A Brief Tutorial on CORBA

Kate Keahey (kksiazek@cs.indiana.edu)

This tutorial aims to give a short, practical introduction to the The Object Management Group's Common Object Request Broker Architecture (CORBA). It should provide a good understanding of the basic mechanics of the architecture, give a rough overview of its components and provide the reader with some vocabulary used in the OMG documents which are the most reliable resource of information about CORBA. At the end it also contains references to systems similar to CORBA, and some of the research connected with it.

For more information on CORBA take a look at the links to related sites at the end of this page. The newsgroup comp.object.corba provides a good discussion forum. If you have questions or comments about this tutorial, please send me mail .

The Object Management Group (OMG)

The Object Management Group, is a non-profit consortium created in 1989 with the purpose of promoting theory and practice of object technology in distributed computing systems. In particular, it aims to reduce the complexity, lower the costs, and hasten the introduction of new software applications. Originally formed by 13 companies, OMG membership grew to over 500 software vendors, developers and users.

OMG realizes its goals through creating standards which allow interoperability and portability of distributed object oriented applications. They do not produce software or implementation guidelines; only specifications which are put together using ideas of OMG members who respond to Requests For Information (RFI) and Requests For Proposals (RFP). The strength of this approach comes from the fact that most of the major software companies interested in distributed object oriented development are among OMG members.

The Object Management Architecture (OMA)

OMA is a high-level vision of a complete distributed environment. It consists of four components that can be roughly divided into two parts: system oriented components (Object Request Brokers and Object Services), and application oriented components (Application Objects and Common Facilities).

Of these parts Object Request Broker is the one which constitutes the foundation of OMA and manages all communication between its components. It allows objects to interact in a heterogeneous, distributed environment, independent of the platforms on which these objects reside and techniques used to implement them. In performing its task it relies on Object Services which are responsible for general object management such as creating objects, access control, keeping track of relocated objects, etc. Common Facilities and Application Objects are the components closest to the end user, and in their functions they invoke services of the system components.

The Common Object Request Broker (CORBA)

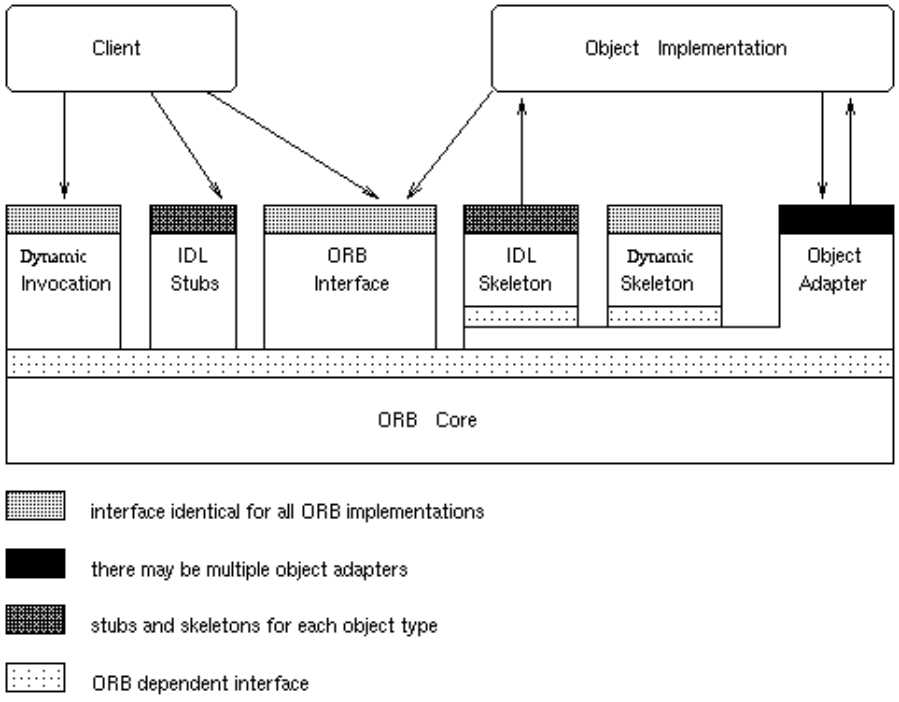
CORBA specifies a system which provides interoperability between objects in a heterogeneous, distributed environment and in a way transparent to the programmer. Its design is based on OMG Object Model.

The OMG Object Model

The OMG Object Model defines common object semantics for specifying the externally visible characteristics of objects in a standard and implementation-independent way. In this model *clients* request services from *objects* (which will also be called servers) through a well-defined interface. This interface is specified in OMG IDL *(Interface Definition Language)*. A client accesses an object by issuing a *request* to the object. The request is an event, and it carries information including an operation, the *object reference* of the service provider, and actual parameters (if any). The object reference is an object name that defines an object reliably.

