

CORBA^{2.0}

Common Object Request Broker Architecture

July, 1997

Michael S. Pallos

X

Principal Consultant, InterNetworking Team
Internet Virtual Team

Table of Contents

I.	INTRODUCTION.....	1
II.	OBJECT MANAGEMENT GROUP	1
III.	CORBA OVERVIEW.....	1
IV.	THE ANATOMY OF A CORBA 2.0 ORB	2
	A. DYNAMIC INVOCATION INTERFACE (DII)	3
	B. CLIENT IDL STUBS.....	4
	C. ORB INTERFACE	4
	D. STATIC SKELETONS	4
	E. DYNAMIC SKELETON INVOCATION	4
	F. OBJECT ADAPTER	5
	G. INTERFACE REPOSITORY	5
	H. IMPLEMENTATION REPOSITORY	5
V.	INTERFACE DEFINITION LANGUAGE.....	5
VI.	THE OBJECT REQUEST BROKER.....	7
	A. STATIC AND DYNAMIC METHOD INVOCATION.....	8
	B. HIGH-LEVEL LANGUAGE BINDINGS	8
	C. SELF-DESCRIBING SYSTEM	8
	D. LOCAL / REMOTE TRANSPARENCY.....	8
	E. BUILT-IN SECURITY AND TRANSACTIONS	9
	F. POLYMORPHIC MESSAGING	9
VII.	CORBASERVICES	9
	A. LIFE CYCLE SERVICE	9
	B. PERSISTENCE SERVICE	10
	C. NAMING SERVICE	10
	D. EVENT SERVICE	10
	E. CONCURRENTLY CONTROL SERVICE	10
	F. TRANSACTION SERVICE.....	10
	G. RELATIONSHIP SERVICE	10
	H. EXTERNALIZATION SERVICE	11
	I. QUERY SERVICE	11
	J. LICENSING SERVICE.....	11
	K. PROPERTIES SERVICE.....	11
	L. TIME SERVICE.....	11

M. SECURITY SERVICE.....	11
N. TRADER SERVICE	12
O. COLLECTION SERVICE	12
P. STARTUP SERVICE	12
VIII. EXAMPLE ORBS	12
A. CLIENT- AND IMPLEMENTATION-RESIDENT ORB	12
B. SERVER-BASED ORB.....	12
C. SYSTEM-BASED ORB.....	13
D. LIBRARY-BASED ORB.....	13
IX. CORBA VERSE DCOM	14
X. THE CURRENT SHORT COMINGS	15
A. SLOW TRANSPORT MECHANISM.....	15
B. PORTABILITY OF OBJECT IMPLEMENTATION CODE.....	15
XI. REFERENCES	15

I. Introduction

AS CTG continues to excel in the computer technologies industry, it is imperative that we continue learning and developing new skills. Continuing education is more important now more than ever before since technology is changing so rapidly. Within the world of Internet and client/server technologies, one of the most important and ambitious middleware projects ever undertaken by the industry is the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). It appears as though the next generation of middleware is CORBA. The CORBA object bus defines the shape of objects that live within the bus and their interoperability. The goal is to create an open playing field for all components.

CORBA is positioned to subsume every other form of existing client/server middleware. More specifically, CORBA provides a solid foundation for a component-based future, while using objects as a unifying metaphor for bringing existing applications to the bus.

II. Object Management Group

The Object Management Group is an international organization supported by over 650 members. The members include all of the major vendors of systems and software from around the world. The notable exception is Microsoft which has its own competing object broker called the Distributed Component Object Model (DCOM). One third of OMGs members come from outside the United States, most from Europe, but about 5 percent of the total from other countries, chiefly in the Asian Pacific Rim but also including Australia, Africa, and South America. Founded in 1989, the OMG promotes the use of object-oriented software development.

Unlike some other consortium, the OMG does not produce software, only specifications. The specifications are freely available for any company (OMG member or not) to implement; neither explicit permission nor fee is required. OMG expects that implementations will come from many companies.

