Test bed for Distributed Object Technologies using Java

490.450DT Project Report
(November 12, 1998)

By

Michael (Thuan-Duc) Ta

Supervised by

Prof. Clark Thomborson and Dr. Radu Nicolescu

The University of Auckland

Information Technology Faculty

Contents

Tables and Figures

Tables

Figures

1. Abstract

Distributed Object Technology (DOT) has become an important part in doing network computing nowadays. There are quite a few implementation (DOT) software packages available on the market that allow developers to write applications that can be distributed across network and across platforms. Java programming language designed from the ground up with networking in mind, it is uniquely suited to build the next generation of network applications. These and other features of Java in-built within the DOT software packages provide developers a flexible way and great simplicity in the distributed object programming development. This project evaluates Java RMI, HORB, Voyager, Microsoft DCOM for Java, two popular commercial CORBA IIOP implementations and compares their performance in respect of some popular DOT characteristics such as Remote Object Connection, Remote Object Creation, Object Data Transfer and Numerical Array Transfer. A socket version of Java or C is also used to compare the operations with these DOT packages where it's appropriate. This project mainly uses a micro-benchmark suite developed by ETL-Japan in measuring the performance of these software packages.

2. Acknowledgement

First of all, I would like to thank those people at the University of Auckland, who have made it possible for me to undertake this project. These are my project supervisors, Professor Clark Thomborson who has helped me to understand benchmarking process and performance analysis in evaluating software; Dr. Radu Nicolescu who has provided his help in giving me an insight view of the Distributed Object Technology, and especially for his initiation in giving me the opportunity to work on such an interesting topics. Also I would like to thank Tamaki System Administrator James Harper for his assistant in software installations and configurations on the two Windows NT machines in the lab.

Secondly, I would like to thank HORB Open for supplying the ETL's micro-benchmark suite, which serves as the fundamental tool for evaluation the performance of the DOT packages in this project. A very special thanks to Dr. Satoshi HIRANO-author of the HORB software that has made writing distributed object technology extremely easy- for helping me in understanding more details in HORB and giving me a wonderful opportunity to be a guest research at ETL, Japan.

3. Introduction

The proliferation of the personal computers and the growing of the web have triggered a need for distributed computing applications. Distributed computing can be seen as an effort to bring multiple network machines, sometimes specialized, hardware that will cooperate with each other in such a way that information and other resources can be shared by all of these connected computers. To meet the demands of the next generation of network application, which will be based on the object-oriented Client/Server computing, the information system today and the future will need to be built upon Distributed Object Computing Technologies. Distributed Object Computing extends the object-oriented programming by allowing the objects to be distributed across heterogeneous network environment. This technology allows objects to reside outside of the application address space but still appear as though they were local to that application. In a simple term, distributed object technology allows objects on different machines to communicate each other using method calls.

There are various implementations based on distributed object technology, currently, the three mainstreams that are dominant to support the distributed object applications development are Java RMI, CORBA and DCOM.

3.1 Project Objectives

The goal of this project is to understand Distributed Object Technology as well as to understand software-benchmarking comparison. It requires to identify and understand the most common set of Distributed Object Technology operations such as Remote Objection Connection, Remote Object Creation, Remote Method Call and the Primitive Array Transfer and to compare the performance of some popular DOT software packages based on these set of operations. For measurement and testing purposes, a Client and Server standalone GUI Java application were developed which uses ETL's micro-benchmark suite as an underlying tool to carry out the benchmarking.

3.2 Documentation Organization

The document starts with a brief overview of the popular Distributed Object Technologies that will be evaluated in the project, putting a special emphasis on the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation and Microsoft's Distributed Component Object Model. Following the overview, I will discuss the evaluation points with respect to the performance that I am going to measure. This chapter will serve as a template in which each individual comparison will be explained in detail for each subsequent chapter. The comparison result is presented and summarized to give the results of the individual measurements of the DOT packages.

4. Overview of Distributed Object Technologies

Object technology has proven to be the next wave of building business applications in corporations. Application development using object technology will benefit the reuse of existing functionality of the objects, it isolates the development and implementation of objects so that modification of the object components will not affect other components in the system as a whole. Lightweight client that interacts with multiple servers will get more comprehensive services and provide load balancing in the network.

Distributed Object Computing (DOC) is a breakthrough framework for computing that has resulted from the convergence of Object Technology and the Client/Server Technologies. In the DOC, objects can work cooperatively with others to fulfill their task, during any given interaction; objects will dynamically assume the role of clients and servers. The physical glue that ties distributed objects together is an Object Request Broker (ORB). This ORB provides the means for objects to locate and activate other objects across a network, regardless of the platform or programming language used to develop either client or server objects.

To simplify the network programming, there are three mainstream distributed object model that serve as standard, they are CORBA by OMG, Java RMI by Sun and DCOM by Microsoft. The following sections will discuss these three technologies together with a couple of other DOT software package that enable developers to build a distributed object version of their application.

4.1 Common Object Request Broker Architecture (CORBA)

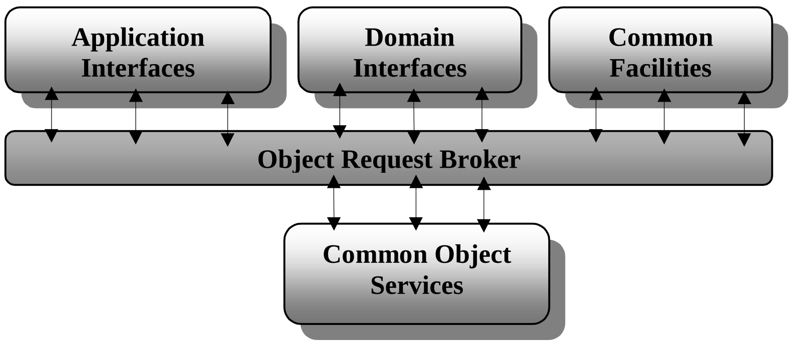
In 1990, the Object Management Group (an international consortium now consisting of more than 800 companies) has developed Object Management Architecture (OMA) as a standard architecture for doing distributed computing in a heterogeneous environment. The OMA is composed of an Object Model and a Reference Model. The Object Model defines how objects distributed across a heterogeneous environment can be described, while the Reference Model is used to characterize the interactions between those objects. In the OMA, objects interacts with other objects through the Object Request Broker (ORB), specified by the Common Object Request Broker Architecture (CORBA), which details the interfaces and characteristics of the ORB component of the OMA.

The idea of CORBA system is that an object is defined in terms of a set of interfaces written in CORBA interface definition language. The implementation of the object is independent of the interface and hidden form other objects.

4.1.1 Object Management Architecture

Figure 4-1 shows the components of the OMA Reference Model. The ORB component is mainly responsible for facilitating communication between clients and implementation objects. There are four object interface categories that utilizes the ORB component.

Figure 4-1: OMA Reference Model Interface Categories



- **Common Object Services**: These are domain-independent interfaces that are used by many distributed object programs. E.g. 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; the Security Service which provides a complete framework for distributed object security.

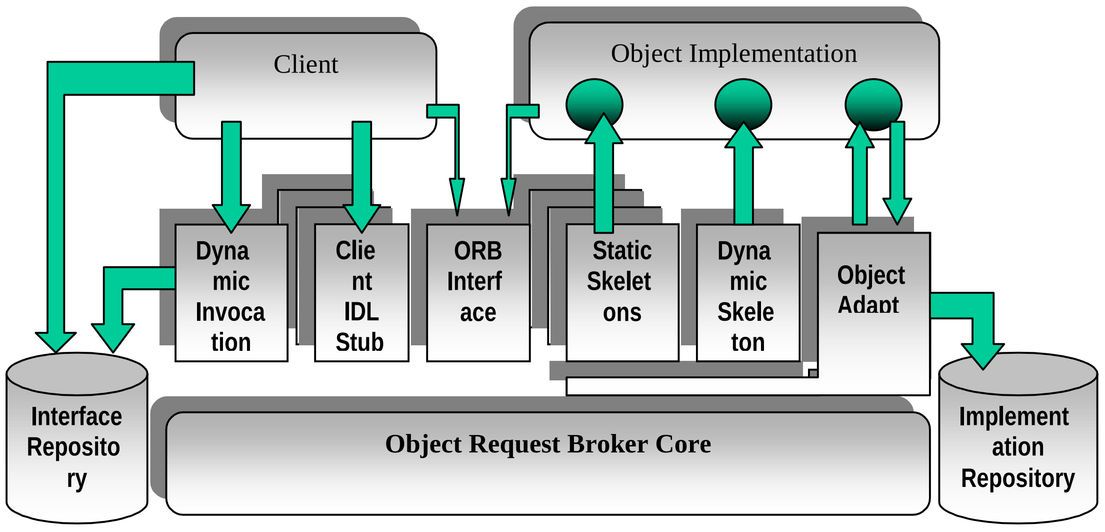
- **Application Interfaces**: These are interfaces developed specifically for a given application. Because they are application-specific, they are not standardized by OMG.

- **Common Facilities**: These interfaces are oriented towards end-user applications. E.g. Mobile Agent facilities, Data Interchange facilities, Business Object Framework facilities.

- **Domain Interfaces**: These interfaces similar to the Object Services and Common Facilities but are oriented towards specific application domains. E.g. Telecommunication domain, Medical domain, Financial domain.

4.1.2 Object Request Broker (ORB)

Figure 4-2: The Common Object Request Broker Architecture



- **Dynamic Invocation**: This interface allows the discovery of new objects, their interfaces and their definitions.
- **Client IDL Stub**: This is a predefined interface to an object service. It defines how clients invoke corresponding services on the server and includes code to perform parameters marshalling.
- **ORB Interface**: For accessing the general services provided by an ORB, both client and implementation use the ORB interface. The operations of this interface include converting object references to strings and vice versa, determining the object implementation and interface, duplicating and releasing copies of object references, testing object references for equivalence and ORB initialization.
- **Static Skeleton**: This provides a predefined interface to each service exported by the server.
- **Dynamic skeleton**: This provides on-demand binding for requests for server components.
- **Object Adapter**: This acts on behalf of the objects on the server. It provides the run-time environment for instantiating server objects, passing requests to them, and assigning them object IDs
- **Interface repository**: This is the place that stores the definitions of known interfaces and let objects to be registered and queried about their interfaces
- **Implementation repository**: This is the place that stores information about the classes a server supports, the objects that are instantiated and their object IDs.

