technical overview | A White Paper

a corba primer

the **e**-business management company

Preface

The distributed and heterogeneous nature of today's computing systems requires a middleware infrastructure capable of supporting a three-tier computing architecture. Business logic can be built, or existing applications encapsulated, into middle tier components that interact with end users via standard interfaces such as web browsers and standard GUI desktops, and back-end data repositories.

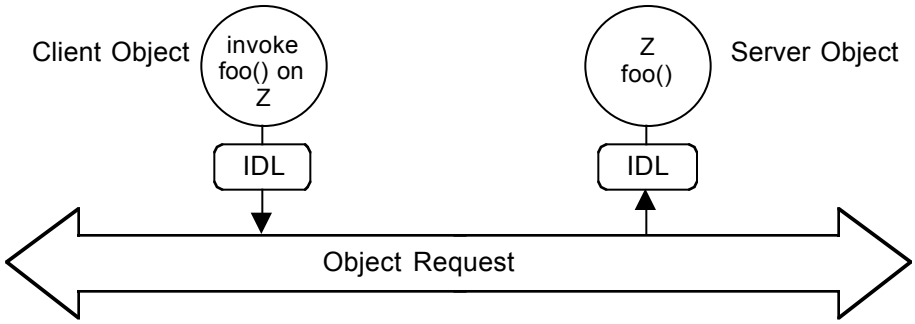
The Common Object Request Broker Architecture, or CORBA for short, is a specification produced by the Object Management Group (OMG) that addresses interoperability in distributed heterogeneous environments. The CORBA standard represents industry consensus from more than 800 companies. It defines an Object Request Broker (ORB) for transparent invocation on remote objects, as well as supporting system level Object Services and higher level Common Facilities. CORBA assumes a heterogeneous environment in which objects implemented in different languages on different platforms can interoperate There are many implementations of the CORBA standard, some of them in the form of commercial products that have demonstrated strong market acceptance.

This paper presents a technical introduction to CORBA. For additional technical information about successfully deploying applications built on a CORBA infrastructure, please refer to the *CORBA Products Technical White Paper* from Segue Software.

Introduction to CORBA

CORBA can be conceptualized as a communication bus for client-server objects. Since CORBA is a three-tier distributed object mechanism, the terminology "client-server" applies within the context of a specific request. That is, if object A invokes a method on object B, A is the client and B is the server. If B then calls A, the roles are reversed. Exported server interfaces must be specified in the CORBA standard Interface Definition Language (IDL). An IDL interface description is then mapped using an IDL compiler to native language bindings such as Java, C++ and others. This allows each programmer to write source code independently in the most appropriate language. For example, a server object implemented in C++ can be accessed by a Java applet. The applet programmer invokes methods on the server as though they are local Java method calls.

The ORB is the mediator, responsible for brokering interactions between objects. Its job is to provide object location and access transparency by facilitating client invocations of methods on server objects. A client can connect – or bind – to a server object statically if the server interface is known at build time. Alternatively, the client can use dynamic binding to ascertain the server interface and construct a call to the server. The following diagram illustrates a client invoking a server method.



CORBA client invoking a server method.

The ORB is just one element within the OMG's Object Management Architecture. Also specified are interfaces for lower level Object Services and higher level Common Facilities for building robust applications and systems. Examples include the naming service for locating objects, event service for subscribing to and publishing messages, security, transaction support, and object license and metering.