III. CORBA Overview

The Common Object Request Broker Architecture (CORBA) is design to integrate a wide variety of object systems. CORBA objects are blobs of

intelligence that can live anywhere on a network and interact with each other without having any prior knowledge. They are packaged as binary components that remote clients can access via method invocations. Figure 1.0 shows a request being sent by a client to an object implementation. The client is the entity requesting a service and the object implementation is the entity responsible for fulfilling the requested service. The ORB is the entity responsible for all of the mechanics required to locate the object implementation to receive the request, and communicate the results back to the requesting client. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is not reflected in the object's interface.

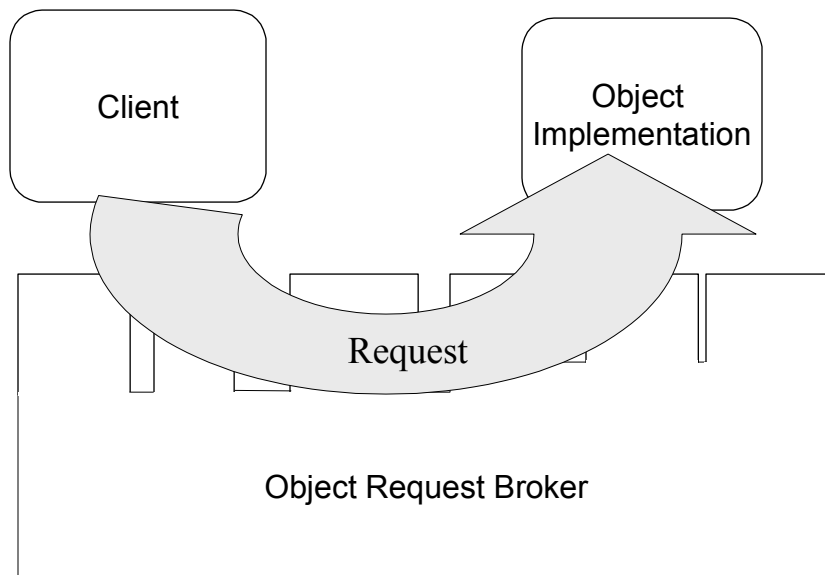


Figure 1.0 - A client request being sent through the Object Request Broker to the Object Implementation.

IV. The Anatomy of a CORBA 2.0 ORB

A CORBA 2.0 Object Request Broker (ORB) is the middleware that establishes the relationship between objects. Figure 2.0 shows the structure of an individual ORB. The grayed boxes reflect the ORB interfaces, and the arrows represent the flow of execution.

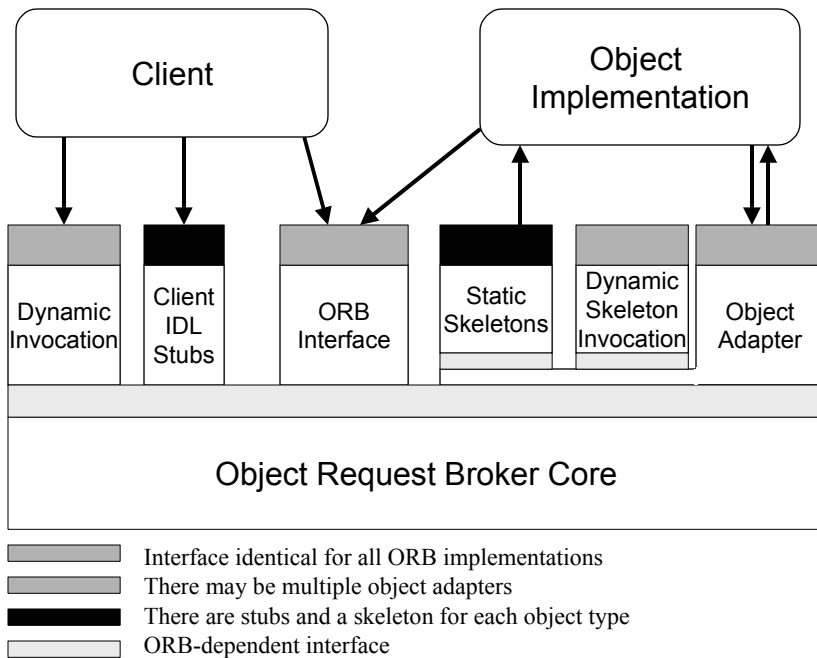


Figure 2.0 - The structure of Object Request Broker Interfaces.