4.1.2.1 The Client Side

The client can request a service from an object by use either static or dynamic invocation. In static invocation, client knows the IDL definition of the requested service at compile time. The definition is used to automatically generate stubs for all requested operations. The client calls the appropriate stub, which translates the data and passes the request to the ORB core for delivery.

In the dynamic invocation, the client discovers the IDL definition of the requested service at run time. To request an operation, the client specifies the target object, the name of the operation, and the parameters of the operation via a series of calls to the standardized Dynamic Invocation Interface. Then the requested service is passed to the ORB core for delivery. The client uses the Interface Repository to obtain an IDL definition at run time. The ORB is responsible for serving the requested service, the client need not care where about the object, how it is implemented or what language of the implementation.

4.1.2.2 The Implementation Side

The implementation side contains the actual service implementation. An automatically generated operation-specific skeleton is used by the ORB to call the implementation of the requested service. In the static mechanism, the IDL definition of the requested service is available at compile time. However, in the dynamic request delivery mechanism, it expects the implementation of the standardized Dynamic Skeleton Interface (DSI) on the implementation side. No compile time information about the interfaces is needed for the dynamic request delivery mechanism. The information necessary to locate and activate implementations of requested services is stored in the Implementation Repository; it provides a run-time repository of information about the classes a implementation side supports. The implementation repository is specific to a particular ORB and environment and is not standardized by CORBA.
In the diverse environment, several different object adapters are implemented by the ORB to provide the services of the ORB itself. However, each ORB must provide the standardized Object Adapter interface. For CORBA 2.2 ORBs and 3.0 ORBs, this Object Adapter is called the Portable Object Adapter.

4.1.3 The Inter-ORB Protocols

The CORBA 2.0 introduced a general ORB interoperability architecture that provides for direct ORB-to-ORB interoperability and for bridge-based interoperability. Two ORBs can communicate directly if they are in the same domain. This is not always true, however, since organizations often need to establish local control domains.

When information in an invocation must leave its domain, the invocation must traverse through a bridge. The role of the bridge is to map ORB-specific information from one ORB domain to the other.

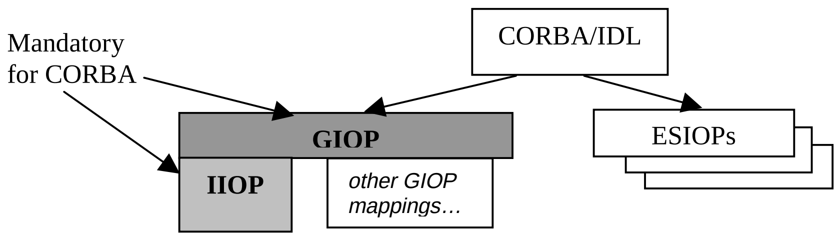
The general ORB interoperability architecture is based on the General Inter-ORB Protocol (GIOP), it specifies standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs over any connection-oriented transport. GIOP is designed to be simple and easy to implement while still allowing for reasonable scalability and performance.

The Internet Inter-ORB Protocol (IIOP) specifies how GIOP is built over TCP/IP transports. The IIOP specifies a standardized interoperability protocol for the Internet, providing “out of the box” interoperation with other compatible ORBs based on the most popular product- and vendor-neutral transport layer. The relationship between IIOP and GIOP is somewhat like the relationship between an object's OMG IDL interface definition and its implementation. GIOP specifies protocol, just as an OMG IDL interface effectively defines the protocol between an object and its clients. IIOP, on the other hand, determines how GIOP can be implemented using TCP/IP, just as an object implementation determines how an object's interface protocol is realized.

Most commercially available ORB products already support IIOP and IORs and have been tested to ensure interoperability. This testing is currently done directly between ORB vendors rather than by an independent testing body.

The ORB interoperability architecture also provides for other environment-specific inter-ORB protocols (ESIOPs). The ESIOPs allow ORBs to be built for special situations in which certain distributed computing infrastructures are already in use.

Figure 4-3: **Inter-ORB Protocol Relationships**



4.1.4 CORBA Compliant Software Products:

4.1.4.1 VisiBroker

VisiBroker is a full implementation of the CORBA 2.2 standard. VisiBroker is now distributed by the Inprise Corporation. It was formerly distributed by Visigenic, and is based on the code of Orbeline (PostModern Computing.) The ORB is provided for several C++ and Java platform. In this project, the version VisiBroker 3.3 for Java is tested again other package. Since CORBA version 2.2 does not support transfer object by value, VisiBroker has come up with its own function that facilitates the transfer of objects by value. VisiBroker's **java2iiop** utility generates codes that can be used to program for object transfer by value. The **java2iiop** compiler lets us define interfaces and data types in Java that can be used as interfaces and data types in CORBA.

4.1.4.2 OrbixWeb

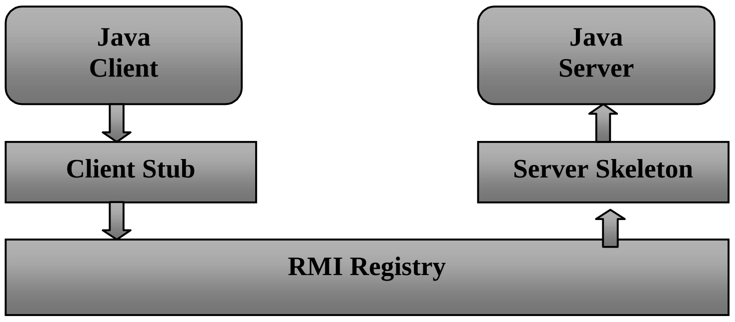
Iona is a leading provider of CORBA technology. Its Orbix ORB now runs on more than 20 operating systems-including twelve Unix variants, OS/2, Windows NT, Windows 95, Windows 98, Macintosh System 7.5 Open VMS, MVS...Orbix is a full implementation of the CORBA 2.2 standard. Orbix provides complete ORB, BOA, DII, DSI, Interface Repository and Implementation Repository. It also supports server activation policies, exceptions and context handling.
The Iona's OrbixWeb version 3.0 is a complete all-Java ORB. Iona is also a provider of CORBA firewalls, DCOM-to-IIOP bridges, and CORBA-based transaction services.

4.2 Sun's Java Remote Method Invocation (Java RMI)

With the release of JDK version 1.1 by Sun in 1996, Java has its own, built-in native ORB, called Remote Method Invocation (RMI). RMI is the action of invoking a method of a remote interface on a remote object. However, the important thing here is that a method invocation on a remote object has the same syntax as a method invocation a local object. In Java RMI, Java objects running on different Java Virtual Machines (JVM) can communicate each other, providing a homogeneous environments. Though the version 1.1 RMI is an ORB in the generic sense that it supports making method calls on remote objects, it's not a CORBA-compliant ORB. However, the new version of JDK Java 1.2 extends the Java's distributed object capabilities by adding CORBA capability to Java, providing standards-based interoperability and connectivity. Java 1.2 enables distributed Web-enabled Java applications to transparently invoke operations on remote network services using OMG IDL and IIOP defined by the OMG.

Java RMI allows programmers to pass object by value, i.e. the actual state of the object is copied in the call. Java RMI is native to Java, it depends on many features of Java such as object serialization, portable, down-loadable object implementations, Java interfaces definitions. Java uses its own Java interfaces instead of IDL code to describe remote object. Implementing remote objects requires extending or implementing the Remote interface, the Serializable interface and the UnicastRemoteObject class. The resulting is that it is natural for Java programmers to use.

<u>Figure 4-4</u>: The Java RMI Architecture



To invoke a remote method, the Java client makes a call to the client proxy (stub), the client stub packs the call parameters into a request message and invoke a wire protocol like Java Remote Method Protocol in Java RMI (other possible protocols are Internet Inter-ORB Protocol (IIOP) in CORBA or Object Remote Procedure Call (ORPC) in DCOM) to ship the message to the server. At the server side, the wire protocol delivers the message to the server side stub (skeleton), the server side skeleton then unpacks the message and calls the actual method on the object.

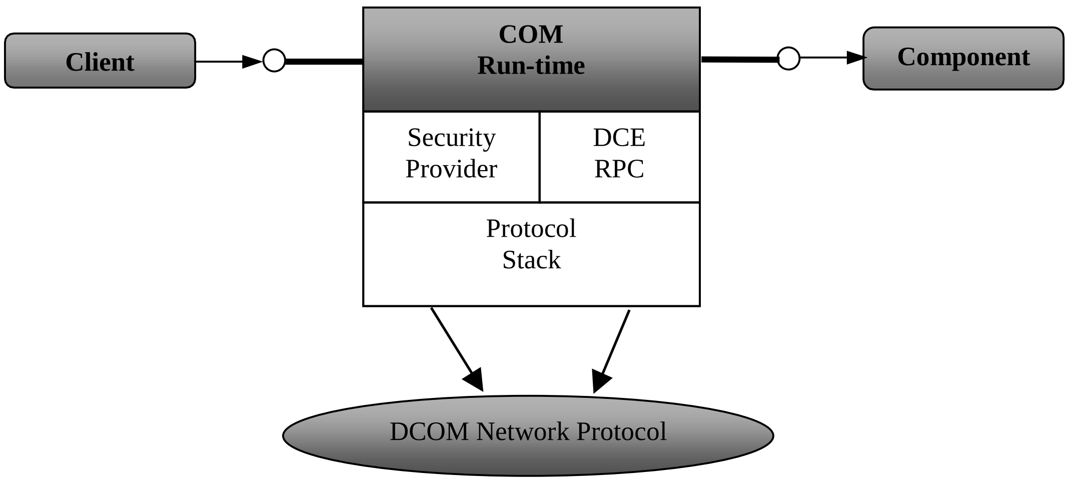
4.3 Microsoft's Distributed Component Object Model (Microsoft DCOM)

If CORBA is the industry's leading standard for distributed objects, then Microsoft's Distributed Component Object Model (DCOM) is the defacto "other standard". DCOM is introduced in 1996 by Microsoft as a solution to distributed objects, DCOM, previously known as Network OLE, is an extension of the COM design to networked applications. DCOM has its own network protocol called Object Remote Procedure Call as corresponding to CORBA. DCOM is the foundation for Microsoft's Internet and component strategy. Today, DCOM is included in NT 4.0, in every copy of windows 98. DCOM can also be downloaded separately at Microsoft site.

