

CORBA IIOP Protocol and Applications

Introduction

The Common Object Request Broker Architecture (CORBA) and the Internet Inter-ORB Protocol (IIOP) are two technologies that enable distributed computing across heterogeneous systems.

CORBA defines the total architecture required for communication between distributed objects and IIOP is the most important specification of CORBA. IIOP focuses on interoperability of distributed objects in heterogeneous environments. CORBA, created by the Object Management Group (OMG) in 1991, enables an application's components to communicate without regard for their locations on a network. A CORBA-compliant object is guaranteed to be able to communicate with other distributed objects because the technology defines a common interface. IIOP was introduced in Dec. 1994 as a component of CORBA 2.0 specification. Netscape and Sun Microsystems support CORBA and IIOP in their next-generation products.

Group project by: Fei Zhang
Chunsong Ji
Fenghua Ji
7/21/98

Background

Today, using of Internet and Web has been changing our life in all aspects. It also changes the way we use computers. One of the biggest changes is that it gives a new meaning to distributed computing. Instead of the traditional client/server network, the new era of distributed computing make it possible for enterprise systems, whose objects are distributed across multiple computers, that they can all communicate, regardless of operating system, platform, or programming language. Even more amazing is that these distributed objects can be components of a single application or of the dozens of applications that form the enterprise system. The application modular evolved from monolithic to multi-tiered.

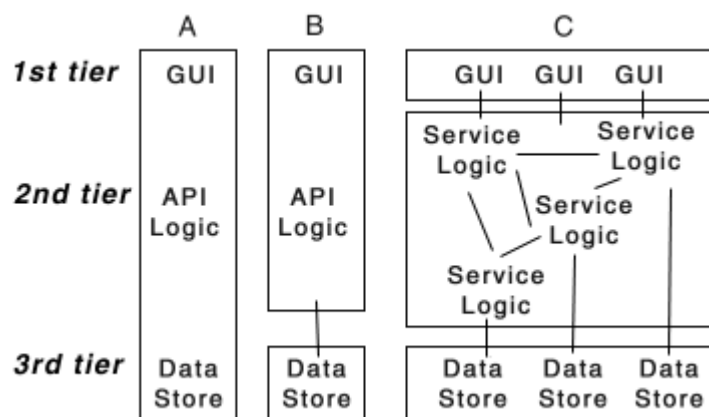


Figure 1 Application Modular: A, B--monolithic, C--multi-tiered

It is the technology such as the Common Object Request Broker Architecture (CORBA) and the Internet Inter-ORB Protocol (IIOP) that enables distributed computing across heterogeneous systems. CORBA specifies the complete architecture necessary for communication between distributed objects. The specification includes IIOP and a host of other technologies. IIOP is the most important piece of CORBA, because it focuses on interoperability of distributed objects in heterogeneous environments. In addition, CORBA and IIOP define the middleware that could cause developers around the world to rethink how they create applications in networked environments.

CORBA is a standard created by the Object Management Group. It is first implemented in 1991. The organization's goal is to provide a common framework for application development using object-oriented techniques. Instead of applications, OMG produces specifications that make distributed-object computing possible. By describing a common architecture for communication between distributed objects, CORBA makes it possible for an application's components to communicate, regardless of their locations on a network. Further, because CORBA defines a common interface between objects, the operating system and the programming language of the object don't matter. As long as an object is CORBA-compliant, it can communicate with other distributed objects.

IIOP is another success standard of OMG. It is introduced as part of the CORBA 2.0 specification in December 1994. Before IIOP, the CORBA specification defined interaction only for distributed objects created by the same vendor; the objects had to be designed for a specific implementation. Using IIOP, the second CORBA specification became the definitive solution for object interoperability that wasn't tied to a specific platform or implementation.

Inside CORBA

As a technology that provides the framework for interaction between dissimilar objects, CORBA succeeds admirably, yet that is only a piece of a greater picture called the Object Management Architecture (OMA), the primary components of which are:

- CORBA ORB--handles requests among objects;
- CORBA Services--defines system-level services that help manage and maintain objects;
- CORBA Facilities--defines facilities and interfaces at the application level;
- application objects--the objects themselves.

ORB

The Object Request Broker (ORB) is the heart of CORBA. It is the object bus. The ORB allows objects to make requests of each other. Although the ORB operates in a client/server environment, objects in ORB can function as either clients or servers, depending on the circumstances. If an object is receiving and processing a request, then it is acting as a server. If the object is making a request, then it is acting as a client.