CORBA provides both static and dynamic interfaces to its services. This is a result of the OMG receiving two requests with strong business cases to its original Request For Proposal (RFP). One was from Sun and Hewlett-Packard based on static APIs and the other from Digital and HyperDesk based on Dynamic APIs. The combined approach gave birth to the "Common" in CORBA, resulting in architecture supporting both static and dynamic APIs.

To make a request the Client uses the OMG Interface Definition Language (IDL) stub or the Dynamic Invocation interface. The client can also directly interact with the ORB for some methods.

The Object Implementation receives a request as input either through the dynamic skeleton or the OMG IDL generated skeleton. The Object Implementation may call the Object Adapter and the ORB while processing a request.

A. Dynamic Invocation Interface (DII)

This interface allows the dynamic construction of object invocations, as opposed to the client invoking a specific method on a particular object. The client may supply the object to be invoked, the method

to be executed, and the parameters required for invocation. The nature of the dynamic invocation interface usually varies for the implementation of the different development languages.

B. Client IDL Stubs

A programming interface is required to the stubs for the mapping of non-object oriented languages. Both the client and the skeleton stubs are generated via the IDL compiler. A client must have an IDL stub for each service/interface it requests of the object implementation (server). If more than one ORB is available, then different stubs may be required for the different ORBs. Object-oriented programming languages like C++ and SmallTalk do not require stub interfaces.

C. ORB Interface

The ORB interface is the interface that is the same for all ORBs and goes directly to the ORB. These are the common services that can be utilized by both the client and object implementation (server). Most of the other functionality of the ORB is obtained via the object adapter, skeleton, stubs, and dynamic invocation.

D. Static Skeletons

The static skeletons contain interface to the methods that implement each type of object. The IDL compiler creates the skeleton stubs. Skeletons may exist without a corresponding client stub; these would be the object implementations available to dynamic invocation.

E. Dynamic Skeleton Invocation

This interface was introduced in CORBA 2.0 and allows dynamic handling of object invocations. That is, a run-time binding mechanics for skeletons responsible for handling incoming method calls for components that do not have IDL-based skeletons. Unlike an IDL-generated skeleton, the dynamic skeleton reviews the parameter list of the incoming request, and passes the request on of the target object. Dynamic skeletons are extremely useful for implementing generic bridges between ORBs. The Dynamic Skeleton Invocation can receive either dynamic client invocation or static.

F. Object Adapter

The primary ways in which services accessed by an OBR are implemented are via the object adapter. The object adapter is responsible for providing the run-time environment for instantiating server objects, passing them requests, assigning object references, and registering the class that the object supports and their run-time instances with the implementation repository.

G. Interface Repository

The interface repository is the data-housing component containing the IDL information in machine-readable form that is available at run-time - (Similar to a dynamic metadata repository of ORBs). This pervasive use of metadata allows every component that is available on the ORB to contain self-describing interfaces. The ORB itself is a self-describing bus.

H. Implementation Repository

The implementation repository is the data-housing component containing information about the classes server supports, the objects that are instantiated, and their IDs. The implementation repository is also a common place to store additional information associated with implementations of ORB objects, such as; trace/debugging information, audit trails, resource allocations, administrative control, security, and other administrative data.

V. Interface Definition Language

The CORBA Interface Definition Language (IDL) is purely declarative, providing no implementation details. The interface is defined in the IDL and/or in the Interface Repository. The definition is used to generate the client stubs and the object implementation skeletons (server stubs). Figure 3.0 represents the interface and implementation information made available to clients and object implementations.