In October 1996, Microsoft shipped its Visual J++ development tool for Java which has Java language bindings for DCOM, client objects written in Java can use DCOM to invoke remote Java server objects as well as COM components written in other languages.

The key features incorporated into the DCOM architecture comprise of language independence, integrated Windows NT wire-level security, transport neutrality (with the ability to communicate using TCP/IP, UPD/IP, IPX/SPX, AppleTalk and HTTP), and static or dynamic invocation of objects.

Figure 4-5: DCOM architecture

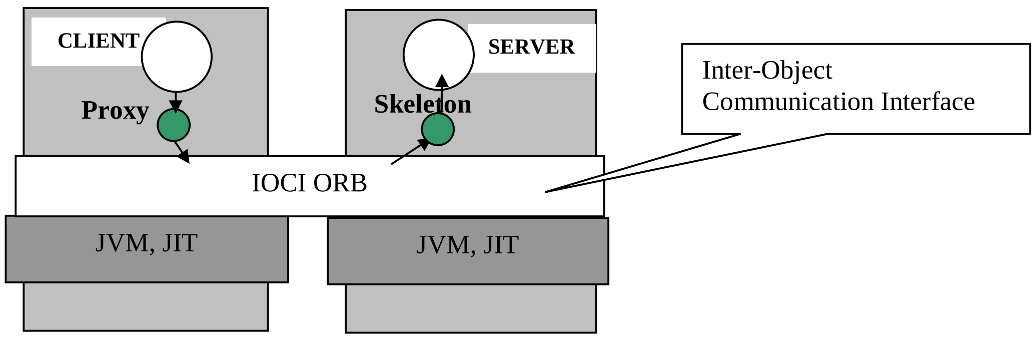


4.4 HORB

HIRANO ORB (HORB) is a light weight pure-Java ORB that works seamlessly with Java JDK to provide distributed object capability. Programming distributed object in HORB is very simple and natural to Java programmer because the Java language specification has not been altered. There is no need to write IDL code for remote object. Any Java classes can become remote object by using HORB compiler and can be used remotely by declaring it as XXX_Proxy, where XXX stands for class name. The key features of HORB include supporting transfer object by value or by reference, dynamic remote object creation, remote object connection, remote method call (synchronous or asynchronous) and mobile objects.

Proxy object represents the remote object to the client, Skeleton object on the server interacts directly with the actually implementation of the object to facilitate the requested service from the client. Proxies and Skeletons communicate with each other through ORB. Proxies and Skeletons are generated by the HORBC compiler.

Figure 4-6: HORB architecture



4.5 Voyager

Voyager is a very popular agent ORB for Java developed by ObjectSpace. The Voyager version 2.0 has just been released on the September 1998. Voyager for Java leverage the language feature to simplify distributed computing. When a remote object is constructed using Factory.create(String classname, String destination), a proxy object whose class implements the same interfaces as the remote object is returned. Voyager dynamically generates the proxy class at run time. The proxy can receive message, forward them to the object, receive the return value, and pass the return value to the original sender.

Key features of Voyager 2.0 are Mobile Agents, Object Persistence, Multicast and Publish/Subscribe for Group Communication, Network Class loading as well as interoperable with CORBA system. With

Voyager, a programmer can use any existing Java class as a CORBA server and let Java object connect to it.

5. Evaluation Discussion

There are many features that can be used to justify a preferable package for distributed application development such as ease of programming, rapid development, performance, robustness, cross languages and platforms, service support, garbage collection, etc. In the project, I evaluate the performance of the DOT packages based on a small set of criteria that are considered to be most prevalent on most of the DOT packages, these include: Remote Object Connection, Remote Object Creation, Remote Method Call, Object Array Transfer, Numerical Array Transfer. These criteria will be discussed in the following sections. Since performance of OO software often relates to the garbage collection mechanism that particular software can handle, I also compare the way garbage collection mechanism that is used in each DOT package.

Performance evaluation of the DOT package in the project is relating to the latency aspect in doing distributed computing. Latency is the most obvious difference between a local object invocation and the distributed object invocation. Latency in distributed computing depends on two factors: hardware and software. The advance in hardware technology has made the hardware latency factor a less relevant than the software latency. Latency depending on software makes developers to find out more efficient in designing their software product, in this case the implementation of DOT package.

In the following section, I will discuss the evaluation points that are going to be tested for DOT packages. Each operation is discuss separately, any special considerations will be listed out as well.

5.1 Remote Object Connection

The ability for a client object to connect and reconnect to an existing remote object that was registered on the server is an essential requirement in the distributed object computing because the underlying connection between objects can be failed any time. This operation provides a way for binding and rebinding an existing object using a bootstrap reference. Each DOT has each own method to resolve the binding mechanism, except Microsoft's DCOM which we can't connect to an existing DCOM object because DCOM objects only expose their interfaces to outside world, and interface object can potentially point to DCOM object implementation which resides on different locations. In this project, the remote object connection is evaluated to see how long it takes to bind a client object to an existing object on the server. This operation is related to the automatic reconnection mechanism that allows a client to automatically reconnect to a server in case of a communication failure.

5.2 Remote Object Creation

Creating a remote object in a specified location is a basic feature in the distributed object computing. After the remote object is created, its services can be requested by client object. Creating a remote object includes these steps:

- instantiating of the actual implementation of an object
- creating a object ID and returning it to the client
- registering of the implementation object with the reference into the repository

This mechanism is implemented differently for each DOT package. In general, a proxy object or stub is created on the client side to represent the remote object on the server side. In the project, I evaluated the time it takes a client object that had already connected to the existing server object to create a new object remotely.

5.3 Remote Method Call

The purpose of building distributed object application is that the request for services on the object can be delivered transparently across the network. For remotely invoking a method on an object, a number of steps occurred as follows:

- A call is first directed to the proxy (stub) object on the client side
- The proxy (stub) object is responsible for communicating the request to the actual implementation of objects on the server. This operation includes marshalling an unmarshalling the request, consisting of initializing and filling the object name, operation name, arguments of the call. The performances of the operation will be mostly related to how efficiently the proxy encoding of marshaled data. Therefore, the speed of the operation is an important comparison in the project.
- The marshaled form of the request is delivered across the network by ORB. In the project, TCP/IP protocol stack is used as and underlying transport mechanism that the ORB interacts with in order to deliver the packet over the network and is considered the same for all DOT packages.
- The marshaled form of the request arrives on the server and dispatched by the ORB. The identification of the service request on the actual object is processed.
- The skeleton (server-proxy) on the server side performs the process of unmarshalling the request. This is an opposite process to marshalling on the client side
- The call on the object implementation is made

As we can see, there are a lot of processes occurred for a call on a remote object. The step for a callback from the server to the client is the same as an up-call as the described above.

In the project, I measured the operation of an up-call made by the client to the server. The performance of a DOT with respect to the remote method call test will depend heavily on how clever the DOT packages implementing the process of marshalling and unmarshalling the request. The speed of delivery the request also affected by the communication mechanism of the underlying network.

5.4 Object Data Transfer

Passing object instances as a remote method call parameter or receiving object instances from the server as the return value for a remote method call is called " object passing". In the project, I measured the time it takes a client object to transfer by value an array of data object to the server. An "int" data is encapsulated in the object data.

Object's state is preserved when passing object-by-value inside a parameter of a method call. Object-by-value passing gives programmer a flexible way in using object polymorphism features, i.e. we can use a variable of a superclass type to hold a reference to an object whose class is the superclass or any of its subclasses. The current CORBA version does not support passing object-by-value and currently implemented by individual vendor. Passing object-by-value will be supported in the CORBA version 3.0.

CORBA 3.0 support for Objects-by-Value provides a straightforward and efficient method of passing information in an object-oriented form across a network. Support for Objects-by-Value will help programmers integrate CORBA more seamlessly into modern programming languages like Java. Extending IIOP support to value objects is also an enabler for CORBA 3.0's new Java-to-IDL mapping.

5.5 Numerical Array Transfer

The performance of transferring numerical array by DOT is evaluated between themselves and against a raw socket version of Java and C. Since DOT adds a lot of overhead in doing byte-reordering for the network byte order, its performance on numerical array transfer is crucial in application that involves heavy numerical calculation such as in High-performance network computing, in Digital Image processing, in Matrix calculation. In the project, I measured the performance of DOT in respect of transferring the "byte", "int" and "double" array. The size of the array is determined at runtime.

5.6 Garbage Collection Issue

Garbage collection mechanism is the way that the program reclaims the unused computer resource. In a distributed system, just as in the local system, it is desirable to automatically delete those remote objects that are no longer referenced by any client. This frees the programmer from needing to keep track of the remote object clients so that it can terminate properly.

A garbage-collection algorithm can significantly affect program performance, if it is implemented incorrectly, it can result in memory leaking or exhausting system memory. It can also stop a running program, leading to a low response from the program. In the project, I describe different garbage collections used by the DOT packages without going into details how it is implemented and how best it performs. This is because there is no single garbage-collection algorithm can satisfy the needs of all computing platforms-even for the same programming language. This area has been studied and researched for the past decades in helping to develop better garbage collectors.

