests to the correct servant object. It does so in a number of different ways (policies) which range from entirely automatic association of CORBA objects to servants, to the use of programmer-supplied servant manager objects. There can be more that one POA active in a particular server. However, there is always a root POA from which all of the other POAs are created. A reference to the root POA is always available from the ORB, and is obtained by using the *ORB::resolve_initial_references* operation. When a server is being initialized it is responsible for setting up any other descendant POAs that it requires to support its objects. POA provides operations to map CORBA objects IDs to servants, creating usable object references that can be handed to clients.

To summarize, the Portable Object Adapter in CORBA 3 provides the following new services:

- creation of CORBA objects and their references
- de-multiplexing of requests on target CORBA objects
- dispatching requests to servants (providing implementation of target CORBA objects)
- activation & deactivation of CORBA objects
- and the following new features:

- persistent and transient objects
- different policies of multi-threading
- application control over object existence
- static and dynamic skeleton interface
- DSI - Dynamic Skeleton Interface
- dynamic access to interface repository (IDL definitions)
- multiple POA

The explicit process control, possible in the POA of CORBA 3, allows the designer of multimedia applications to implement the specific implementation techniques which are necessary for handling delay-sensitive multimedia data.

42 The CORBA asynchronous messaging

The basic paradigm of CORBA communications is a blocked, synchronous model. Clients invoke operations on their proxies, which in turn do the work: construct a Request object, marshal arguments, send over the network, await response, and return it to the client. Meanwhile, the client application code is blocked on the request until completion.. Note, of course, that this is the same Basic Paradigm of all RPC (Remote Procedure Call) based systems.

In many cases, the synchronous paradigm is sufficient, and greatly eases development since it extends distributed computing in a straightforward manner from normal non-distributed function calls. However, there are many situations in which the synchronous paradigm is not sufficient. Examples of this include most distributed multimedia applications. There are three CORBA-standardized mechanisms that extend beyond the synchronous paradigm: one-way, DII deferred invocation, and the OMG Event service.

One-way is merely an IDL keyword that identifies an operation as flowing exclusively in one direction. There is no CORBA requirement for the underlying operation to be non-blocking. CORBA says only that the delivery semantics are “best effort”, which implies that the efficiencies of non-blocking sends are allowed, but are not required, leaving the determination up to individual implementations.

DII Deferred Synchronous Invocation offers a deferred mode, via *send* and *get_response* DII operations, to separate the act of sending a message from the act of obtaining its response.

Events in CORBA *Event Channel* are not part of the ORB itself, or a transport mechanism underneath it, but rather are considered as a layered service above the ORB. Nonetheless, this is a fairly fundamental service in that it offers perhaps the richest CORBA-supported alternative to the synchronous paradigm. Basically, Events offer a publish/subscribe model of messaging between directly coupled clients and servers. The real power of Events comes from the injection of an Event Channel between the client and server.

Asynchronous messaging is considered a highly scalable solution for deployment of large scale distributed multimedia systems. Therefore, it is a key extension in CORBA 3.

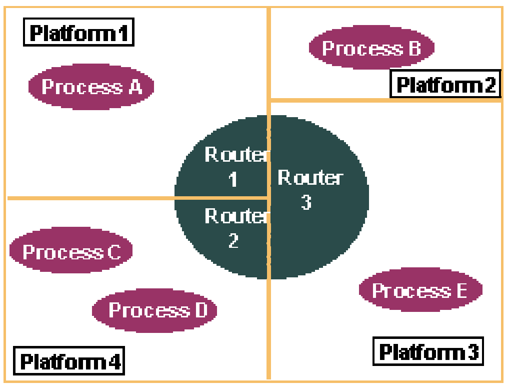

Fig. 3

The next generation of distributed object applications requires a new way of thinking about the CORBA model as its services are expanded and integrated with other technologies such asynchronous messaging. CORBA today does a great job of hiding the details of object distribution, as it was intended. CORBA's infrastructure is hidden under an object model that makes distribution transparent. It truly behaves as though objects were local. But they are not. They are often distributed across a wide, disperse network. Because the objects reside on a variety of distributed systems, load balancing and message replication become the responsibility of the application. This means that applications now have to be aware of infrastructure ñ the very aspect that CORBA was trying to hide. By integrating asynchronous messaging into CORBA, infrastructure details can be hidden while preserving the CORBA object model.