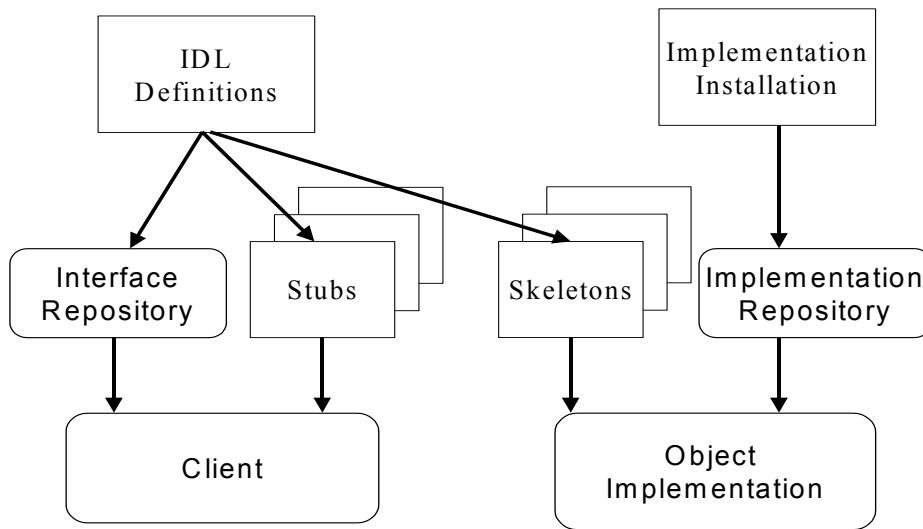
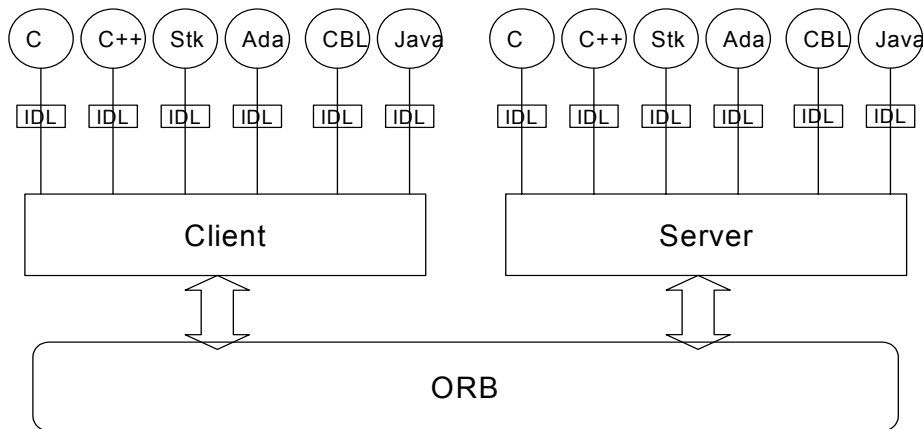


Figure 3.0 - Interface and Implementation Repositories.

CORBA uses IDL contracts to specify a component's boundaries and its contractual interfaces with potential clients. The IDL can be used to define APIs concisely, and also cover important issues such as error handling. As reflected in Figure 4.0, C, C++, Ada, Smalltalk, COBOL, Java and Objective C support IDL specified methods providing CORBA bindings.



Stk = Smalltalk, CBL = COBOL

Figure 4.0 - CORBA IDL language bindings.

VI. The Object Request Broker

The Object Request Broker (ORB) is the object bus. It allows objects to transparently communicate with each other either locally or remotely. The client is not aware of the mechanism required to locate, communicate, activate or store the object implementation (server) objects.

The CORBA ORB allows objects to discover each other at run time, while supplying a wide heterogeneity of distributed middleware services. The ORB is more sophisticated than the alternative forms of client/server middleware including; Remote Procedure Calls (RPCs), Message-Oriented Middleware (MOM), database stored procedures/triggers, and peer-to-peer services. From an academic perspective, CORBA is the best middleware ever defined to date. However, since OMG only develops the CORBA standard, CORBA is only as good as the products developed and implemented by the vendors.