6. Java RMI

DOT: Java RMI (Java Remote Method Invocation)

Developed by: Sun Microsystems. Website: http://www.javasoft.com/

Version tested: JDK 1.1.6

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - Sun's JDK 1.1.6 Java Compiler
- Java VM:
  - Sun's JDK 1.1.6 with Symantec's JIT built in

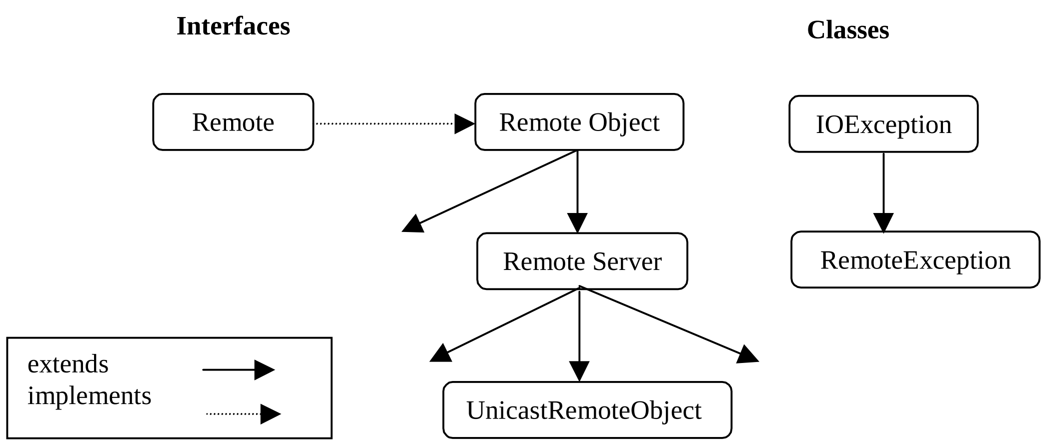
Java RMI is a distributed object model for the Java Platform. RMI is unique in that it is a language-centric model that takes advantage of a common network type system. Specifically, Java RMI enables developers of DOT to treat remote objects and their methods very much like normal Java objects. All the implementation details are hidden. We just need to import one package, look up the remote object in a registry and make sure to catch RemoteExceptions when we make a method call on the remote object.

In the Java distributed object model, a remote object is the one whose methods can be invoked from another Java Virtual Machine (JVM), potentially on different host. A remote object is described by one or more remote interfaces, which are Java interfaces that declare the methods of the remote object. The interface serves as a contract to ensure type consistency between the client and the server object.

The interfaces and classes that are responsible for specifying the remote behavior of the RMI system are defined in the java.rmi and the java.rmi.server packages.

The following figure shows the relationship between these interfaces and classes.

Figure 6-1: Relationship between interfaces and classes in Java RMI



The Java RMI introduces a new object model which extends the pre-JDK 1.1 object model, the Java Distributed Object Model. Here are the new features that were added to the Java object model:

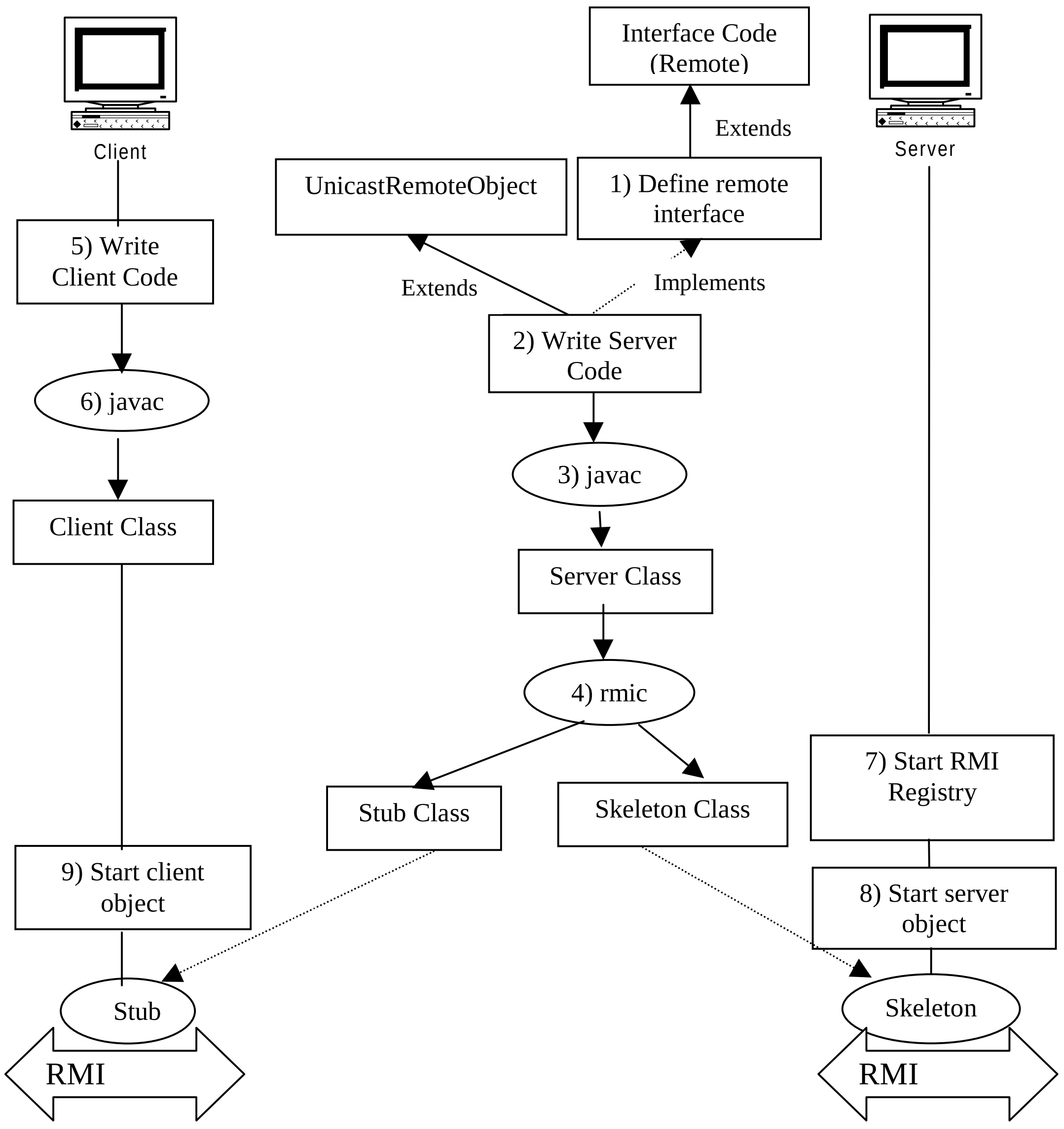
- A *Remote* object is passed by reference, not by copying the actual object implementation
- *Nonremote* arguments in a remote method call, as well as the results returned by the call if there is any are passed by value rather than by reference. This is because references to objects are only useful within the same Virtual Machine. This passing mechanism in the Java RMI is relying on the *Object Serialization* service, which converts the Java objects into a serial stream so that the object state is preserved and can be recovered after they had been transmitted across the network.
- The client of the remote objects only deals with the remote interfaces, not with the actual implementation of the object.
- The client invokes methods on the remote objects has to deal with additional exceptions that can occur during a remote method invocation. This is because the failure modes of invoking remote objects are more complicated than the failure modes of invoking local objects.
- Extra security mechanisms are introduced to control the activities that a remote object can perform on a system. A SecurityManager object needs to be installed to check all operations of the remote object on the server. A custom security managers can be defined for specific applications. No remote method invocation occurs if there is no SecurityManager object installed by the server objects.

For more information of Java RMI, please visit Java RMI homepage:
http://www.javasoft.com/products/jdk/rmi/index.html

6.1 The development process

The figure below shows the development process to write a RMI Client and Server Application.

Figure 6-2: Distributed object application development in Java RMI



1. **Define the server interface.** The server object must declare its services via a remote interface. In RMI, this is the java.rmi.Remote interface. Each declared method in the server interface must throw a java.rmi.RemoteException. Here is the code for this:

```
//Server.java
package DOTBenchmark.rmi;
import java.rmi.*;
public interface Server extends java.rmi.Remote {
        Server createInstance() throws RemoteException;
        int methodA1(int a1) throws RemoteException;
        int methodA2(int a1, int a2) throws RemoteException;
```

```
            int methodA3(int a1, int a2, int a3) throws RemoteException;
            int methodA4(int a1, int a2, int a3, int a4) throws RemoteException;
            int methodA5(int a1, int a2, int a3, int a4, int a5) throws RemoteException;
            int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) throws RemoteException;
            void methodDA(Data[] dat) throws RemoteException;
            void methodB(byte[] a) throws RemoteException;
            void methodI(int[] ia) throws RemoteException;
            void methodD(double[] da)throws RemoteException;
    }
```

2. **Write server code.** The server code implements the server interface defined above. The implementation of this server code must also extend the java.rmi.server.UnicastRemoteObject. The java.rmi.server.UnicastRemoteObject class provides support for point-to-point active object references using TCP-based streams. The server implementation must also include the code that installs a SecurityManager object that will register the server interface within the RMI naming context using the Naming.bind (or rebind) method. This step is required so that an instance of the remote object is registered into the RMI registry. The RMI registry will locate this object so that client can reach them. Here is the code:

```
 // ServerImpl.java
 package DOTBenchmark.rmi;
 import java.rmi.*;
 import java.rmi.server.*;

 public class ServerImpl extends UnicastRemoteObject implements Server {
     String serverName;
public ServerImpl() throws Exception {
     super();
}
public Server createInstance() throws RemoteException {
     ServerImpl obj = null;
     try {
          obj = new ServerImpl();
          // server object registered to the rmi registry at the // port 2005 (default port for RMI is 1009)
          // The rebind method binds an object to the register no
          // there exits an old object already binded to it or not
          // With this binding, the old binding is lost
          Naming.rebind("//"+serverName+":2005/Server", obj);
          return obj;
     } catch (Exception e) {
          e.printStackTrace();
     }
     return obj;
}
public int methodA1(int a1) throws RemoteException {
     return(a1);
```

```
    }
    public int methodA2(int a1, int a2) throws RemoteException {
        return(a1);
    }
    public int methodA3(int a1, int a2, int a3) throws RemoteException {
        return(a1);
    }
    public int methodA4(int a1, int a2, int a3, int a4) throws RemoteException {
        return(a1);
    }
    public int methodA5(int a1, int a2, int a3, int a4, int a5) throws RemoteException {
        return(a1);
    }
    public int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) throws RemoteException {
        return(a1);
    }
    public void methodDA(Data[] dat) throws RemoteException {}
    public void methodB(byte[] a) throws RemoteException{}
    public void methodI(int[] a) throws RemoteException{}
    public void methodD(double[] a) throws RemoteException{}
    public static void main(String argv[]) throws Exception {
        System.setSecurityManager(new RMISecurityManager());
        ServerImpl obj = new ServerImpl();
        Naming.rebind("//"+serverName+":2005/Server", obj);
        System.out.println("bind done");
        System.out.println("Server is listening at " + serverName + ":2005/Server");
    }
}
```