Since asynchronous messaging is considered a highly scalable solution for deployment of large scale distributed multimedia systems, it is considered a key extension in CORBA 3. This is accomplished with software routers and by renaming remote objects with a logical name instead of a hard-coded host name and port number.

To overcome the limitations of a tightly bound object reference and object implementation, software routers are introduced (Fig.3). It has been said that any software problem can be solved with enough layers of indirection. This is precisely what the software routers do. They de-couple the object reference and object. With this architecture, the client maintains a single connection to the router, instead of having a network connection from a client process to all server processes. The router maintains a routing table that directs requests from a client to the correct server process, resulting in a loose coupling between the object reference and object implementation. This provides a vastly more scalable architecture, because a single connection to the router is maintained regardless of the number of server processes it communicates with.

When a client invokes a method on the ambiguous object reference, the message is actually sent to the software router. The software router looks at the *QoS* specification and determines which objects should receive the message. The router then fans out the message to all qualifying objects. To the developer of the client there is no difference in programming. The infrastructure details are completely hidden, as they should be. Load balancing of client requests has received much attention and fostered much discussion. However, current load balancing schemes tend to distribute the connections evenly across the servers, without considering the actual load generated by the clients. While connection distribution is better than no solution at all, it is far from what is required for reliable, high performance load balancing.

Consider what would happen if you had two servers, with one at idle and the other running at capacity. A simple round-robin load-balancing algorithm would send some connections to the server running at capacity. This is not the desired behavior and not what anyone would consider good load balancing. What is required is real-time load balancing. Every time a client request is made, it should go to the least loaded server, not the one to which the client is randomly connected. Fig. 4 illustrates how load balancing is achieved. When the client issues the request and sets the *QoS* to load balance, the router determines which server is least loaded and directs the message to that server only. The servers are the elements that determine the load balancing policy.

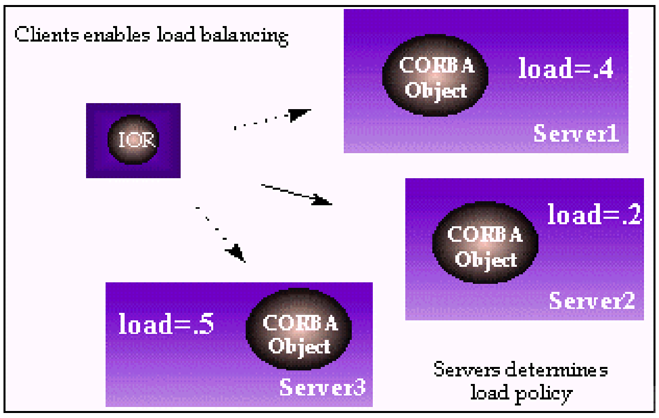

Fig. 4

To summarize, CORBA 3 introduces two new message specification:

- callback (additional object reference, invoked when the answer arrives)
- polling (value type returned, used to poll or wait for response)

5. Conclusions

Considering the state-of-the-art in implementing Network Based Training systems , the use of actual CORBA technology brings:

- Advantages with respect to native Web tools (based on HTML dialog)
- Advantages with respect to native Java tools (based on RMI dialog)

Future Network Based Training systems, based on specialized networks which allow greater service quality, will enhance their functions adopting multimedia paradigms. In this case, the actual CORBA technology is limited, while CORBA 3 extensions are intended to support also multimedia functions.

References

[1] Buzzi, M.C. and Venerosi, P., "Internet technology supporting distance education: a critical overview", Technical Report CNUCE-B4-1998-002, Jan. 1998.

[2] Mobray, T.J. and Ruh, W.A., "Inside CORBA: Distributed Objects Standards and Applications", Addison Wesley, 1997

[3] Pope, A., "The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture, Addison Wesley, 1998

[4] Gemmell, S. and S. Christodoulakis, S., "Principles of Delay Sensitive Data Storage and Retrieval", ACM Transactions on Information Systems , 10(1), Jan. 1992.

[5] Vinoski, S., "New Features for CORBA 3.0", Communications of the ACM, 41(10), pp.44-52, Oct.1998.