The Basic Mechanics of issuing a request

The picture below shows the main components of the ORB architecture and their interconnections:



The central component of CORBA is the *Object Request Broker* (ORB). It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. In general, the ORB is not required to be a single component; it is simply defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for communication of requests.

The basic functionality provided by the ORB consists of passing the requests from clients to the object

implementations on which they are invoked. In order to make a request the client can communicate with the ORB Core through the IDL *stub* or through the *Dynamic Invocation Interface* (DII). The stub represents the mapping between the language of implementation of the client and the ORB core. Thus the client can be written in any language as long as the implementation of the ORB supports this mapping. The ORB Core then transfers the request to the object implementation which receives the request as an up-call through either an IDL skeleton, or a dynamic skeleton. Exactly what happens is best explained by a simple, concrete example.

Overview of Architectural Components

The communication between the object implementation and the ORB core is effected by the *Object Adapter* (OA). It handles services such as generation and interpretation of object references, method invocation, security of interactions, object and implementation activation and deactivation, mapping references corresponding to object implementations and registration of implementations. It is expected that there will be many different special-purpose object adapters to fulfill the needs of specific systems (for example databases).

OMG specifies four policies in which the OA may handle object implementation activation: *Shared Server Policy*, in which multiple objects may be implemented in the same program, *Unshared Server Policy*, *Server-per-Method Policy*, in which a new server is started each time a request is received, and *Persistent Server Policy*. Only in the Persistent Server Policy is the object's implementation supposed to be constantly active (if it is not, a system exception results). If a request is invoked under any other policy the object will be activated by the OA in the policy specific way. In order to be able to do that, the OA needs to have access to information about the object's location and operating environment. The database containing this information is called *Implementation Repository* and is a standard component of the CORBA architecture. The information is obtained from there by the OA at object activation. The Implementation Repository may also contain other information pertaining to the implementation of servers, such as debugging, version and administrative information.

The interfaces to objects can be specified in two ways: either in OMG IDL, or they can be added to *Interface Repository*, another component of the architecture. The *Dynamic Invocation Interface* allows the client to specify requests to objects whose definition and interface are unknown at the client's compile time. In order to use DII, the client has to compose a request (in a way common to all ORBs) including the object reference, the operation and a list of parameters. These specifications --- of objects and services they provide --- are retrieved from the Interface Repository, a database which provides persistent storage of object interface definitions. The Interface Repository also contains information about types of parameters, certain debugging information, etc.

A server side analogue to DII is the *Dynamic Skeleton Interface (DSI)*; with the use of this interface the operation is no longer accessed through an operation-specific skeleton, generated from an IDL interface specification, instead it is reached through an interface that provides access to the operation name and parameters (as in DII above the information can be retrieved from the Interface Repository). Thus DSI is a way to deliver requests from the ORB to an object implementation that does not have compile-time knowledge of the object it is implementing. Although it seems at the first glance that this situation doesn't happen very often, in reality DSI is an answer to interactive software development tools based on interpreters and debuggers. It can also be used to provide inter-ORB interoperability which will be discussed in the next section.

Interoperability

There are many different ORB products currently available; this diversity is very wholesome since it allows the vendors to gear their products towards the specific needs of their operational environment. It also creates the need for different ORBs to interoperate. Furthermore, there are distributed and/or client/server systems which are not CORBA-compliant and there is a growing need for providing interoperability between those systems and CORBA. In order to answer those needs OMG has formulated the ORB interoperability architecture.

Implementational differences are not the only barrier that separates objects; other reasons might include strict enforcement of security, or providing a protected testing environment for a product under development. In order to provide a fully interoperable environment all those differences have to be taken into account. This is why CORBA 2.0 introduces the higher-level concept of a domain, which roughly denotes a set of objects which for some reason, be it implementational or administrative, are separated from some all other objects. Thus, objects from different domains need some *bridging mechanism* (mapping between domains) in order to interact. Furthermore, this bridging mechanism should be flexible enough to accommodate both the scenarios where very little or no translation is needed (as in crossing different administrative domains within the same ORB), but efficiency is an issue, and scenarios which can be less efficient, but need to provide general access to ORB.

