**CORBA**

The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing infrastructure being standardized by the Object Management Group ([OMG](http://www.omg.org/)). CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching. See the [OMG](https://www.dre.vanderbilt.edu/~schmidt/www.omg.org) Web site for more [overview](http://www.omg.org/corba/beginners.html) material on CORBA. See my [CORBA](https://www.dre.vanderbilt.edu/~schmidt/corba.html) page for additional information on CORBA, including our [tutorials](https://www.dre.vanderbilt.edu/~schmidt/tutorials-corba.html) and [research](https://www.dre.vanderbilt.edu/~schmidt/corba-research.html) on [high-performance](https://www.dre.vanderbilt.edu/~schmidt/corba-research-performance.html) and [real-time](https://www.dre.vanderbilt.edu/~schmidt/corba-research-realtime.html) ORBs. Results from our research on high-performance and real-time CORBA are freely available for [downloading](https://www.dre.vanderbilt.edu/~schmidt/TAO-obtain.html) in the [open-source](http://www.opensource.org/) [TAO](https://www.dre.vanderbilt.edu/~schmidt/TAO.html) ORB.

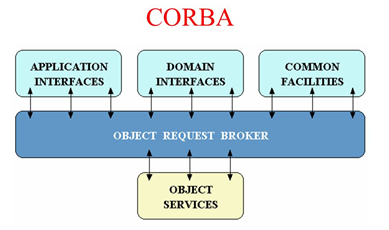
****

Figure 1. OMG Reference Model Architecture

* **Object Services** -- These are domain-independent interfaces that are used by many distributed object programs. For example, a service providing for the discovery of other available services is almost always necessary regardless of the application domain. Two examples of Object Services that fulfill this role are:
  + The Naming Service -- which allows clients to find objects based on names;
  + The Trading Service -- which allows clients to find objects based on their properties.

There are also Object Service specifications for lifecycle management, security, transactions, and event notification, as well as many others.

* **Common Facilities**-- Like Object Service interfaces, these interfaces are also horizontally-oriented, but unlike Object Services they are oriented towards end-user applications. An example of such a facility is the *Distributed Document Component Facility* (DDCF), a compound document Common Facility based on OpenDoc. DDCF allows for the presentation and interchange of objects based on a document model, for example, facilitating the linking of a spreadsheet object into a report document.
* **Domain Interfaces**-- These interfaces fill roles similar to Object Services and Common Facilities but are oriented towards specific application domains. For example, one of the first OMG RFPs issued for Domain Interfaces is for Product Data Management (PDM) Enablers for the manufacturing domain. Other OMG RFPs will soon be issued in the telecommunications, medical, and financial domains.
* **Application Interfaces**- These are interfaces developed specifically for a given application. Because they are application-specific, and because the OMG does not develop applications (only specifications), these interfaces are not standardized. However, if over time it appears that certain broadly useful services emerge out of a particular application domain, they might become candidates for future OMG standardization.

### CORBA ORB Architecture

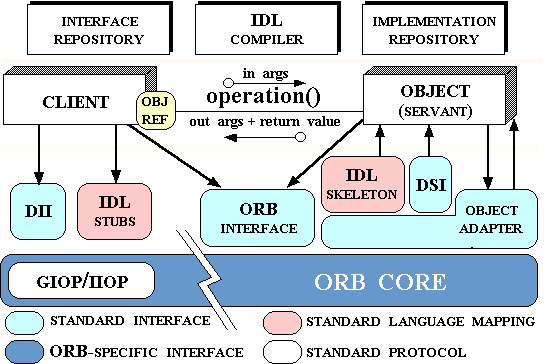


Figure 2. CORBA ORB Architecture

* **Object** -- This is a CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *Servant*.
* **Servant** -- This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.
* **Client** -- This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, i.e., obj->op(args). The remaining components in Figure 2 help to support this level of transparency.
* **Object Request Broker (ORB)** -- The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.
* **ORB Interface** -- An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.
* **CORBA IDL stubs and skeletons** -- CORBA IDL stubs and skeletons serve as the ``glue'' between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.
* **Dynamic Invocation Interface (DII)** -- This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred synchronous* (separate send and receive operations) and *oneway* (send-only) calls.
* **Dynamic Skeleton Interface (DSI)** -- This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.
* **Object Adapter**-- This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).
* **String Reverse using corba, idl and java implementation:**

1. Create the all **ReverseServer.java** , **ReverseClient.java** , **ReverseImpl.java** & **ReverseModule.idl** files.

2. Run the IDL-to-Java compiler idlj, on the IDL file to create stubs and skeletons. This step assumes that you have included the path to the java/bin directory in your path.

**idlj -fall ReverseModule.idl**

The idlj compiler generates a number of files.

3. Compile the **.java files**, including the stubs and skeletons (which are in the directory newly

created directory). This step assumes the java/bin directory is included in your path.

**javac \*.java ReverseModule/\*.java**

4. Start orbd. To start orbd from a UNIX command shell, enter:

**orbd -ORBInitialPort 1050&**

**5.** Start the server. To start the server from a UNIX command shell, enter:

**java ReverseServer -ORBInitialPort 1050& -ORBInitialHost localhost&**

6. Run the client application:

**java ReverseClient -ORBInitialPort 1050 -ORBInitialHost localhost**