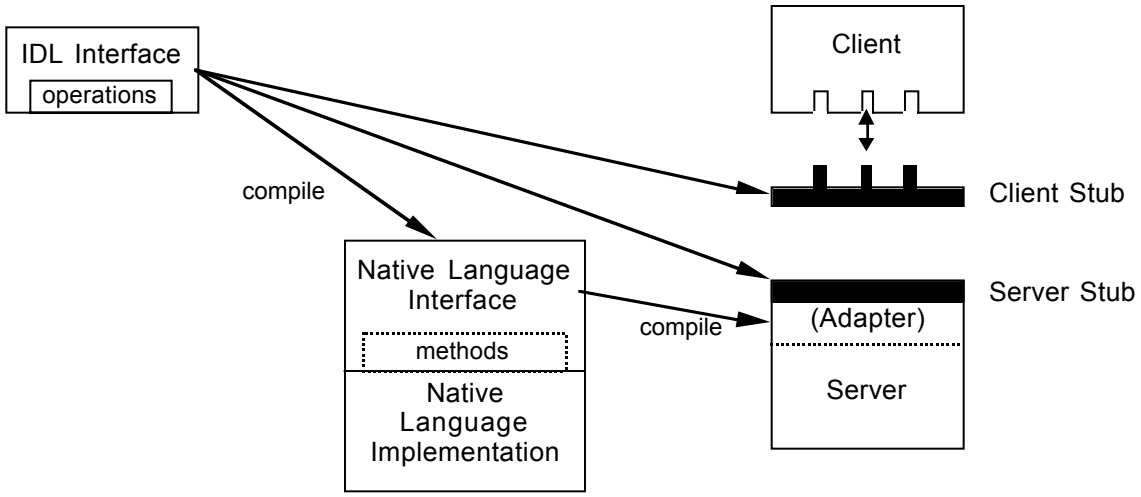
Server Side

The server side of CORBA begins with the specification of an interface in IDL. IDL is an object-oriented declarative language for specifying server interfaces. It is *not* a programming language; that is, server implementations are not written in IDL. Rather, IDL interface descriptions are mapped to a programming language for implementing the server. In IDL an `interface` corresponds to a class. A property is defined as an `attribute`, which is mapped by the IDL compiler to get and set methods; a `readonly attribute` maps to a get method only. An `operation` corresponds to a method; parameters must be specified as `in`, `out`, or `inout`. The following example illustrates an IDL interface description.

```
// IDL
interface Account
{
       //Attributes
       attribute float balance;
       readonly attribute string owner;
       //Operations
       void makeDeposit(in float amount,
                        out float newBalance);
       void makeWithdrawal(in float amount,
                        out float newBalance);
};
```

Many other features are supported by IDL, such as inheritance for specifying derived interfaces, modules for establishing interface naming scopes, definition of exceptions supported by an interface and which operations can raise them, and type codes supporting the creation of self-describing data.

An IDL compiler is used to generate a *server stub*, also referred to as a *skeleton*, which gets linked to the server program. The server stub provides static interfaces to call methods of an object implementation. It unmarshals methods and parameters that come from the client via the ORB. The IDL compiler also generates a native language interface for implementing the server, as well as a client side stub.



Generation of stubs from IDL description.

CORBA also supports a *dynamic skeleton interface (DSI)*, which is a run-time binding mechanism for creating server interfaces on the fly. An example where this could be used is for a schema mapper that dynamically generates implementation classes from a source schema such as a relational database, and then exports IDL interfaces for the classes using the DSI.

An important component of the ORB is the *object adapter*, which is responsible for managing server objects and related object references. It uses the CORBA *implementation repository* to locate a server name and launch command, identify the activation mode, and determine the launch and access permissions. Server *activation modes* define the run-time characteristics of a server, including whether a server process can contain one or more objects and if multiple clients are permitted to access the same object.

Client Side

As mentioned in the Server Side discussion above, the IDL compiler generates a *client stub* that gets linked to a program wishing to statically invoke a server method through the associated interface. The client stub maps a CORBA server side object to a native object in the client's language. It acts as a proxy for remote server objects, marshaling methods and parameters to be transmitted via the ORB.

CORBA also supplies the *dynamic invocation interface (DII)* for client programs to discover server interfaces at run-time. The CORBA *interface repository (IFR)* contains compiled IDL descriptions that can be interrogated via the DII. For example, a client may present a graphical user interface that is dynamically created based on available IDL interfaces. For each operation supported in an interface, the client program presents a push button that the user selects to invoke the operation. Input parameters are presented as text boxes or toggles, and output parameters as scrolled lists.