3. Compile the server implementation code. We use javac to compile the server implementation code into server object. Enter this command at the dos command prompt windows:

   prompt>javac -O ServerImpl.java

   Note. In this benchmark project, we will compile all Java code with option O (optimizer turned on).
4. Generate server object stub and skeleton. We use RMI Compiler rmic to obtain the remote object stub and skeleton. Since our ServerImpl.class is reached via package statement, we need to provide a fully qualified path to ServerImpl class. This is done by entering this command at the dos command prompt windows as follows:

   prompt> rmic –O DOTBenchmark.rmi.ServerImpl.class

   If this command runs successfully, it generates ServerImpl_Stub.class which we need to copy it to the client machine and the ServerImpl_Skeleton.class which we need to copy it to the server machine
5. Write the client code. The client code is just an ordinary Java program, it can locate the remote object by using the java.rmi.Naming class. Here is the piece of code that our client will need to look up the remote object.

```
//Client.java
package DOTBenchmark.rmi;
import java.rmi.*;
public class Client  {
    public static void main (String[] args) {
    System.setSecurityManager(new RMISecurityManager());
    try {
        System.out.println("looking up...");
        Server server = (Server)Naming.lookup("//"+host+":2005/Server");
        System.out.println("binding to the server...done");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

6. Compile the client code. This can be done by using javac (with option O) to compile Client.java into Client.class
7. Start up the RMI Registry. We need to startup the RMI Registry and let it run in a separate process on the server machine. Open a new dos command prompt, enter this command:
   prompt> rmiregistry
8. Start up the server object. We must load the server object and then create instances of the remote objects on the server. This can be done by entering this command at the command prompt windows:
   prompt> java DOTBenchmark.rmi.ServerImpl
9. Start the client. Run the client object with an iteration and location of the server option by entering this command at the command prompt windows:
   prompt> java DOTBenchmark.rmi.Client 100 *serverhostname*

This command runs the client program with an input command line iteration argument of 100 and a server host name as *serverhostname*.

6.2 Remote Object Connection

Java RMI supports the remote object connection model. Multiple clients can share one remote object. Before a client can call a remote method, it needs to retrieve a remote reference to the remote object on the server. It asks for the remote reference by calling the registry's `lookup` or `list` (`url`) method. The RMI registry is defined by the **Registry** interface. In fact, the registry is a remote RMI server object itself, i.e. its services can be accessed across a network using remote method invocations. The Registry is a non-persistent implementation and it only supports flat names scheme and it forgets all its contents when it is restarted. Here is the code for client to obtain a remote reference to the remote object.

```
Server server = (Server)Naming.lookup("//"+host+":2005/Server");
```

As we can see, the client program tries to lookup a remote object with an alias in RMI named "Server" at a particular host at the port 2005. Before it can use this reference, it must be cast to the remote interface that is implemented by the remote object implementation (not the actual class which is hidden from clients). Once the object is retrieved, the client can make the methods call supported by this server interface.

In the benchmark, I measure the time taken a client program to connect to an existing remote object on the server. The benchmark below shows the average time of 100 `lookup()` method call by the client on the same machine and between 2 NT machines connected by a 100 Mbps Ethernet LAN.

Table 6-1: Remote Object Connection

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6
10.90 msec	11.01 msec

Later in this project, I will give a comparison between Java RMI and some other DOT packages such as HORB, Voyager and CORBA Compliant packages.
However, with the precision of the JVM on Windows 98 and Windows NT 4 SP3 is 55ms, i.e. the interval between 2 clock tick of the JVM is 55ms, we can see that the performance of the lookup() method of Java RMI is reasonable good.

6.3 Remote Object Creation

Remote object creation is a process of creating a new object so that its services can be used any client program. We need to be able to construct a new object only when we needed because it would a waste of resources if we make all objects available for remote access but there is no need. Java RMI does not directly support remote object construction. However, we can emulate this feature using the following steps:

- Construct a custom factory object on the server
- Pass the name of the class and the URL of the codebase to the factory object
- Use RMIClassLoader to load the class across the network
- Use reflection to construct a remote instance using the default constructor. To construct an object using parameter needs some further steps.

Class Loader in RMI

For an RMI application to act as a source for classes that can be loaded across the network, the program must be running an HTTP server. This situation sometimes is not favorable and pretty clumsy because we need to install a full-brown HTTP server on the machine just for loading classes. Java RMI provides developers a network class loader that can be used for this purpose.

To enable downloading from a network source, each remote object server must be configured with the `java.rmi.server.codebase` property which specifies where application classes and generated stubs/skeletons reside. When the codebase property is specified, the RMI system embeds the URL of a class in the serialized form of the class. This codebaseproperty is read the RMIClassLoader.loadClass method to load classes over the network.

For example: This example is taken directly from the Java RMI specification.

```
import java.rmi.RMISecurityManager;
import java.rmi.server.RMIClassLoader;

public class LoadClient
{
        public static void main()
        {
                System.setSecurityManager(new RMISecurityManager());
               try {
                        Class cl = RMIClassLoader.loadClass("myclient");
                        Runnable client = (Runnable)cl.newInstance();
                        client.run();
                } catch (Exception e) {
                        System.out.println("Exception: " + e.getMessage());
                        e.printStackTrace();
                }
        }
}
```

In order for this code to work, we need to specify the java.rmi.server.codebase property when you run the bootstrapping program so that the loadClass method will use this URL to load the class. For example:

prompt> java –Djava.rmi.server.codebase=http://host/rmiclasses/ LoadClient

Instead of relying on the property, we can supply our own URL:

```
Class cl = RMIClassLoader.loadClass(url, "myclient");
```

Once the client is started and has control, all classes needed by the client will be loaded from the specified URL. This bootstrapping technique is exactly the same technique Java uses to force the AppletClassLoader to download the same classes used in an applet.

Without this bootstrapping technique, all the classes directly referenced in the client code must be available through the local CLASSPATH on the client, and the only Java classes that can be loaded by the `RMIClassLoader` over the net are classes that are not referred to directly in the client program; these classes are stubs, skeletons, and the extended classes of arguments and return values to remote method invocations.

Security in RMIClassLoader

There must be a security manager in place when the RMIClassLoader attempts to load classes from the network. Applications can either define their own security manager or use the restrictive `RMISecurityManager`. If no security manager is in place, an application cannot load classes from network sources. Once a class is loaded by the `RMIClassLoader`, any classes used directly by that class are also loaded by the `RMIClassLoader` and thus are subject to the security manger restrictions.

In this project, the benchmark for Java RMI Remote Object Connection is simulated as follows:

- we creates a remote server object and register it into the registry in advance
- we then call the `createInstance` method which result in a new remote server object
- the result benchmark is an average of 100 time of calling `createInstance` method by the client

Here is the part of code used for testing this operation:

```
// The createInstance() implementation in the ServerImpl.java
public Server createInstance() throws RemoteException {
    ServerImpl obj = null;
    try {
        obj = new ServerImpl();
        Naming.rebind("//"+serverName+":2005/Server", obj);
        return obj;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return obj;
}

// The remote object creation benchmark in the Client.java
Server server = null;
Server server2 = null;
System.gc();
start = System.currentTimeMillis();
 for (i = 0; i < iteration; i++) {
    server = (Server)Naming.lookup("//"+host+":2005/Server");
    server2 = server.createInstance();
    // ?? how do I release the connection???
  }
  time = System.currentTimeMillis() – start;
  elapse = (double)(time)/(double)(iter);
  System.out.println("Remote object creation: "+elapse+" msec");
```

Table 6-2: Remote Object Creation benchmark

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6
36.20 msec	40.66 msec

From the performance above, we can see that the time it takes for creating a remote object on the server is about 4 times longer than the time it takes to connect to an existing remote object. Users have to pay the cost for creating a new remote on demand to exchange for flexibility. Later on we will see that some other DOT packages which has its own native method to directly create a remote object on the server. The result will shows much higher performance then the way we simulate object creation in JDK 1.1.

However, in the new version of JDK 1.2, the RMI has several new enhancements. Among these are the Object Serialization enhancement, the Remote Object Activation, which introduces a way to support for persistent references to remote objects and automatic object creation via these references. Also with the introduction of the class java.rmi.activation.Activable and the RMI daemon, remote objects can now be created and executed as and when they are needed. Programs need only register the implementation information for the remote objects and the rmid daemon will provide the spawned JVM instance as needed. I expect that RMI remote object creation to show much better performance with this new way of creation remote object. However, this feature is not confirmed yet in this project.

6.4 Remote Method Call

Remote method call in Java RMI is very straightforward, after a remote object is returned, the client program can make method call on that remote object reference just as an ordinary local method call. All the detail of how the method invoked is totally hidden from the programmers. The process on marshalling and unmarshalling the call message is taken care by the stub/skeleton classes which is generated by the "rmic" compiler.

Here is the code to measure the performance of making a remote method call with different number of arguments in the Client.java program.
(Please refer to the appendix for more detail implementation of the Java RMI Client/Server benchmark code.)