The interoperability approaches can be most generally divided into *immediate* and *mediated* bridging. With mediated bridging interacting elements of one domain are transformed at the boundary of each domain between the internal form specific to this domain and some other form mutually agreed on by the domains. This common form could be either standard (specified by OMG, for example IIOP), or a private agreement between the two parties. With immediate bridging elements of interaction are transformed directly between the internal form of one domain and the other. The second solution has potential to be much faster, but is the less general one; it should be therefore possible to use both. Furthermore, if the mediation is internal to one execution environment (for example TCP/IP) it is known as a *"full bridge"*, otherwise if the execution environment of one ORB is different from the common protocol we say that each ORB is a *"half bridge"*.

Bridges can be implemented both internally to an ORB (say just crossing administrative boundaries), or in the layers above it. If they are implemented within an ORB they are called *in-line* bridges, otherwise they are called *request-level* bridges. The in-line bridges can be implemented through either requiring that the ORB provide certain additional services or through introducing additional stub and skeleton code. Interacting through the request-level bridges goes roughly like that: the client ORB "pretends" that the bridge and the server ORB are parts of the object implementation and issues a request to this object through the DSI (remember, DSI needn’t know the specification of its object at compile time). The DSI, in cooperation with the bridge, translates the request to a form which will be understood by the server ORB and invokes it through DII of the server ORB; the results (if any) are passed back via a similar route. Naturally, in order to perform its function the bridge has to know something about the object; thus if either needs to have access to the Interface Repository, or be only an interface specific bridge, with the applicable interface specifications "hardwired" into it.

In order to make bridges possible it is necessary to specify some kind of standard transfer syntax. This function is fulfilled by General Inter-ORB Protocol (GIOP) defined by the OMG; it has been specifically defined to meet the needs of ORB-to-ORB interaction and is designed to work over any transport protocol that meets a minimal set of assumptions. Of course, versions of GIOP implemented

using different transports will not necessarily be directly compatible; however their interaction will be made much more efficient.

Apart from defining the general transfer syntax, OMG also specified how it is going to be implemented using the TCP/IP transport and thus defined the Internet Inter-ORB Protocol (IIOP). In order to illustrate the relationship between GIOP and IIOP, OMG points out that it is the same as between IDL and its concrete mapping, for example C++ mapping. IIOP is designed to provide "out of the box" interoperability with other compatible ORBs (TCP/IP being the most popular vendor-independent transport layer). Further, IIOP can also be used as an intermediate layer between half-bridges and in addition to its interoperability functions, vendors can use it for internal ORB messaging (although this is not required, and is only a side-effect of its definition). The specification also makes provision for a set of environment-Specific Inter-ORB Protocols (ESIOPs). These protocols should be used for "out of the box" interoperability wherever implementations using their transport are popular.

Ongoing projects using CORBA

The success of the CORBA standard has spurred research in many domains based on this technology. Check out the KQML page to see their RFI on agent facilities. Integrating CORBA and the Web in order to provide transactional information is also the theme of another project . Look also at the Sunrise project in Los Alamos. I am working on PARDIS, a system extending the CORBA functionality to accommodate parallel processing.

Implementations of CORBA

There are many implementations of CORBA currently available; they vary in the degree of CORBA compliance, quality of support, portability and availability of additional features. Unfortunately, to date there are no fully compliant public domain implementations; ILU from Xerox Parc is probably the closest you will get to a free implementation of CORBA right now.

Orbix from Iona is a very solid, fully compliant commercial implementation with excellent support. VisiBroker from VISIGENIC is also 2.0 compliant and offers interoperability with Java. ObjectBroker from digital is 1.2 compliant. Other implementations include ObjectBroker from Digital, Expersoft’s XShell, a Distributed Object Oriented Management (DOME) system which also is CORBA compliant. HP’s DistributedSmalltalk product also implements OMG CORBA 1.1 Object Request Broker.

Similar projects

There are also many products which are not necessarily CORBA-compliant, or based on CORBA in any way. An example is LOG BROKER , which extends the C++ language to make it transparently support Distributed Object Computing. Other examples include Fresco, and ILU from Xerox Parc.

Further Reading

Apart from the OMG recommended readings and their documents mentioned at the beginning of the tutorial there are many other sources of information on CORBA. A lot of information is available on the Web; the LANL CORBA Web page is among the most comprehensive. In addition to many interesting web pointers it contains sources of many tasks and papers written on CORBA (C++ Report articles by

Steve Vinoski among others). Other popular articles can be found in Dr Dobb’s Journal, October 1994 ("Interoperable Objects" by Mark Betz) and the April ’95 issue of Byte (they have a whole Special Repoort section on Client/Server systems). Other interesting Web pages are Doug Schmidt’s CORBA page , the Object-Orientation FAQ, and Index to object-oriented and distributed systems.