A client can locate a server object by obtaining an object reference directly from a server, by requesting an object by name via the CORBA naming service, or by asking the CORBA trader service to return references to objects that match some criteria. The CORBA standard specifies a format for an *interoperable object reference (IOR)* which can be passed across different ORB implementations. After obtaining an object reference, a client can invoke methods on a server object using the following call modes:

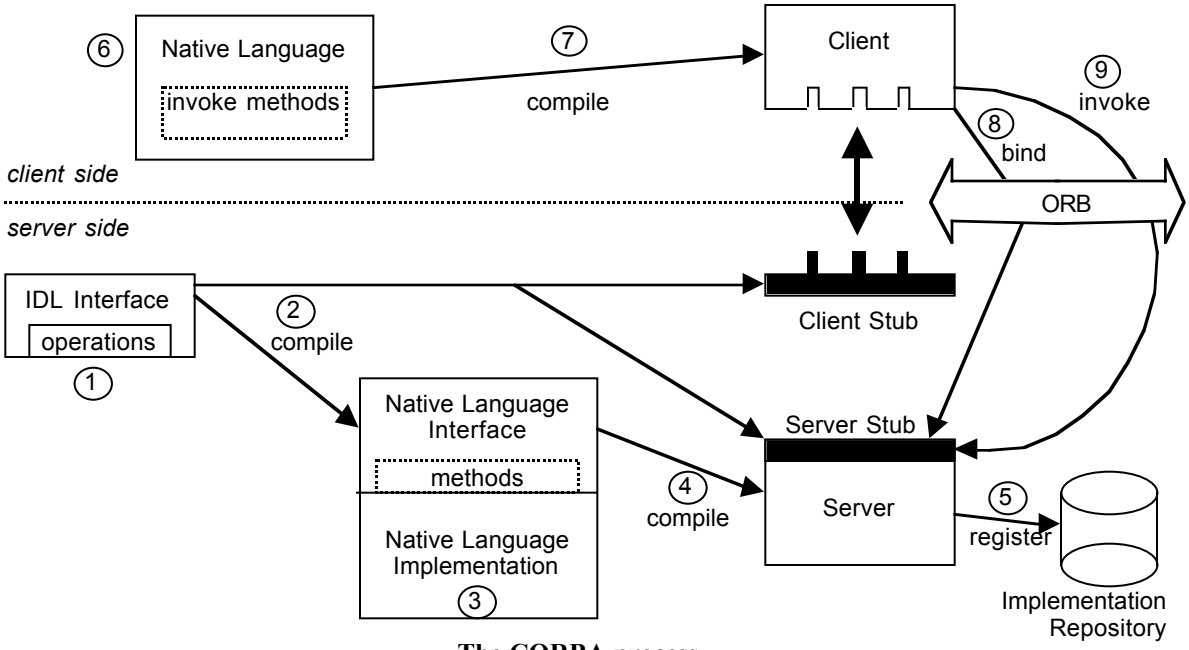
- **Synchronous:** The client sends a request and blocks until it receives a reply from the server.
- **One-way / Poll:** The client sends a request, but does not await a reply. Rather, the client polls until the server has completed servicing the request.
- **One-way / Callback:** The client sends a request, passing a callback object reference to the server. When the server is done, it invokes a method on the client's callback object. The implication of this approach is that the client acts like an event-driven server with an IDL interface.
- **One-way / Event Service:** The publish-subscribe model can be employed using the CORBA event service. The client first asks to be notified by the event service when a completion message is dispatched. The client then sends a request to the server. When the server completes processing, it sends a completion message to the event service, which in turn notifies the client.

IIOP: the Internet Inter-ORB Protocol

The CORBA standard guarantees interoperability between ORB implementations as well as applications built with different vendors' ORBs. The General Inter-ORB Protocol (GIOP) defines standard message formats, a common data representation for mapping IDL data types to flat messages, and a format for interoperable object references. The Internet Inter-ORB Protocol, commonly known as IIOP, defines GIOP message exchange over TCP/IP networks. In other words, IIOP is the CORBA wire level protocol for the Internet and intranets.

CORBA Process

The process for creating a static CORBA server and client is detailed in the following steps.



The CORBA process.