The ORB lets objects transparently make requests to and receive responses from other objects located locally or remotely. The client is not aware of the mechanisms used to communicate with, activate, or store the server objects. The ORB lets objects discover each other at runtime and invoke each other's services. Brokering requests and returning results is all the job for the ORB. It involves intercepting each request from one object to another, locating the object that's supposed to handle the request, invoking the appropriate method in the receiving object, passing parameters if necessary, and returning the result to the object that made the request. Because the ORB handles requests transparently, it doesn't matter whether the request is from a local or a remote object.

ORBs are the common denominators that bridge the differences in location, platform, and programming language that can separate a client and a server. ORBs can contact each other across the network, can create and interpret object references, and can marshal parameters into and out of the format used by IIOP (Internet InterORB Protocol).

The ORB is an architectural abstraction rather than a process executing on a specific machine. In practice, this abstraction is usually implemented as a library linked into clients and servers. An ORB daemon can optionally support the automatic launching of ORB servers for persistent objects.

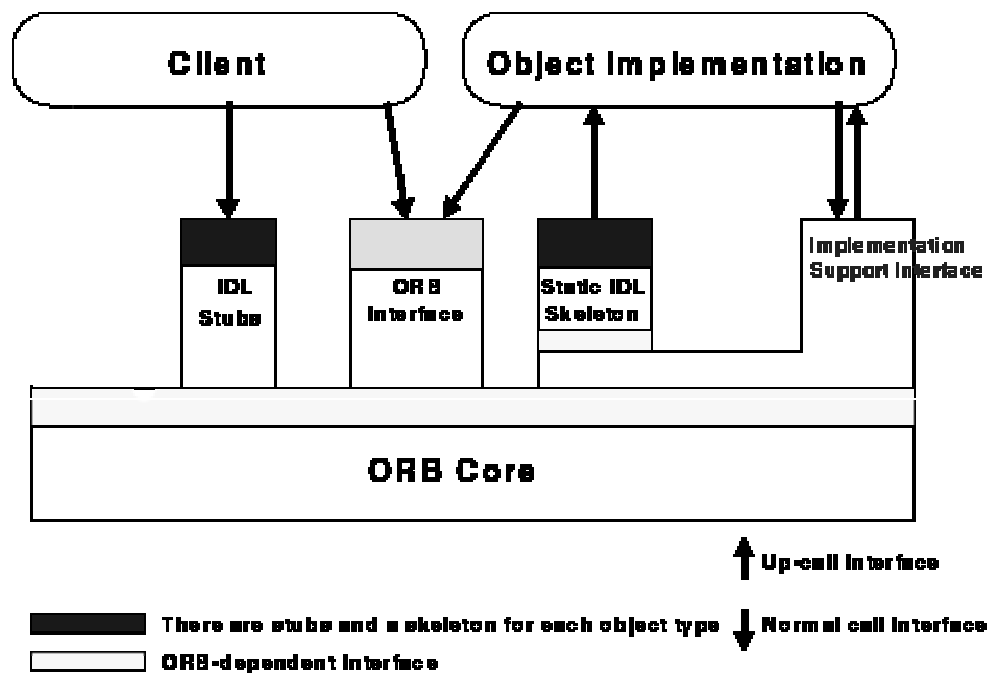


Fig 2, Structure of Object Request Broker interface

IDL

The ORB handles these requests regardless of programming language, operating system, or platform. The mechanism that allows ORBs to handle requests transparently is the Interface Definition Language (IDL), which is used to declare the boundaries and interfaces of an object. Much like an independent arbitrator, the IDL is neutral and independent of the object and the ORB, yet it binds providers of distributed-object services to their clients. CORBA supports multiple inheritance, and its IDL uses inheritance to encapsulate objects. This makes reuse of code very easy.

IDL expresses operating system and programming language-independent interfaces to all the services and components that reside on a CORBA bus. It allows client and server objects written in different languages to interoperate across networks and operating systems. IDL is purely declarative; it provides no implementation details. IDL-specified methods can be written in and invoked from any language that provides CORBA bindings, including Java, C, C++, and Smalltalk. IDL is mapped into each programming language to provide access to object interfaces from that language. The advantage of IDL is that you can concisely define APIs yet still have the freedom to define the IDL methods in any programming language that provides CORBA bindings.

As anyone who has done object-oriented programming knows, you need to know a receiving object's interface before you can make a request, and you design your objects so that they know the interfaces of the objects they interact with. But with distributed computing between heterogeneous objects, you'll run into lots of problems if you try that approach.