Figure 5.0 represents the ORB. Provided below is a list of the benefits of CORBA ORBs extracted from *Client/Server Programming with JAVA and CORBA*.

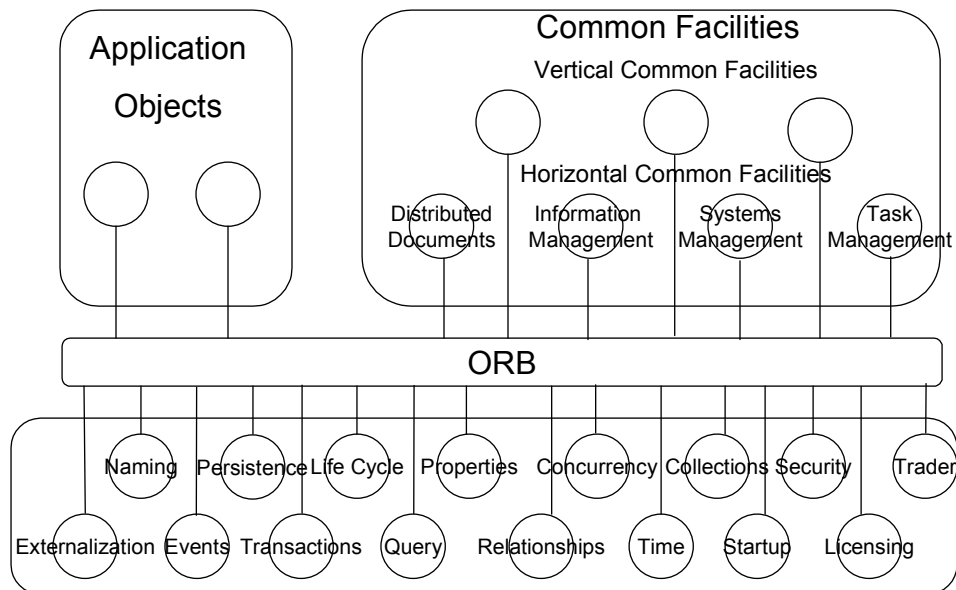


Figure 5.0 - The Object Management Architecture.

A. Static and dynamic method invocation

A CORBA ORB lets you either statically define your method invocation at compile time, or it lets you dynamically discover them at run time. So you either get strong (compile time) type checking or you get flexibility associated with late (run time) binding. Most forms of middleware only support static binding.

B. High-level language bindings

A CORBA ORB lets you invoke methods on server objects using your high-level language of choice - currently, C, C++, Ada, and Smalltalk. It doesn't matter what language server objects are written in. CORBA separates interface from implementation and provides language-neutral data types that make it possible to call objects across language and operating system boundaries. In contrast, other types of middleware typically provide low-level, language-specific, API libraries. And they don't separate implementation from specification - the API is tightly bound to the implementation, which makes it very sensitive to change.

C. Self-describing system

CORBA provides run-time metadata for describing every server interface known to the system. Every CORBA ORB must support an Interface Repository that contains real-time information describing the functions a server provides and their parameters. The clients use metadata to discover how to invoke services at run time. It also helps tools generate code 'on-the-fly'. The metadata is generated automatically either by an IDL-language precompiler or by compilers that know how to generate IDL directly for an OO language. For example, the MetaWare C++ compiler generates IDL directly from C++ class definitions (it may also write the definition in the interface repository). To the best of our knowledge, no other form of client/server middleware provides this type of run-time metadata and language-independent definitions of all its services.

