Introduction
CHAPTER 1

INTRODUCTION

1.1 Introduction

Distributed Object Models allow clients to invoke methods on distributed objects. Distributed Object Systems facilitate software reuse and hence improve the productivity. Some of the commonly used request brokers in distributed systems are Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI) and Distributed Component Object Model (DCOM). These brokers bring about heterogeneity and interoperability in the distributed systems. They also shield the application developers from low-level, tedious and error prone platform details. They provide platform, language, network, hardware, protocol and object location transparency by including an abstraction layer between the application programs and the networking protocols.

In a distributed object model, client is a distributed object that performs application tasks by obtaining object references to remote server objects and invoking methods on them as shown in figure 1.1. Server object implementation is an instance of a class, associated with a public remote interface that implements the services provided by the object. The client/server roles can be exchanged during the run time. The client/server objects communicate by passing messages. For sending/receiving messages, the objects need not know the communication details. For this purpose, an abstraction layer called the Object Request Broker (ORB), which is an integral part of the Distributed Object Model is used. It is responsible for handling the details of communication like marshalling the client's request, locating the target object, transferring and receiving the message request, demultiplexing, demarshalling and dispatching the request, performing operation upcall and returning the result. The ORB core uses the operating system services to provide communication and a wire protocol for message interchange between the distributed objects. The wire protocol can be Java Remote Method
Invocation Protocol (JRMP) in RMI, the General Inter-ORB Protocol (GIOP) in CORBA or Object Remote Procedure Call (ORPC) in DCOM.

![Diagram of Remote Method Invocation]

In Java RMI [1], the interfaces are specified and implemented in Java, but it utilizes a strict separation of interfaces and implementation. The three independent layers that constitute the RMI system are the stub/skeleton layer, the remote reference layer and the transport layer. Some of the features offered by RMI include passing objects by value, distributed garbage collection, URL based object naming, dynamic class and stub downloading. But, RMI supports only two-way static synchronous method invocation. Further, it lacks support for interface and implementation repository, dynamic method invocations and one-way operations. Ann Wollrath [2] uses pickling to transmit arguments and return values. Dynamic stub loading is used to make the correct stub for the remote object available to client using java's mechanism for downloading code. It avoids the code mismatch by loading the exact stub code at runtime to handle remote method invocations on remote object. Since RMI uses JRMP for wire communication, it lacks interoperability with other languages.

In Distributed Component Model (DCOM) [3], the client makes an interface call to the proxy, which talks to Remote Procedure Call (RPC). RPC uses the underlying network protocol Transport Control Protocol/Internet Protocol (TCP/IP) to cross machines. On the server machine, RPC talks to the stub and the stub makes an interface call to the server. RPC uses
Universally Unique Identifiers (UUID) of the COM Interface Definition Language (IDL) to ensure that the RPCs are made to the correct server object. Microsoft IDL (MIDL) compiler is used to translate IDL file to code in a particular programming language. It thus produces the remote procedure stubs in client and server side.

Java RMI run over Internet Inter-Orb Protocol (RMI-IIOP) [4] delivers Common Object Request Broker Architecture (CORBA) distributed computing capabilities to the Java 2 platform. Like CORBA, RMI over IIOP is based on open standards and uses IIOP as its communication protocol. IIOP facilitates legacy application and platform integration. Like java RMI, RMI over IIOP facilitates application development in Java programming language. Further, it does not require a separate Interface Definition Language (IDL) and facilitates passing of serializable Java object (Objects By Value) between application components. In RMI over IIOP, objects can be passed both by reference and by value over IIOP. It provides transparent interoperability between RMI and CORBA distributed object models.