To make IDL truly independent, CORBA uses an interface repository, the purpose of which is to store the method signatures of objects so that the signatures can be dynamically retrieved and updated at runtime. In this way, all objects in the enterprise system can learn about other objects' interfaces, the methods the interfaces support, and the parameters the interfaces require.

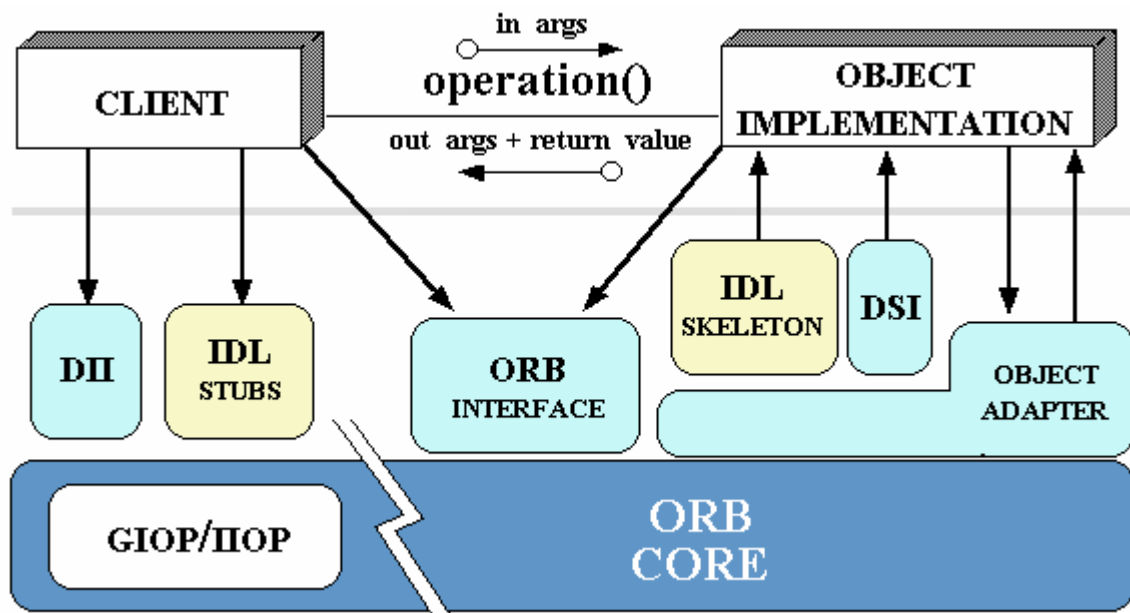


Figure 3 CORBA ORB Architecture

CORBA Object Services

CORBA services is used to manage and maintain objects through their life cycles. It provides the interfaces that can be used to create objects, handle object security, determine the location of objects, and perform class-level management of objects. Because they are system-level rather than application-level services, it can be implemented across the enterprise. It is easy to create consistency throughout the enterprise system. There are 16 object services, including:

Collection Properties

Concurrency control	Query
Event notification	Relationships
Externalization	Security
Licensing	Startup
Life cycle	Time
Naming	Trader
Persistence	Transactions

CORBA object services let developers focus their efforts on their objects, without having to worry about system-level services. You can develop a class for your object and then use the object services to add the functionality they provide. You handle this with subclassing and multiple inheritance. For example, if you create a Widget class, you can then create a subclass called aWidget that is persistent simply by inheriting from the Persistence service. You could also handle event notification, security, and queries by inheriting from the appropriate services.

CORBA extends the power of subclassing and multiple inheritance with metaclasses. A metaclass is a class of objects that can be created at runtime. Using metaclasses, you can dynamically add services to CORBA objects, allowing you to customize objects on demand.

CORBA Object Facilities

CORBA Facilities define frameworks that provide interfaces and other services. These frameworks are divided into broad categories that include application interfaces, domain interfaces, and networking facilities.

Application interfaces consist of non-standardized facilities specific to individual applications, such as a reservation system or inventory management system. The domain interfaces consist of facilities for end users in specific application domains. (Here a domain refers to a specific vertical market or industry.)

Whereas domain interfaces focus on specific markets, Internet facilities are the application-level functions and interfaces that reach a broad market. These facilities include common user-oriented tasks within applications, such as printing or saving a file, yet also reach into common networked tasks, such as using e-mail and networking facilities.