```
// rmc with one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// rmc with two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA2(100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

// rmc with three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA3(100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

// rmc with four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA4(100, 100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");
```

```
// rmc with five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

// rmc with six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA6(100, 100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
```

Below is the table shows the performance of making remote method calls on the same machine and on 2 separate Windows NT machines connected through a 100 Mbps Ethernet LAN. The remote method takes integers as arguments and returns one integer.

Table 6-3: Remote Method Call benchmark with different arguments

Remote Method Call with different arguments											
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6 (Unit time in msec)					
1 arg	2 args	3 args	4 args	5 args	6 args	1 arg	2 args	3 args	4 args	5 args	6 args
1.6	2.2	2.2	2.2	2.2	2.2	1.4	1.6	1.6	1.7	1.7	1.7

Looking at the result above, we can see that the result performance varies very little with small different number of argument call. And the remote method call over the network shows better performance than remote method call on the same machine. This shows that the Java VM is to be quite CPU-intensive.

6.5 Object Data Transfer

Java RMI relies heavily on Java Object Serialization, which allows objects to be marshaled as a stream. An argument to or a return value from a remote object can be any Java type that is serializable. This includes Java primitive types, remote Java objects, and nonremote Java objects that implement the java.io.Serializable interface. Subclasses of a serializable class are also serializable. For e.g., java.lang.Integer and java.lang.Double are serializable because the class the extend, java.lang.Number is serializable.

Object passing can be one of the following:

- instance object passing:
    - When passing a remote object as a parameter, the stub for the remote object is passed. The remote object implementation remains in the Java VM in which it was created and does not move, i.e. a remote object is passed by reference in an RMI remote method call
    - When passing a nonremote object as a parameter of a remote method invocation or returned as a result of a remote method invocation, the object is passed by copy, i.e. the content of the nonremote object is copied before invoking the call on the remote object. By default, only the **nonstatic** and **nontransient** fields are copied in Java RMI object passing. When a nonremote object is returned from a remote method invocation, a new object is created in the calling virtual machine. If an object contains an object that is not **static** or **transient** field, then its content is also

transferred as long as and up to the highest serializable object can be reached from the object's graph. This means the shape of object graphs are kept when transferring.

- class passing:
  - For applets, the class is dynamically loaded via the AppletClassLoader if the class is not available locally.
  - For an applications, these classes are loaded by the network class loader that loaded the application; this is either the default class loader (which uses the local class path) or the RMIClassLoader.loadClass method (which uses the server's codebase property).

More on Java Object Serialization
The objective of the Java Object Serialization system is to allow a bytestream to be produced from a graph of objects, sent out of the Java environment (either saved to disk or sent over the network) and then used to recreate an equivalent set of new objects with the same state.

When a local machine passes a remote object to a remote machine, the proxy of the object is transferred. The actual implementation of the object has never really left the remote machine. When the local machine passes one of its own objects to the remote machine, it makes a copy of the object and sends the copy of the object to the server.

The processes of copy an object and serializing it is done with the serialization system in Java. This is more difficult than it first sounds because objects include other object as fields also need to be copied and transferred when the object is copied transferred. However, for security reasons, not all fields in the object can be serialized and transferred; for e.g. we don't want an object which includes a private password field to be serialized and transferred when that object is serialized and transferred. Therefore, fields that contain this kind of information should be declared **transient** which prevents them from being serialized and transferred by the Serialization System.

Object Serialization defines two protocols, which allow the container to ask the object to write and read its state. To be stored in an Object Stream, each object must implement either the **Serializable** or the **Externalizable** interface.

For a Serializable class, Object Serialization can automatically save and restore fields of each class of an object and automatically handle classes that evolve by adding fields or supertypes.

For an Externalizable class, Object Serialization delegates to the class complete control over its external format and how the state of the supertype is saved and restored.

The object data transfer benchmark
In this project, I measure the performance of transferring an array of simple object Data which encapsulates a single field of type "int" from the client to the server. The time measurement is an

average of 100 remote method calls for transferring array of different size from the client program. The Data object is defined as follows in Java class:

```
// Data.java
package DOTBenchmark.rmi;
import java.io.*;
import java.rmi.*;

public class Data implements Serializable {
  int a;
}
```

From the client program, it calls the method of transferring the array of data as defined in the following code:

```
// The transferring array of object Data in Client.java
int[] sizes = {1, 10, 20, 30, 40, 50};
Data data;
for (int s = 0; s < sizes.length; s++) {
    data = new Data[sizes[s]];
    for (j = 0; j < sizes[s]; j++) {
        data[j] = new Data();
    }
    System.gc();
    double start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        server.methodDA(data);
    }
    double time = System.currentTimeMillis() – start;
    double elapse = (double)(time)/(double)(iter);
    System.out.println("void server.methodDA(DOTBenchmark.rmi.Data["+sizes[s]+"]): "+elapse+" msec");
}
```

Table 6-4: Object Array of Data Transfer with variable length

Object Data Transfer with variable length											
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java JDK 1.1.6 (Unit time in msec)					
1 obj	10 objs	20 objs	30 objs	40 objs	50 objs	1 obj	10 objs	20 objs	30 objs	40 objs	50 objs
3.8	3.3	4.9	6.1	7.1	8.8	4.1	4.11	6.01	7.31	9.01	10.82

The test shows that time to pass an array of simple data object depends mostly on the number of items. This means more processing in marshalling and unmarshalling is involved

6.6 Numerical Array Transfer

Here we test the ability of transferring array of primitive data type in Java to see its performance. Again the test is carried out on the same machine and across two networked machines. The client program make remote method call to transfer the array of "byte", "int" and "double" which has the array size determined at run-time. The codes we use for this benchmark are as follows. (For a complete client code, please refer to the appendix section in this project)

```
// byte[]
bufb; int[]  bufi;
double[] bufd;
int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4, 4096*8, 4096*16, 4096*32, 4096*64 };
// to send byte array;
for ( j = 0; j < array_sizes.length; j++) {
    bufb = new byte[array_sizes[j]];
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        server.methodB(bufb);
    }
    time = System.currentTimeMillis() – start;
    elapsed = (double)(time)/(double)(iter);
    System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapsed" msec, "
            +((double)(array_sizes[j])/elapse) + " KB/sec");
}


 // to send int array
```

```
for ( j = 0; j < array_sizes.length; ; j++) {
    bufi = new int[array_sizes[j]];
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        server.methodI(bufi);
    }
    time = System.currentTimeMillis() – start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
  }