1. Specify the server interface in IDL.
2. Run the IDL description through an IDL compiler, which generates a native language interface, server stub, and client stub. The server and client stubs can be for different programming languages. During this step, the IDL compiler can optionally write a compiled interface description to the interface repository.
3. Implement the server.
4. Compile the server program and link in the server stub that was generated by the IDL compiler. The result is an executable server program that can accept method invocations via CORBA.
5. Register the server in the implementation repository, specifying server name, launch command, activation mode, and launch and access permissions. The server is now available for activation.

6. Implement the client. The programmer writes server object method calls as though they are local using the syntax and conventions of the client's native programming language.
7. Compile the client program and link in the client stub that was generated by the IDL compiler.
8. When the client is executing, it uses the ORB to bind to the server object and obtain an object reference.
9. Using the object reference, the client invokes server object methods.

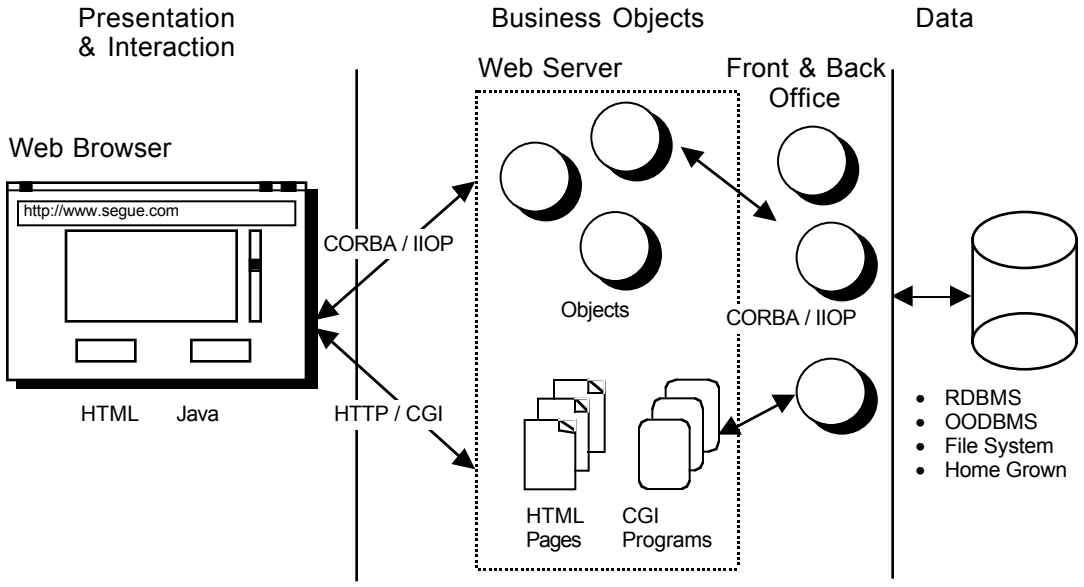
If the client program uses the dynamic invocation interface instead of the static approach, the compiled IDL must be written to the interface repository, and the DII access library is linked to the client instead of the stub generated by the IDL compiler. Using the DII, the client can obtain a method description and then create an argument list and request.

It is a common misconception that the ORB is a singular process, which would introduce a communication bottleneck. In fact, the functionality of an ORB is typically implemented in three distinct components: a library that is linked to each client, a library that is linked to each server, and an activation daemon that runs on each host where CORBA servers will run. The activation daemon is responsible for ensuring CORBA servers are launched properly and that clients can make connections to servers. After that, all communication occurs directly between the client and server processes.