Application Objects

Application objects are the actual objects that form the core of a CORBA-compliant application. They are also called business objects. A business object is a way of describing concepts that are independent of an application, such as a customer, order, or payment. You will use a collection of business objects to create an application.

Unlike system-level objects, which handle tasks such as persistence, business objects handle real-world business processes. Although a typical business object may handle only a single task (such as working with reservations, customers, or

inventory), collections of business objects can be used to handle an entire business process, such as booking and managing reservations for an airline. Because business objects come with ready-to-use functionality, a developer can stitch them together to create a custom application.

In order to manage the complexity of an application with distributed business objects, the business objects know only what services other objects provide, not how those services are actually implemented. Business objects hide the complexities of back-end processing by focusing on interfaces rather than on implementation.

COBRA/IOP

OMG create standards for distributed object computing that are realistic, commercially available and usable through its Object Management Architecture (OMA) at the heart of which is the Common Object Request Broker Architecture (CORBA). CORBA specifies the Object Request Broker (ORB) that allows applications to communicate with one another no matter where they reside on a network. The CORBA 2.0 specification, adopted in 1994, created true out-of-the-box interoperability in heterogeneous environments and delivers this interoperability over the Internet through IOP – the Internet Inter-ORB Protocol.

A common misconception about IOP is that it is a "separate" specification that developers need to write in order to allow CORBA to work over the Internet. This is not so. A properly constructed CORBA 2.0 ORB already incorporates IOP. IOP is an underlying mechanism of CORBA technology which is transparently managed by ORBs. So IOP and CORBA are, essentially, inseparable. Therefore, programmers and users are never required to interact with IOP in any way; it is invisible to them. IOP allows their programs to interact transparently while executing, so one does not have to write "IOP programs."

The IOP specification defines a set of data formatting rules, called CDR (Common Data Representation), which is tailored to the data types supported in the CORBA Interface Definition Language (IDL). Using the CDR data formatting rules, the IOP specification also defines a set of message types that support all of the ORB semantics defined in the CORBA core specification. Together, the CDR formatting rules and the message formats constitute an abstract protocol called GIOP, which stands for General Inter-ORB Protocol. GIOP messages can be sent over virtually any data transport protocol, such as TCP/IP, Novell SPX, SNA protocols, etc. To ensure "out-of-the-box" interoperability between ORB products, the IOP specification requires that ORBs send GIOP messages over TCP/IP connections because TCP/IP is the standard connection-oriented transport protocol for the Internet. To put it very simply, GIOP + TCP/IP = IOP.

Objects publish their identities and locations in the form of object references. The CORBA 2.0 specification dictates a common format for object references exchanged over IOP, called IOR (Interoperable Object Reference) format. An IOR contains one or more profiles. Each profile describes how a client can contact and send requests to the object using a particular protocol. All legal IORs must have at least one IOP profile, thus ensuring that wherever that reference goes, any CORBA-compliant ORB will be able to locate the object and send

requests to it. The IOP profile contains the Internet address of the object's server and a key value used by the server to find the specific object described by the reference.

Object references can be converted into character strings, which can be published arbitrarily, like URLs, in email messages, files, databases, directories, and so on. Any CORBA-compliant application can convert the string into an IOR and use it to locate and invoke the object.

When a client program built with ORB vendor B's product needs to talk to an object in a server built with ORB vendor A's product, the client program opens a TCP/IP connection to the server, and sends one or more IOP request to the server. The ORB component linked into the server locates or activates the object specified in the request and invokes the appropriate method on the object. The fact that the object is not built with the same ORB product is invisible to the client.

One of the most important aspects of CORBA/IOP is its platform independence. CORBA ORBs interoperate without regard to vendor origin making CORBA/IOP the truly ideal solution for the Internet, especially when one considers its vast size and the disparate hardware and software that is deployed there. This platform independence allows businesses to take advantage of the Internet without having to rebuild systems and networking hardware and software or forcing them to commit to a single vendor solution. CORBA/IOP will allow most every system that is now in operation to be incorporated with comparatively minor modifications so that a business's installed base, even if it is made up of equipment and software packages from a variety of vendors, will be able to work together seamlessly.

IOP, the Internet Inter-ORB Protocol, is a projection on TCP/IP of the more general GIOP. GIOP, the General Inter-ORB Protocol which is defined in the CORBA 2.0 specifications, allows different ORBs to work together. The GIOP specification consists of three main elements:

- The transport management requirements
- The GIOP message formats
- The CDR, Common Data Representation, transfer syntax definition

With respect to GIOP, IOP is not a distinct interoperability protocol. IOP implements GIOP over the TCP/IP transport layer. It extends the GIOP specification by mapping GIOP connections to TCP/IP socket connections.