  // to send double array
  for ( j = 0; j < array_sizes.length; ; j++) {
       bufd = new double[array_sizes[j]];
       System.gc();
       start = System.currentTimeMillis();
       for (i = iter; i > 0; --i) {
           server.methodD(bufd);
       }
        time = System.currentTimeMillis() – start;
       elapse = (double)(time)/(double)(iter);
       System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
           +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
  }
```

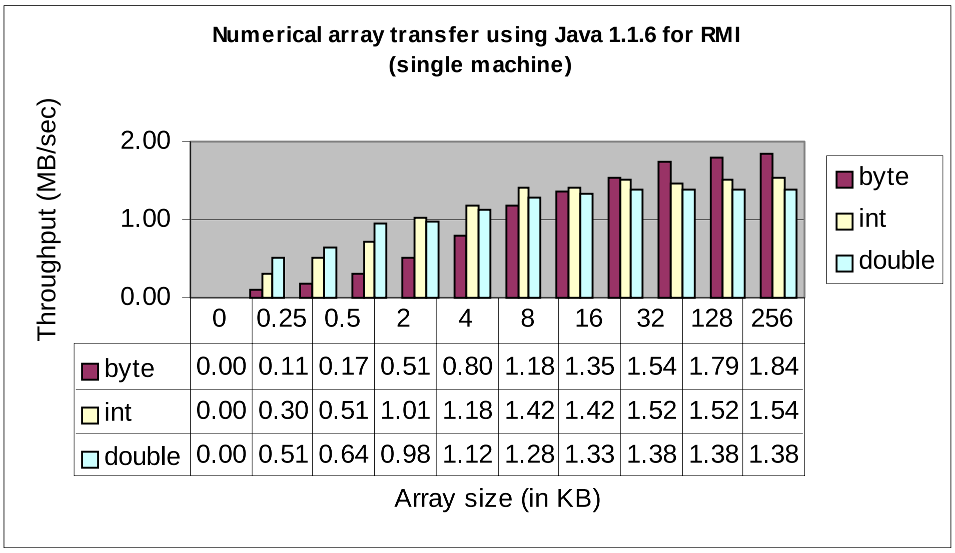
The table below shows the performance of Java RMI on a single machine and on 2 Windows NT machine connected through a 100 Mbps Ethernet LAN.

Table 6-5: Numerical Array Transfer using Java RMI on the same machine

Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	2.2	3.3	3.3	0.00	0.00	0.00
0.25	2.2	3.3	3.8	0.11	0.30	0.51
0.5	2.8	3.8	6.1	0.17	0.51	0.64
1	3.3	5.5	8.2	0.30	0.71	0.95
2	3.8	7.7	16	0.51	1.01	0.98
4	4.9	13.2	28	0.80	1.18	1.12
8	6.6	22	49	1.18	1.42	1.28
16	11.6	44	94	1.35	1.42	1.33
32	20.3	82	181	1.54	1.52	1.38
64	35.7	171	363	1.75	1.46	1.38
128	69.8	330	725	1.79	1.52	1.38
256	135.7	648	1445	1.84	1.54	1.38

Figure 6-3: Numerical Array Transfer using Java RMI on the same machine



	0	0.25	0.5	2	4	8	16	32	128	256
■ byte	0.00	0.11	0.17	0.51	0.80	1.18	1.35	1.54	1.79	1.84
□ int	0.00	0.30	0.51	1.01	1.18	1.42	1.42	1.52	1.52	1.54
□ double	0.00	0.51	0.64	0.98	1.12	1.28	1.33	1.38	1.38	1.38

The graph shows that, transferring the array of byte and int primitive data reaches its peak throughput at the array size of 256 KB, at which its throughput is 1.84 MB/sec and 1.54 MB/sec respectively. For transferring array of double, its transfer rate becomes saturated at 1.38 MB/sec, which corresponds to the array size of 16 KB.

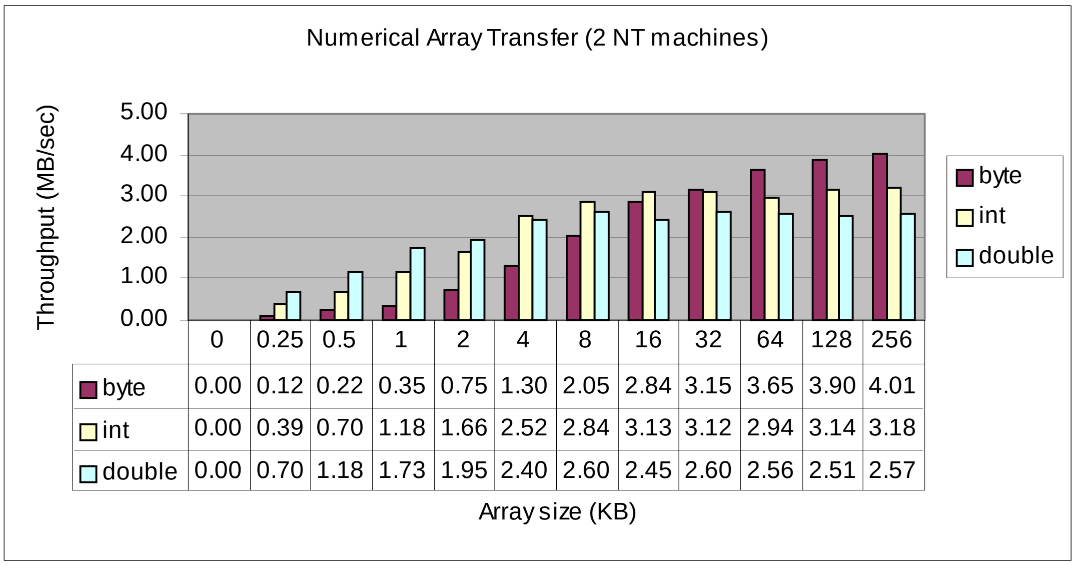
Table 6-6: Numerical Array Transfer using Java RMI on 2 NT machines

Two Windows NT Machines

- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6

Array	Time (msec)			Throughput (MB/sec)		
Size in KB	byte	int	double	byte	int	double
0	2	3.11	2.71	0.00	0.00	0.00
0.25	2.1	2.5	2.8	0.12	0.39	0.70
0.5	2.21	2.81	3.31	0.22	0.70	1.18
1	2.8	3.3	4.51	0.35	1.18	1.73
2	2.61	4.71	8	0.75	1.66	1.95
4	3	6.21	13	1.30	2.52	2.40
8	3.81	11	24	2.05	2.84	2.60
16	5.5	20	51.1	2.84	3.13	2.45
32	9.91	40.1	96.1	3.15	3.12	2.60
64	17.12	85.1	195.2	3.65	2.94	2.56
128	32.04	159.3	398.5	3.90	3.14	2.51
256	62.39	314.4	779.1	4.01	3.18	2.57

Figure 6-4: Numerical Array Transfer using Java RMI on 2 NT machines



	0	0.25	0.5	1	2	4	8	16	32	64	128	256
byte	0.00	0.12	0.22	0.35	0.75	1.30	2.05	2.84	3.15	3.65	3.90	4.01
int	0.00	0.39	0.70	1.18	1.66	2.52	2.84	3.13	3.12	2.94	3.14	3.18
double	0.00	0.70	1.18	1.73	1.95	2.40	2.60	2.45	2.60	2.56	2.51	2.57

The graph shows that, transferring the array of byte and int primitive data reaches its peak throughput at the array size of 256 KB, at which its throughput is 4.01 MB/sec and 3.18 MB/sec respectively. For transferring array of double, its transfer rate reaches its peak at 2.60 MB/sec, which corresponds to the array size of 8KB and 32 KB.

The performance of transferring of large array of primitive data using Java RMI depends heavily on the serialization process which uses lot of CPU resource. This can be seen from the graphs above that the transfer rate across 2 networked machines is more than twice the transfer rate on a single machine. The cost of byte reordering as we will see later on the Comparison Result section is very expensive as compared to the transferring primitive data using raw socket implementation.

6.7 Garbage Collection

Java RMI uses a reference counting garbage collection algorithm similar to Modula-3's Network Object. (See "Network Objects" by Birrel, Nelson and Owicki, Digital Equipment Corporation Systems Research Center Technical Report 115, 1994.)
http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-116.html

RMI has a lease-based garbage collection system. A reference to a remote object is leased for a period of time by the client holding the reference. The lease period starts when the dirty call is received. It is the client's responsibility to renew the leases by making additional dirty calls on the remote references it holds before the leases expire. A remote object is garbage-collected when all its leases expire.

More information about distributed garbage collection in Java RMI can be found in the Java RMI Specification:
http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmi-arch.doc.html#5097

7. HORB

DOT: HORB (Hirano Object Request Broker)

Developed by: Hirano Satoshi of Electrotechnical Laboratory, Japan. Website: http://www.horb.org/

Version tested: HORB version 1.3 beta 1

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - HORBC Compiler, Sun's JDK 1.1.6 Java Compiler
- Java VM:
  - Sun's JDK 1.1.6 with Symantec's JIT built in

HORB is a lightweight Object Request Broker for Java that extends the Java functionality in distributed object oriented computing. Applications of HORB and HORB itself are network portable and transferable, since HORB provides both portability and interoperability among different kinds of OSs. Developing distributed application in HORB is very easy and straightforward. Here are the features supported by current version of HORB.

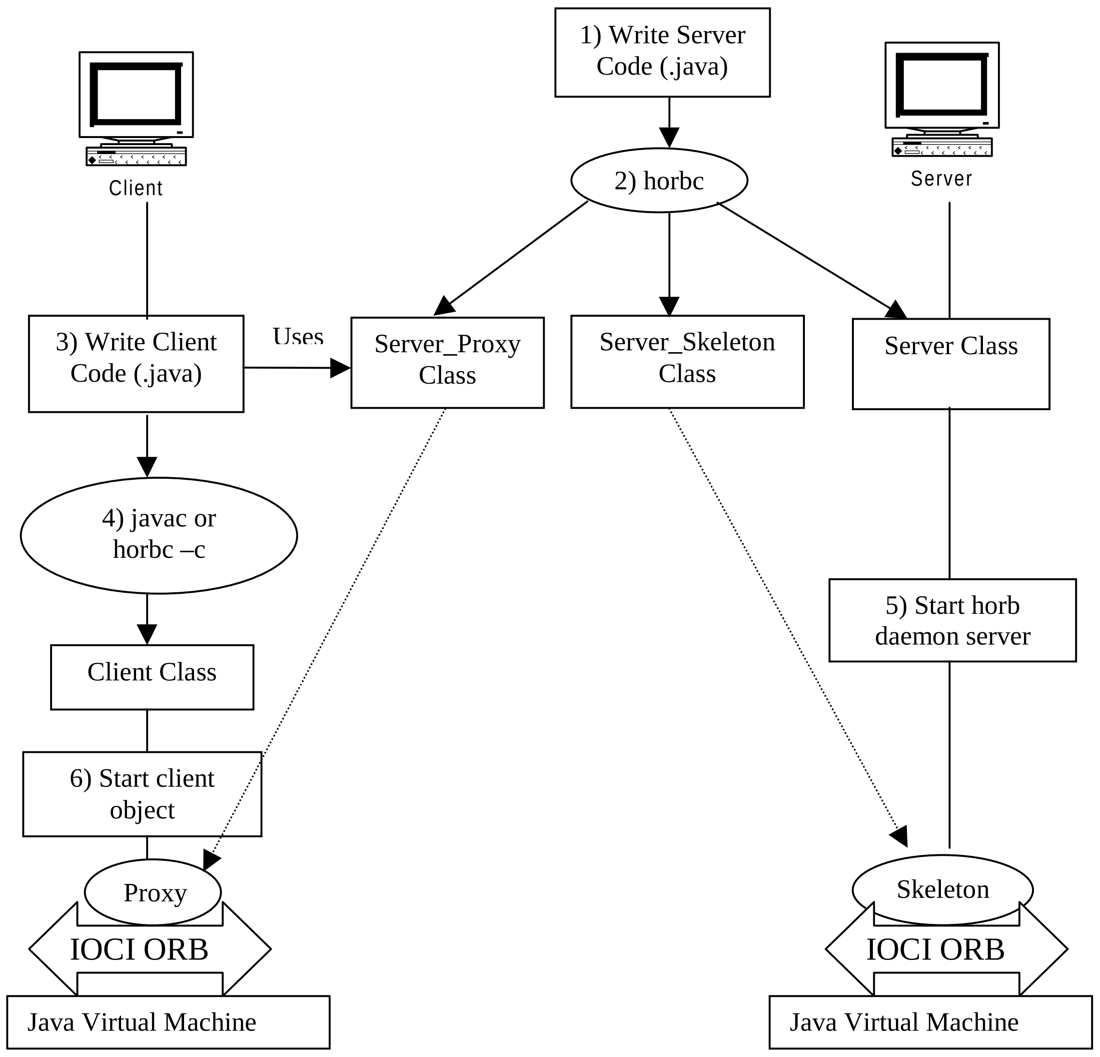
- Dynamic Remote Object Creation
- Remote Object Connection
- Remote Method Call (synchronous or asynchronous)
- Object passing by value (copy) or by reference
- Mobile Agent
- Distributed persistent objects
- Distributed Garbage Collection
- Distributed Access Control lists (built-in security feature of HORB)
- Hook methods
- Distributed Object management
- World Wide Web integration

For more information of HORB, please visit the HORB Web Site:
http://www.horb.org/

7.1 Development process

With HORB we don't need to write an interface for remote object. Here is the figure shows how to write a distributed application in HORB.

<u>Figure 7-1</u>: Distributed Object application development in HORB



As we can see, there are six steps to develop a distributed object application in HORB. The detail description of each is as follows:

1. Write a server code. Here is the Server.java code which is used for our benchmark