CORBA in E-business Applications

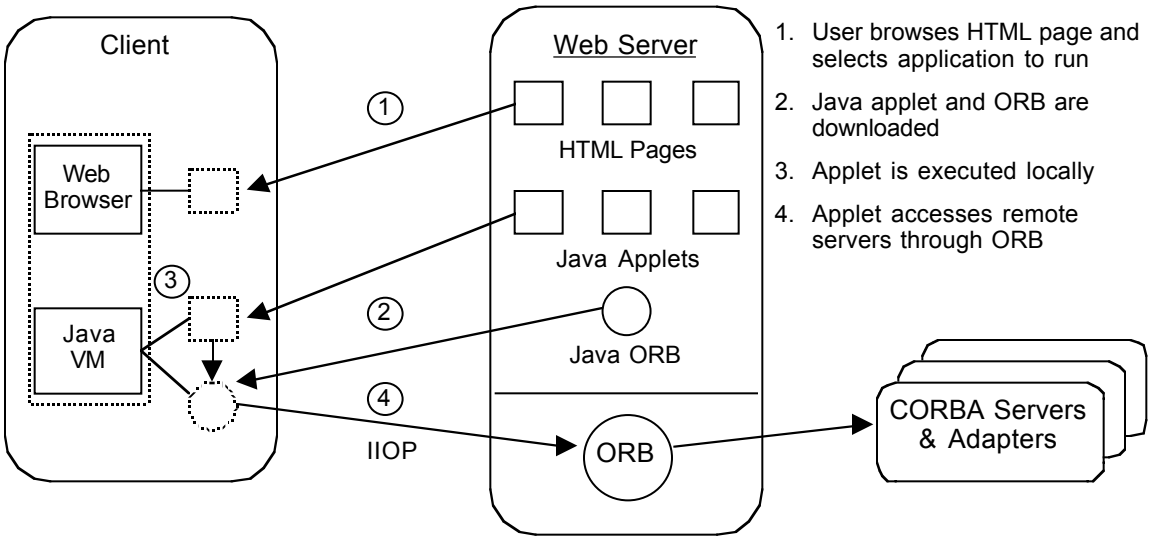
Electronic-business ("e-business") implies business transactions being conducted electronically over an internet-based infrastructure. E-business applications are typically implemented in multiple tiers. The first tier represents the presentation and interaction layer; for example, a web browser. The middle tier consists of the application logic, which can be constructed from multiple components, such as web and application servers. The back tier includes data repositories, such as relational or object-oriented databases. With the growing need to integrate multiple heterogeneous systems, CORBA is increasingly used as the platform for tying together these systems as distributed objects. Objects can be new application components, or existing applications that are encapsulated so they can be integrated into the environment.

Java has accelerated the proliferation of distributed objects by extending the three-tier architecture to the Internet and intranets. The introduction of web browsers has clearly made the Internet more usable for end users, and Java has proven to be very viable for adding local behavior in a platform independent way. The addition of CORBA provides an infrastructure for distributed object communication between Java and e-business applications developed in any language. IIOP implements a standard wire protocol, allowing CORBA to become the basis for distributed application development and deployment.



Three-tier e-business architecture.

The Internet paradigm involves pulling Java byte codes from a web server to a front-end deployment platform such as a PC or network computer. Java implementations of ORBs exist today, which means an ORB is downloaded from a web server just like any other Java class. IIOP ensures that a Java ORB from one vendor can communicate with ORBs from other vendors, regardless of implementation language.



Java object request broker (ORB).

In the Internet or intranet scenario, the CORBA process is exactly the same as described earlier. Combining Java and CORBA has several important benefits:

- The CORBA three-tier model combined with the platform independence of Java supports the pull paradigm for application distribution, greatly simplifying maintenance of the application install base. Furthermore, an application architecture based on CORBA is far more scaleable than the traditional two-tier client server model. Middle tier logic can be partitioned and dynamically distributed, thereby moving processing closer to data sources and supporting load balancing.
- Since CORBA facilitates language independence and location transparency, Java objects can communicate with other distributed objects. This provides an effective means of web-enabling existing applications without incurring the bottlenecks experienced with HTTP and CGI based approaches.
- Unlike CGI, Java clients can directly invoke server methods, allowing servers to maintain state between calls. In addition, clients can pass parameters of various data types, not just strings.
- Browsers can behave as live servers. Callbacks allow clients to be updated dynamically as information becomes available.
- The highly scaleable and interoperable nature of CORBA makes it ideal as a platform for basing higher level object frameworks, such as Enterprise Java Beans.