Java RMI is language dependent and DCOM is platform dependent. But CORBA is transparent to platform and programming language. CORBA [5] is based on the Object Management Architecture and the Core Object Model. The CORBA specification includes the ORB, object services (CORBA services) common facilities (CORBA facilities), domain facilities and application objects. CORBA utilizes strict separation of the interface and implementation by specifying the interfaces in a proprietary IDL called OMG IDL. Implementation is in programming language for which mapping from IDL exists. The architecture of a typical CORBA compliant ORB includes static stubs and skeletons, Dynamic Invocation Interface, Dynamic Skeleton Interface, Object Adapter and the ORB core. For inter-ORB communication, CORBA standard specifies the General Inter-ORB Protocol (GIOP), which is a language independent wire protocol. The Internet Inter-ORB Protocol (IIOP) is a specialization of the GIOP to the TCP/IP transport protocol.
Figure 1.2 shows a request being sent by a client to an object implementation. The Client is the entity that wishes to perform an operation on the object and the Object Implementation is the code and data that actually implements the object. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object location and programming language.

The client makes a request using Dynamic Invocation Interface, which is independent of the target object's interface or an OMG IDL stub. Definitions of the interfaces to objects can be done in the following ways:

1) Interfaces can be defined statically in an Interface Definition Language (IDL), called the OMG IDL. This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations.

2) Interfaces can be added to an Interface Repository service. This service represents the components of an interface as objects, permitting run-time access to these components.

Thus CORBA provides a flexible communication and activation substrate for distributed heterogeneous object-oriented computing environments. The following are the strengths of CORBA:

1) **Heterogeneity**: The use of OMG IDL to define object interfaces allows these interfaces to be used from a variety of programming languages and computing platforms.
2) **Object Model:** The Object Model and Reference Model provided by the Object Management Architecture (OMA) define the rules for interaction between CORBA objects such that the interactions are independent of underlying network protocols. Thus CORBA-based applications are abstracted away from the networking details and can be used in a variety of environments.

3) **Legacy integration:** The CORBA specification is flexible enough to allow ORBs to incorporate and integrate existing protocols and applications, such as DCE or Microsoft COM, rather than replace them.

4) **Object-oriented approach:** CORBA and its applications are designed using object-oriented (OO) software development principles. Since object interfaces are defined in terms of OMG IDL, applications can be viewed as interacting reusable components.

These capabilities and flexibilities offered by CORBA, allow the developers to concentrate more on higher-level problems, than on the integrating issues in distributed heterogeneous environments. Interoperability facility in CORBA makes it suitable to manage communication issues in astronomical data providers [6], electronic brokers [7], diverse applications [8] and legacy systems [9] in a transparent way.

This paragraph lists some important features of the commercial implementations of CORBA. The core of Horus CORBA [10] has a C++ library for image processing. The library contains classes for images, image sequences, histograms and BSplines. Graphical user interfaces implemented in java, provide access to the C++ functionality in a more interactive way. It has java mdi application and video application support. CorbaScript provides access to all CORBA-based functionality. Orbix ORB [11][12] is a widely deployed enterprise CORBA solution for telecommunications, financial services, manufacturing and government sectors. It is a CORBA 2.5 compliant ORB with extensible configuration. It has event service with Asynchronous Method Invocation (AMI) and Secure Socket Layer (SSL) security. It also supports load balancing. VisiBroker by Borland [13] uses CORBA to communicate with non-Java objects. It has both the VisiBroker for Java and
the VisiBroker for C++ ORBs. It is CORBA 2.6 compliant and supports security and web connectivity features. It supports naming and event services. Orbacus [14] is suitable for mission critical systems in the telecommunications, finance, government, defense, aerospace and transportation industries. It is designed for rapid development, deployment and support in your choice of C++ or Java. Its small footprint allows it to be easily embedded into memory constrained applications. ORBit [15] is a CORBA 2.4 - compliant ORB with C, Perl, C++, Lisp, Pascal, Python, Ruby and TCL bindings. It is focused on performance, low resource usage and security. The core ORB is written in C and runs under Linux, UNIX and Windows. ORBit is developed and released as open source software. The CORBA comparison project by MLC systems [16] compares the performance of OmniORB, Orbacus and Orbix ORBs and has identified that OmniORB performs better.