D. Local / remote transparency

An ORB can run in standalone mode on a laptop, or it can be interconnected to every other ORB in the universe (using CORBA 2.0

inter-ORB servers). An ORB can broker interobject calls within a single process, multiple processes running within the same machine, or multiple processes running across networks and operating systems. This is all done in a manner that's transparent to you objects. Note that the ORB can broker among fine-grained objects - like C++ classes - as well as more coarse-grained objects. In general, a CORBA client/server programmer does not have to be concerned with transports, server locations, object activation, byte ordering across dissimilar platforms, or target operating systems - CORBA makes it all transparent.

E. Built-in security and transactions

The ORB includes context information in its messages to handle security and transactions across machine and ORB boundaries.

F. Polymorphic messaging

In contrast to other forms of middleware, an ORB does not simply invoke a remote function - it invokes a function on a target object. This means that the same function call will have different effects, depending on the object that receives it. For example, a *configure_yourself* method invocation behaves differently when applied to a database object versus a printer object.

VII. CORBA services

CORBA services are collections of system-level services packaged with IDL-specified interfaces. The object services augment and complement the functionality of the ORB. To date, OMG has published standards for sixteen services:

A. Life Cycle Service

Life Cycle Service defines operations for creating, copying, moving, and deleting components on the bus.

B. Persistence Service

Persistence Service provides a single interface for storing components persistently on a variety of storage servers - including Object Databases (ODBMSs), Relational Databases (RDBMSs), and simple files.

C. Naming Service

Naming Service allows components on the bus to locate other components by name; it also supports federated naming contexts. The service allows objects to be bound to existing network directories or naming contexts - including ISO's X.500, OSF's DCE, and SUN's NIS+.

D. Event Service

Event Service allows components on the bus to dynamically registers or unregistered their interest in specific events. The service defines a well-known object called an *event channel* that collects and distributes events among components that know nothing of each other.

E. Concurrently Control Service

Concurrently Control Service provides a lock manager that can obtain locks on behalf of either transactions or threads.

F. Transaction Service

Transaction Service provides two-phased commits coordination among recoverable components using either flat or nested transactions.

G. Relationship Service

Relationship Service provides a way to create dynamic associations (or links) between components that know nothing of each other. It also provides mechanism for traversing the links that group these components. You can use the service to enforce referential integrity constraints, track containment relationships, and for any type of linkage among components.

H. Externalization Service

Externalization Service provides a standard way for getting data into and out of a component using a stream-like mechanism.

I. Query Service

Query Service provides query operations for objects. It's a superset of SQL based on the upcoming SQL3 specification and the Object Database Management Group's (ODMG) Object Query Language (OQL).

J. Licensing Service

Licensing Service provides operations for metering the use of components to ensure fair compensations for their use. The service supports any model of usage control at any point in a component's life cycle. It supports charging per session, per node, per instance creation, and per site.

K. Properties Service

Properties Service provides operations to let you associate named values (or properties) with any component. Using this service, you can dynamically associate properties with a component's state - for example, a title or date.

L. Time Service

Time Service provides interfaces for synchronizing time in a distributed object environment. It also provides operations for defining and managing time-triggered events.

M. Security Service

Security Service provides a complete framework for distributed object security. It supports authentication, access control lists, confidentiality, and non-repudiation. It also manages the delegation of credentials between objects.

N. Trader Service

Trader Service provides a "Yellow Pages" for objects; it allows objects to publicize their services and bid for jobs.

O. Collection Service

Collection Service provides CORBA interfaces to generically create and manipulate the most common collections.

P. Startup Service

Startup Service enables requests to automatically start up when an ORB is invoked.

VIII. Example ORBs

A wide variety of possible ORB implementations exist for the Common ORB Architecture. This section obtained from OMG's, *The Common Object Request Broker: Architecture and Specification, Revision 2.0* reflects some of the different options.

A. Client- and Implementation-resident ORB

If there is a suitable communication mechanism present, an ORB can be implemented in routines resident in the clients and implementations. The stubs in the client either use a location-transparent IPC mechanism or directly access a location service to establish communication with the implementations. Code linked with the implementation is responsible for setting up appropriate databases for use by clients.