It is only possible for two ORBS to work together, after both the client and the server have an ORB. It is intuitive that the server would be set up with an ORB already installed and running, but not so intuitive that everyone who is going to access the environment has installed CORBA, so to solve this We just have to use an ORB written in Java, which will be in a class format like any other Java objects. We will then download the ORB within the Web browser, along with the applet that will use it. Those ORBs written in Java are generally called Java ORBs. `` ORBlets" is also sometimes used for Java ORBs that are downloaded like an applet in a Web browser.

Then the Web users can first use HTTP to download texts, images and applets, and then use IOP for Java-CORBA, client-server communications. Moreover, with IOP, the server programs are not limited to be located on the same machine

as the Web server. This means that unlike CGI applications and traditional Java, clients can communicate with any machine on the network, under the only condition that it supports IOP.

With IOP, Web applications are not locked into a single machine that must manage both requests of HTML files and executions of server programs, either through CGI or Java. A Web server can now be installed on a dedicated Internet host that is free to serve only incoming HTTP requests, while all the other client/server applications are run on different machines.

Application of CORBA

Employing CORBA, you can create an enterprise-wide system with objects distributed throughout the network. Within the enterprise system, your Windows NT server could store core file-services objects used by the application. These objects could be programmed in C++. Your mainframe could store the application's primary back-end functions--perhaps using objects programmed in COBOL. Each of your desktop systems running Windows 95 could store the application's front-end interfaces, using objects created in Visual Basic. And all of the objects could communicate by brokering requests with CORBA.

IBM, Netscape Communications Corp., Oracle Corp., and Sun Microsystems Inc. – jointly are going to make enhancements to the CORBA specification in an attempt to create a component model that better interacts with a number of other component models, most notably JavaBeans (attempt to fight off Microsoft Corp.). Sun uses CORBA and IOP to implement heterogeneous remote procedure calls in Java 1.1, and without those technologies, the Java Platform for enterprise networks wouldn't be possible. Further, the Enterprise JavaBeans API will use CORBA and IOP to allow scalable business applications with reusable server components.

There are many successful stories of companies use CORBA to solve problems or development software. For example, CNN Interactive, a division of Cable News Network from Atlanta, GA is just one of the benefices. It has built and is deploying a system using CORBA (Common Object Request Broker Architecture) to gather, store and disseminate news material. The project has enabled the organization to tie together its various clients and servers that include Windows and Mac desktops, Windows NT, Sun Solaris and Netscape server platforms so that all internal users can access the material and all servers can talk to one another. CNN Interactive has realized many benefits through its use of CORBA: Ability to build both clients and servers simultaneously and IDL interfaces allow them to be married together quickly and seamlessly. Time savings, ability to build clients and servers simultaneously is allowed because IDL interface works so well. Shortened development cycle Object reusability saves time, money and minimizes debugging of new programs.

CORBA/IOP: Empowering the Web

As the Internet and network computing evolved, protocols were needed to allow applications to work over these networks. As an application specific (actually display specific) protocol HTTP is fine but as web applications become more complex, they are running up against the limitations of HTTP. Any problem that

requires dynamic interaction with the user, such as an on-line reservation system, exposes some of the problems inherent in using HTTP to build non-trivial applications. The available mechanisms, such as CGI and NSAPI, are not object-oriented -- they do not enforce any type-safety. There is no standard way of defining the service interfaces, so there is no repository of interfaces or self-description capability. Consequently, developers have to write more code with clumsier interfaces, the applications are more error-prone, they are harder to maintain, and are not likely to permit any meaningful re-use. So as web-based applications become more complicated CORBA/IOP becomes the natural solution for handling them. Using CORBA means that applications can communicate with each other over the Internet no matter where they are or who built them.

For the most part, moving from HTTP to IOP will be transparent to end-users, except for the fact that with IOP the applications they use will become more sophisticated and have better performance. Initially, web applications that use IOP will most likely take the form of Java applets downloaded via HTTP. Once the applet is downloaded, it takes over communication with remote objects using IOP. The CORBA model and tools that support it allow programmers to build this kind of application more quickly, develop more functionality, and make use of existing components.