MT DORB [17] is free and opensource CORBA 2.3 compliant ORB for Delphi and Kylix. It is a multi-threaded ORB with IDL to Object Pascal mapping. It has support for multiple protocols and secure communication and authentication using SSL. VBORB [18] is written in Visual BASIC. Using it CORBA clients and servers can be written in Visual BASIC. Mico/E [19] is CORBA 2.3 compliant and uses its own IDL/Eiffel mapping. The CORBA services supported by it include Naming Service, Event Service, Trader Service, and Persistent State Service. It also supports CORBA Component Model (CCM) features. LuaORB [20] is a language binding for the interpreted language Lua. The key point in this language binding is its dynamic nature. Because of this dynamic nature coercions between data types are possible. Java ORB [21] is compliant to CORBA 2.3 Specification. The services offered by it include Security, Transaction, Notification, Trading, Naming, Persistence and Event services. It offers pure Java RMI/IIOP implementation. Engine Room CORBA [22] is written in Java and produces stub code in C and C++ according to CORBA 2.2 specifications. It can be ported to Win95/NT, Solaris and LINUX platforms. PolyORB [23] is a general middleware technology for CORBA and other distributed systems technologies. It provides a uniform solution to build distributed applications relying either on middleware
standards such as CORBA, the Ada 95 Distributed System Annex, SOAP, Web Services, or to implement application-specific middleware. It is a polymorphic, reusable infrastructure for building or prototyping new middleware adapted to specific application needs.

Arachne [24] is a toolkit for distributed component-based software development. It includes a CORBA 2.0 compliant ORB, CORBA Object Services (COS), an IDL-to-C++ translator, cross-platform portability libraries and a CORBA application framework class library. It is available for Windows 95/NT, Linux, HP/UX, SunOS 4.x, and Macintosh. Voyager is a Java-centric platform for developing distributed software systems. Voyager ORB [25] is a high-performance ORB that simultaneously supports universal communication between Voyager, SOAP, CORBA, Remote Method Invocation (RMI) and Distributed Component Object Model (DCOM) objects. Its dynamic proxy generation removes the need for manual stub generation, and the built-in distributed garbage collection system eliminates the need to explicitly track remote object references. Voyager ORB also includes a universal, federated and distributed naming service, an activation framework for object persistence, advanced messaging and mobile agent technology. FnORB [26] is a CORBA ORB written in the Python language with a language mapping for Python and an Interface Repository. MICO [27][28] is a CORBA 2.1 compliant implementation with DII, DSI, IR, support for object migration and ORB prepared for multiprotocol support. Its IDL compiler tool is used for translating IDL-specifications to C++ as well as feeding IDL-specifications into the interface repository. OmniORB [16] is compliant to CORBA ver 2.6, with C++ and Python language bindings. It is portable to Windows NT / XP, Linux and Solaris 2 platforms. It does not support objects by value and portable interceptors. JacORB [29][30] is an open source ORB based on simple design. It is written in java and has a mapping onto java programming language. It has naming and event services.

Some of the Real Time and Embedded ORBs include e*ORB [31] and ORBexpress [32]. OpenFusion e*ORB has a modular micro-kernel architecture which enables ORB image sizes up to 35Kbytes to be realized.
Footprint optimization has been achieved by exploiting features such as configurable type support and selectable single / multi threading support combined with advanced strategies to reduce code and data segment sizes. Extensive use of static configuration techniques has resulted in a smaller, more predictable data segment with the added benefit of improved real time performance. UORB [33] aims at providing resources for the participation of devices into a CORBA network without requiring a CORBA profile inside the device. Here, CORBA processing occurs in a host other than the device. Visibroker RT [13] is CORBA compliant and its platform independence and high-availability extensions facilitate the development of heterogeneous, mission-critical applications. ORBexpress [32] supports customization of pluggable transports for embedded CORBA applications like avionics. Sankya Vardhi [34] is a high performance, easily configurable, small footprint object middleware for embedded systems. K-ORBs [35] is the general term used to describe a family of customised and configurable ORBs that are based on the minimumCORBA specification that is tailored to the particular requirements of their target domain.