B. Server-based ORB

To centralize the management of the ORB, all clients and implementations can communicate with one or more servers whose job it is to route requests from clients to implementations. The ORB could be a normal program as far as the underlying operating system is

concerned, and normal IPC could be used to communicate with the ORB.

C. System-based ORB

To enhance security, robustness, and performance, the ORB could be provided as a basic service of the underlying operating system. Object references could be made unforgettable, reducing the expense of authentication on each request. Because the operating system could know the location and structure of clients and implementations, it would be possible for a variety of optimizations to be implemented, for example, avoiding marshalling when both are on the same machine.

D. Library-based ORB

For objects that are lightweight and whose implementations can be shared, the implementation might actually be in a library. In this case, the stubs could be the actual methods ORB.

IX. CORBA verse DCOM

CORBA appears to currently be the industry standard for distributed objects. However, the other standard is Microsoft's Distributed Component Object Model (DCOM). What makes DCOM so important is Microsoft. Microsoft ships DCOM with NT 4.0, and has announced plans to also ship DCOM with every copy of Windows 97. DCOM is also the foundation from Microsoft's Internet and component strategy. The bottom line - DCOM is the ORB CORBA must beat. The art of guesstimation as to the winner of this battle is beyond the scope of this article. However, Robert Orfali and Dan Harkey in *Client/Server Programming with JAVA and CORBA* have compiled the following report card.

Feature	CORBA/ IIOP	DCOM	RMI	HTTP/CGI	Sockets
Abstraction level	★★★★	★★★★	★★★★	★★	★
Seamless Java integration	★★★★	★★★	★★★★	★★	★★
OS platform support	★★★★	★★	★★★★	★★★★	★★★★
All-Java implementation	★★★★	★	★★★★	★★★★	★★★★
Typed parameter support	★★★★	★★★★	★★★★	★	★
Ease of configuration	★★★	☆	★★★	★★★	★★★
Distributed method invocations	★★★★	★★★	★★★	☆	☆
State across invocations	★★★★	★★★	★★★	☆	★★★
Dynamic discovery and metadata support	★★★★	★★★	☆	☆	☆
Dynamic invocation	★★★★	★★★★	★	☆	☆
Performance (remote Pings)	★★★★ 3.3 msecs	★★★ 3.9 msecs	★★★ 5.5 msecs (extrapo- lated)	☆ 603.8 msecs	★★★★ 2.0 msecs
Wire-level security	★★★★	★★★★	★★★	★★★	★★★
Wire-level transactions	★★★★	★★★	☆	☆	☆
Persistent objects	★★★★	★	☆	☆	☆
URL-based naming	★★★	★	★★	★★★★	★★★
Multilingual object invocations	★★★★	★★★	☆	★★★	★★★★
Language-neutral wire protocol	★★★★	★★★★	☆	★★★★	☆
Intergalactic scaling	★★★★	★	★	★★	★★★★
Open standard	★★★★	★★	★★	★★★★	★★★★

Source: *Client/Server Programming with JAVA and CORBA* by Robert Orfali and Dan Harkey

X. The Current Short Comings

A. Slow Transport Mechanism

The current generation of ORBs are slow and are not suitable for mission critical client/server applications. The products are still in the bleeding edge of the technology and for large enterprise applications do not contain the load-balancing, concurrency control and replication abilities of the commercial TP Monitors available in today's market place.

B. Portability of Object Implementation Code

The current specification of the object implementation (server) code does not clearly define all of the interfaces required to write portable server code. However, the specifications do exist for the client code. OMG is currently designing the solution.

XI. References

Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.0, <http://www.omg.org>, 1996

Orfali, Robert. Client/Server Programming with JAVA and CORBA, John Wiley & Sons, Inc. New York, USA, 1997

Orfali, Robert. Essential Client/Server Survival Guide, Second Edition, John Wiley & Sons, Inc. New York, USA, 1996

Siegel, Jon. CORBA Fundamentals and Programming, John Wiley & Sons, Inc. New York, USA, 1996