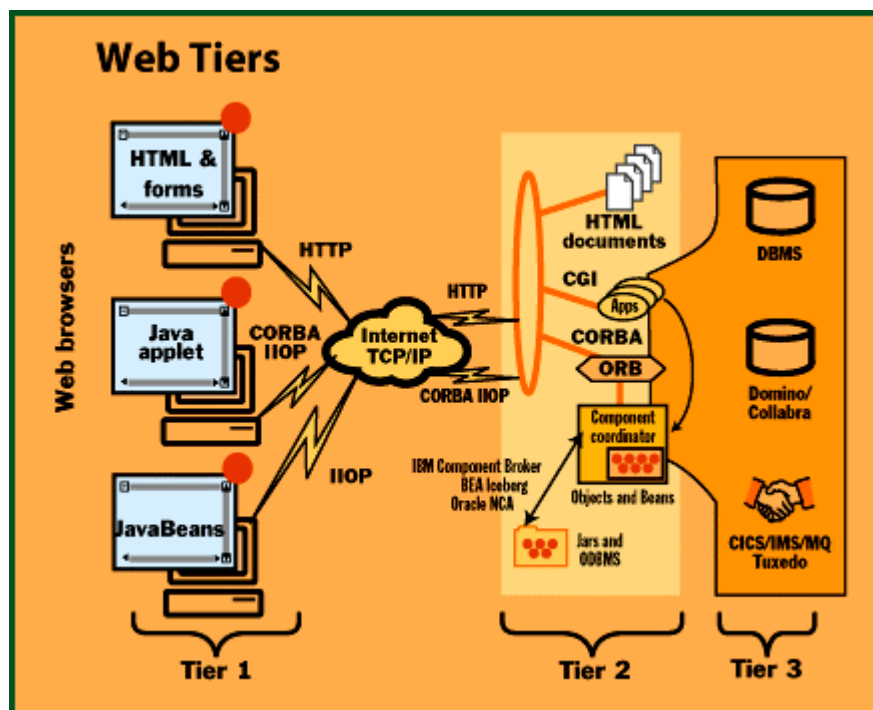


Figure 4. Web tiers using CORBA/IOP

Compare IOP with DCOM

CORBA and IOP aren't the only distributed-computing solutions. Microsoft has a competing architecture called the Distributed Computing Object Model (DCOM), basically an Object Request Broker (ORB) that is part of Windows NT 4.0 and will be part of the next release of Windows.

It's possible to make remote procedure calls to Java using DCOM. The necessary hooks are available with Visual J++. But DCOM is available only on Windows NT 4.0 and Windows 95 (through an Internet Explorer Extension), which means you cannot use DCOM to implement communications to other operating systems. Microsoft has announced that DCOM will be ported to other platforms in the future.

IIOIP lets Netscape's Open Network Environment (ONE) communicate with enterprise-wide systems. IIOIP also lets programmers connect Java, JavaScript, and C or C++ code to enterprise-wide systems. In Netscape Navigator, IIOIP is the protocol within LiveConnect that lets applets, plug-ins, and scripts communicate with each other. If you like to see more information about the difference between DCOM and IIOIP, you can visit .

Using CORBA, you can easily scale up from a network of three computers to an Internet-size network. CORBA provides the framework that lets you connect objects programmed in different languages, and you can do so regardless of the platform or operating system for which the objects were designed, as long as CORBA mappings are available. Because CORBA can operate across heterogeneous platforms, it has an advantage over DCOM at the present time. But with the Microsoft powerhouse behind it, DCOM is sure to be a force to reckon with in the coming years.

As some expert said, CORBA has a handicap. ORBs based on CORBA are too difficult to implement and write applications for. As a result, they take up too much time and money for most users to work with. However, as we all know that there is an inherent immaturity to distributed object computing in general. The stranglehold for CORBA, DCOM, and even Distributed Java, is the lack of expertise in the mainstream marketplace for developing distributed systems. OMG is working to deal with the complexity issue. In June 1997, a Component Initiative was submitted to the OMG Technical Committee, the purpose of which is to ease application construction by composing components with a simple scripting language.

Summary

With its backing consortium of more than 700 companies, CORBA is anything but the latest craze to hit the market. CORBA and IIOIP are developments that are in the right place at the right time, and not a prototype churned out quickly in response to "web time" demands. From the beginning, CORBA/IIOIP was intended to meet even the needs of the large, multinational enterprise where robustness, scalability, security, and value are critical elements of their information systems. CORBA/IIOIP perfectly meets these criteria making it the architecture/protocol of choice for the future of World Wide Web.