A

Seminar report

on

CORBA

Submitted in partial fulfillment of the requirement for the award of degree of Bachelor of Technology in Computer Science

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Preface

I have made this report file on the topic **CORBA,** I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the prepration of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

What is CORBA?

CORBA (Common Object Request Broker Architecture) is a distributed Object-oriented client/server platform.
It includes:

• An object-oriented Remote Procedure Call (RPC) mechanism
• object services (such as the Naming or Trading Service)
• Language mappings for different programming languages
• Interoperability protocols
• Programming guidelines and patterns

CORBA replaces ad-hoc special-purpose mechanisms (such as socket Communication) with an open, standardized, scalable, and portable Platform.

The Object Management Group (OMG)

The OMG was formed in 1989 to create specifications for open Distributed computing. Its mission is to "... Establish industry guidelines and object management Specifications to provide a common framework for distributed Application development."

The OMG is the world's largest software consortium with more than 800 member organizations.

Specifications published by the OMG are free of charge. Vendors of CORBA technologies do not pay a royalty to the OMG.

Specifications are developed by consensus of interested submitters.

CORBA History

OMG (Object Management Group)

- Established in 1989 with 8 members
- Whose charter is to "provide a common architectural framework for object-oriented applications based on widely available interface specifications?"
- Object Management Architecture is a set of standards deliver the common architectural framework on which applications are built.
- CORBA's role in OMA is to implement the ORB functions.

CORBA 1.0

- Was introduced and adopted in December 1990.
- It was followed in early 1991 by CORBA 1.1, which defined the Interface Definition Language (IDL) as well as API for applications to communicate with an ORB.

CORBA 2.0 and IIOP

- CORBA 1.x was an important first step is providing distributed object interoperability, but wasn't a complete specification.
- Although it provided standards for IDL and for accessing an ORB through an application, its chief limitation was that it did not specify a standard protocol through which ORBs could communicate with each other.
- As a result, a CORBA ORB from one vendor could not communicate with an ORB from another vendor, a restriction that severely limited interoperability among distributed objects.
- CORBA 2.0 is adopted in December 1994.
- The primary accomplishment was to define a standard protocol by which ORB from various CORBA vendors could communicate.
- This protocol, known as the IIOP is required to be implemented by all vendors who want to call their products CORBA 2.0 compliant.
- IIOP ensures true interoperability among products from numerous vendors, thus enabling CORBA applications to be more vendor-independent.

OMG Common Object Request Broker Architecture (CORBA)

The Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA®) middleware standard enables software applications to invoke operations on distributed objects without concern for object location, programming language, operating system platform, communication protocols, interconnections or hardware.

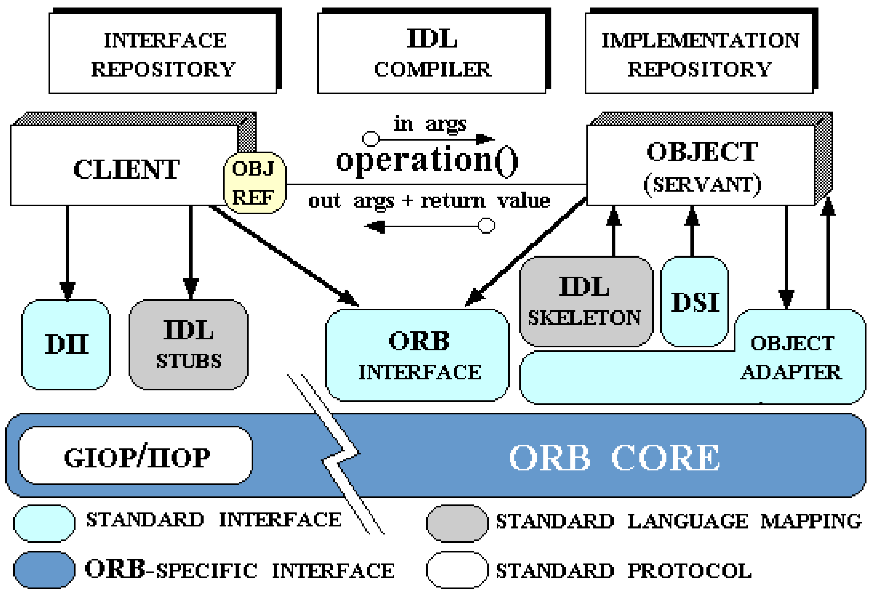
CORBA remains the most successful open standard in supporting distributed heterogeneous mission critical systems that require exceptional levels of performance and QoS. PrismTech's **Open Fusion** provides the most comprehensive range of CORBA middleware products available from any vendor.

CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. CORBA then specifies a “mapping” from IDL to a specific implementation language such as C++ or Java. Standard mappings exist for Ada, C, C++, Lisp, Smalltalk, Java, COBOL, PL/I and Python. There are also non-standard mappings for Perl, Visual Basic, Ruby, Erlang, Tcl and even VHDL implemented by object request brokers (ORBs) written for those languages.

A language mapping requires the developer to create some IDL code that represents the interfaces to his objects. Typically, a CORBA implementation comes with a tool called an IDL compiler which converts the developer's IDL code into some language-specific generated code. A traditional compiler then compiles the generated code to create the linkable-object files for the application. the figure below illustrates how the generated code is used within the CORBA infrastructure.