CORBA uses General Interoperable Protocol (GIOP) as its communication protocol. It fits into the application, presentation and session layers in the OSI model. Presentation conversion deals with the conversion of data from local machine representation to a common network format. This process of marshalling is carried out using static stubs generated by the IDL compiler or generic dynamic stubs. Presentation conversion is the major source of inefficiency in CORBA. So, this thesis proposes different methods to improve the process of presentation conversion and hence enhance the performance of CORBA-based applications.

1.2 Scope of the Thesis

The main challenges faced by CORBA-based applications in presentation conversion include reduction in size of the stub code generated by the IDL compiler, time efficiency of marshalling procedures, time efficiency
of Inter Process Communication (IPC) mechanisms and secure communication.

This thesis proposes methods to improve the performance of CORBA-based applications by increasing the speed and reducing the size of the presentation conversion procedures. The performance of presentation conversion routines is measured in terms of Round Trip Transfer Time (RTT). RTT is a factor of the marshal, network and unmarshal time. The major objective of this thesis is to study, analyse and propose methods to enhance the performance of stub code. The research work is carried out in the following areas:

1) Identification of the common sources of overhead in marshalling in CORBA-based applications. The following are some sources of overhead in the presentation conversion process:
   - Presence of extra padding bytes in Common Data Representation (CDR) for alignment.
   - Size of Compiled stubs.
   - Speed of marshalling.
   - Speed of Internet Interoperable Protocol (IIOP).
   - Link-level security in communication.

2) Proposing novel methods addressing the following issues to improve the process of presentation conversion.
   - Minimization of the alignment problems.
   - Reduction in the size of stub code.
   - Reduction in the number of copy operations.
   - Improvements in the marshalling speed.
   - IPC mechanisms using shared memory, multiple segments of shared memory, multiple sockets and parallelizing the operations on shared memory segments and multiple sockets for improving the RTT.
   - Enhancement of link level security.
1.3 Organisation of the Thesis

Chapter 1 gives an introduction to CORBA and its presentation conversion. Chapter 1 also describes the scope of the thesis and the organization of the thesis.

Chapter 2 gives a review report on existing CORBA and other distributed object models. It lists the advantages, disadvantages and overheads in the distributed object models.

Component reuse and assembly is described in chapter 3. Chapter 3 also gives the performance and overhead analysis of various distributed object models. Based on the results of chapter 3, this thesis proposes methods to overcome the various overheads in CORBA-based applications.

Chapter 4 proposes a method to bring about selective multiple inheritance in CORBA. It also proposes a novel reference counting method for construction and destruction of CORBA objects.

Chapter 5 proposes improvements in marshalling by incorporating changes in the transmission medium. The Round Trip Time (RTT) (which is a factor of the marshalling time, networking time, unmarshalling time) improvements on using faster IPC mechanisms when compared to the conventional IIOP protocol used in CORBA is depicted in this chapter.

CORBA has a high potential to be used in Distributed Embedded Systems. This requires efficient presentation conversion procedures. Chapter 5 also proposes new encoding/decoding procedures to generate efficient marshalling routines. This chapter suggests a method to strike a balance between CORBA static invocation mechanisms like interpreted, compiled and inlined and dynamic invocation mechanism. The experimental results of the proposed method are also given in this chapter.

Chapter 6 proposes improvements in presentation conversion in CORBA by proposing changes in the CDR format. This chapter also proposes a novel method of link level encryption to bring about security in CORBA-
based systems. The experimental results of these proposed changes are also presented in this chapter.

The results of the thesis are summarized in Chapter 7. Suggestions for future work are also stated in this chapter.