The CORBA specification dictates that there shall be an object request broker (ORB) through which the application interacts with other objects. In practice, the application simply initializes the ORB, and accesses an internal Object Adapter which maintains such issues as reference counting, object (& reference) instantiation policies, object lifetime policies, etc.

The Object Adapter is used to register instances of the generated code classes. Generated Code Classes are the result of compiling the user IDL code which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce the CORBA semantics and provide a clean user processes for interfacing with the CORBA infrastructure.



The key components of a CORBA ORB are as follows:

- 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 1 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).

A First CORBA Application

This section introduces the JServer CORBA application development process. It tells you how to write a simple but useful program that runs on a client system, connects to Oracle using IIOP, and invokes a method on a CORBA server object that is activated and runs inside an Oracle8i Java VM.

This section addresses only the purely mechanical aspects of the development process. Application developers know that for large-scale applications the design is a crucially important step. See "For More Information" for references to documents on CORBA design.

The CORBA application development process has seven phases:

1. Design and write the object interfaces.
2. Generate stubs and skeletons, and other required support classes.
3. Write the server object implementations.
4. Use the client-side Java compiler to compile both the Java code that you have written, and the Java classes that were generated by the IDL compiler. Generate a JAR file to contain the classes and any other resource files that are needed.
5. Publish a name for the directly-accessible objects with the CosNaming service, so you can access them from the client program.
6. Write the client side of the application. This is the code that will run outside of the Oracle8i data server, on a workstation or PC.
7. Compile the client code using the JDK Java compiler.
8. Load the compiled classes into the Oracle8i database, using the `loadjava` tool and specifying the JAR file as its argument. Make sure to include all generated classes, such as stubs and skeletons. (Stubs are required in the server when the server object acts as a client to another CORBA object.)

Some advantages:

• CORBA supports many existing languages (alone/mixed).
• CORBA supports distribution and Object Orientation.
• CORBA is an industry standard ; it creates competition among vendors and ensures quality implementations.
• CORBA provides out-of-the-box multi-vendor inter- operability and portability.
• CORBA is backed by over 700 companies : hardware, software, cable and phone companies, banks, etc.
• CORBA offers many services called CORBAServices.
• CORBA is well-suited for request/response applications over lower-speed networks(eg: Ethernet and Token Ring).
• Many application domains (such as avionics, multimedia, and telecommunications) require real-time guarantees from the underlying networks, operating systems, and middleware components to achieve QoS requirements and the applications in these domains must be flexible and reusable.These motivate the use of middleware like CORBA. But, the performance of CORBA implementations is not yet suited for hard real-time systems like avionics and constrained latency systems like teleconferencing.
• Communication software and distributed services for next - generation applications must be reliable, efficient, flexible, and reusable. These requirements motivate the use of CORBA.
• Synchronous and quasi-synchronous communication.

Some disadvantages:

• CORBA is still growing, and not fully mature.

• While CORBA does support a deferred synchronous request/response, it does not directly support distributed requests with callback driven response,ie. to perform an operation on a distributed object, associate a callback with the response, continue with other processing.When the server responds, the associated callback is automatically executed within the original callers application.

• The standard lacks many features required for putting large scale applications. Though specific vendor implemen- tations provide some of these features, they are not part of the standard today.

• The security aspects have not been sufficiently addressed in CORBA 2.0 These have been the major obstacles to serious and mission-critical applications development, esp. those who still employ legacy systems.

• Conventional implementations of CORBA are very good for slow networks like ethernet, but incur considerable over- head when used for performance-sensitive applications over high-speed networks.

Some major projects using CORBA

- Distributed Object Management Integration System(DOMIS)
- Dialogos' ORB based applications using ICL's DAIS ORB.
- University of Minnesota's DAMSEL Project (Dynamic Mul- timedia specification Language) uses CORBA and Java
- The Sunrise project and the TeleMed subproject
- Information Sharing System (ISS)
- A Framework for Distributed Digital Object Services by Robert Kahn and Robert Wilensky
- Distributed Systems Technology Centre (DSTC), Australia
- The ANSA project (Corba/WWW integration)
- GTE Labs' DOC Distributed Object Computing Group
- The Larch/CORBA project
- TCL/Orbix Integration
- The Web Broker
- The Motorola Iridium project Tools for use with CORBA:
- NetLinks Technology's ORBitize IDL builder/browser development tool
- Black & White Software's suite of development tools
- SNiFF+ development tool
- IDE's Software through Pictures OOA/OOD CASE tool
- METEOR multiparadigm Workflow management system
- ObjecTime's ObjecTime Toolset and CORBA features
- ProtoSoft's Paradigm Plus OOA/OOD CASE tool
- Rational's Rose OOA/OOD CASE tool
- I-Kinetics' Database Component product
- Sandia's IDLdoc documentation tools

TINA-ACE CASE tool

Conclusion

The main conclusion is that CORBA can be used to implement group communication services and thereby achieve interoperability in a heterogeneous computing environment.

However, there is a substantial performance cost. As a result, current CORBA technology is not suitable for implementing high performance group communication services.