```
 // Server.java
 package DOTBenchmark.horb;
 public class Server {
int methodA1(int a1) { return a1; }
int methodA2(int a1, int a2) { return a1; }
int methodA3(int a1, int a2, int a3) { return a1; }
int methodA4(int a1, int a2, int a3, int a4) { return a1; }
```

```
    int methodA5(int a1, int a2, int a3, int a4, int a5) { return a1; }
    int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) { return a1; }

    void methodDA(Data[] dat) { }

    void methodB(byte[] a) {}
    void methodI(int[] ia) {}
    void methodD(double[] da){}
}
```

The code for Server.java define the implementation of the methods that are called by our client program. As we can see, the Server.java has most of the methods as defined in the ServerImpl.java but it is much more elegant. We don't need to add RemoteException to every method signature in our Server.java

2. Compile the Server.java. Next we need to compile the Server.java with the HORBC compiler. From the dos command windows, change to the directory contains Server.java and enter this command from there:

   prompt> horbc –O Server.java

Note. In this benchmark project, we will compile all java code with option O (optimizer turned on). If this command runs successfully, HORBC compiler generates Server.class, Server_Proxy.class and Server_Skeleton.class. The Server.class and the Server_Skeleton.class are needed on the server machine and the Server_Proxy.class is needed on the client machine. The client program will use it and create an instance of Server_proxy which acts as the remote object reference of Server object on the server machine.

3. Write the client code. The client program that is used for benchmarking in this project has almost the same as the client program for the Java RMI version. The only different is the way of obtaining the server object reference. In HORB, the client program creates an instance of Server_Proxy to get the remote object reference to the Server. Here is the code for this:

```
Server_Proxy server  = new Server_Proxy("horb://"+host);
```

4. Compile the client code. We can compile the Client.java using javac compiler or horbc -c compiler. horbc -c –O Client.java means only the Client.class file is generated.
5. Start up the HORB daemon on the server machine. Make sure the Server.class and Server_Skeleton.class are copied on the server machine. From the dos command prompt windows, enter this command:

   prompt> horb –v

   In this project, we want to test the remote object connection operation in which a client program connects to an existing "Server" object on the server machine. We can use "start" option command of horb to startup our server, this is done as follows:

   prompt>horb –start DOTBenchmark.horb.Server server

   This command tells HORB to run the Server as an alias "server" on the server machine.
6. Start up the Client code. Invoke the client program using Java interpreter

   prompt>java DOTBenchmark.horb.Client 100 *serverhostname*

This command runs the client program with an input iteration argument of 100 and a server host name as *serverhostname*.

7.2 Remote Object Connection

HORB supports both remote object connection and remote object creation operation. The remote object connection will be discussed right after this section. In the remote object connection case, multiple clients can be served by one remote server object. As a new connection is made from the client to the remote server object, a corresponding thread is attached to the client.

In this project, our client program creates a remote server object at a particular host, which is passed in from the command line, then we create a new server_proxy instance which acts as a reference to that remote object on the server. I measure the time it takes a client object to perform this operation and the average time of 100 iteration is collected.
Here is the part of code in the Client.java that we used for this test.

```
// iter is the iteration input from the command line
Server_Proxy  servers[] = new Server_Proxy[iter];

HorbURL url = new HorbURL(host,"ExistingServer"); // create a new HORB'url at this particular host with
an object
                                                  //ID as "ExistingServer"

Server_Proxy junk = new Server_Proxy(url);  // ignore the first one; just for warming up

// start benchmark test
System.gc();
double start = System.currentTimeMillis();
for (i = 0; i < iter; i++) {
      servers[i] = new Server_Proxy(url);
}
double time = System.currentTimeMillis() - start;
double elapse = (double)(time)/(double)(iter);
System.out.println("bind to server(): "+elapse+" msec");

// release connection we don't need
for( i = 0; i < iter; i++) {
          servers[i]._release();
}
```

As stated above, our existing server had been started up beforehand at the server machine as follows:

prompt>horb -start DOTBenchmark.horb.Server ExistingServer

The table below shows the test of running our client and server on the same machine and on two different machines.

Table 7-1: Remote Object Connection in HORB

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1
16 msec	19.03 msec

The test shows that connecting to an existing object using 2 machines across an idle network takes a longer time than on a local machine.
Compare this result with Java RMI of JDK 1.1.6, connecting to an existing object in HORB takes a bit longer for both situations.

7.3 Remote Object Creation

The remote object creation is different from the remote object connection in that the client dynamically creates a new remote object. This model solves the problem, which may arise in the remote connection model when there is a shortage on sharing data object. In the remote creation model, a new object can be created to serve each of the clients, this makes programming easier than the remote object connection model. Also, dynamically create object on demand is more efficient because object only be created when needed.

In HORB, to create a new remote object on the server, it takes two steps:

- First, it needs to connect to the server, a reference to the remote object is returned
- Second, a constructor of the remote object is called from the remote object reference. This call to the constructor is optional because if the client object does not make that explicit call, the default constructor that has a null argument will be called implicitly by the proxy object.

Here is the code that is used for the test of this operation:

```
// iter is the iteration input from the command line
Server_Proxy  servers[] = new Server_Proxy[iter];
double start = System.currentTimeMillis();
for (i = 0; i < iter; i++) {
     servers[i] = new Server_Proxy(url);
     servers[i].Server();
}
time = System.currentTimeMillis() - start;
for( i = 0; i < iter; i++)
     servers[i]._release();
 elapse = (double)(time)/(double)(iter);
System.out.println("create server(): "+elapse+" msec");
```

The table below shows the average elapsed time of 100 iterations. Again the test is run on a single machine and on 2 NT machines.

Table 7-2: Remote Object Creation in HORB

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1
11 msec	17.03 msec

The tests show that:

- Remote object connection on the same machine is faster than remote object connection on networked machines
- Paradoxically, the operation of remote object creation times is shorter than the remote object connection for both situations. This seems that explicitly calling the default constructor is better than relying on the implicitly calling the constructor.
- Remote object creation operation in HORB is nearly three times faster than the remote object creation operation in Java RMI.

7.4 Remote Method Call

This operation evaluates how best HORB's proxy and skeleton marshalling and unmarshalling message exchanged between client and server. Marshalling and unmarshalling is done via CommonIOCI class. IOCI is Inter-Object Communication Interface which defines APIs between the lower transport layer and the upper application layer (i.e. HORB).

A new HORB serialization patch is just released on the July 1998 this year by Luis F. G. Sarmenta (lfgs@cag.lcs.mit.edu) which shows some improvements in the marshalling and unmarshalling process in HORB. It allows HORB to use Java built-in Serialization System. According to the author of the patch, it has the following benefits and enhancements include:

- Removes the need to run `horbc` on objects that are only used as parameters or return values of remote functions. Enables HORB applications to use built-in objects that `implement Serializable` (e.g., `java.util.Vector`). Backwards-compatible
- works with non-Serializable data and mixed-mode data (requires `horbc`)
- new clients can connect to old servers
- new servers can accept connections from old clients
- automatically uses built-in Serialization when available; uses old HORB serialization otherwise

This serialization patch is available to download at the HORB's web site: http://www.horb.org

In this project, I test the operation of HORB that uses the default HORB's serialization along with the new patch. The remote method call is exactly the same as we did in Java RMI. Here is the code for this:

```
Server_Proxy server = new Server_Proxy("horb://"+host); // new server for the test
// rmc with one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}
 time = System.currentTimeMillis() – start;
 elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// rmc with two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA2(100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

// rmc with three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA3(100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");
```

```
// rmc with four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA4(100, 100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// rmc with five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
 System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

// rmc with six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA6(100, 100, 100, 100, 100, 100);
 }
 time = System.currentTimeMillis() – start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
```

And we just need to add this line

```
HORB.useSerialization();
```

just before we create Server_Proxy

```
Server_Proxy server = new Server_Proxy("horb://"+host);
```

Table 7-3: Remote Method Call in HORB

Remote Method Call with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1, HORB 1.3b1 patch (p) (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1, HORB 1.3b1 patch (p) (Unit time in msec)					
Args	1	2	3	4	5	6	1	2	3	4	5	6
Horb	1.25	1.04	1.04	2.2	1.25	1.25	1.2	1.2	1.0	0.9	1.0	1.0
Horb(p)	1.5	1.25	1.5	1.5	1.25	1.5	1.0	1.1	1.1	1.0	1.1	1.1

The tests show that:

- Performance of remote method calls on the same machine take longer time than calls over network. This again shows that JVM is a very CPU intensive.
- On the same machine, the HORB's patch version which use JDK 1.1 Serialization Process takes longer times in all calls except at the call with 4 arguments, this result might be an outlier. So, the default HORB's serialization (i.e. CommonIOCI) shows better performance in this respect.
- However, for the test over an idle 100Base-T Ethernet shows that the patch version is better sometimes! This is caused by the high computational intensity of the marshalling and unmarshalling operations, which consumes CPU resource is diverted on two machines.

7.5 Object Data Transfer

HORB supports object passing by value or by reference. This feature is similar to the way that rules how an object should be passed by copy or by reference in Java RMI. This means that if the object for transferring is a local object, then passing by value (copy) is used, if the object for transferring is a remote object, then passing by reference is used. In this case a copy of the proxy object itself is passed. HORB can use its own serialization process or Java serialization to pass object by value. HORB's object serialization is called serializer objects. What this means is that to make an object that can be serialized and transferred by HORB, we need to compile that object with HORBC compiler. This process will generate a proxy for the object that includes a serialized method to be called at runtime to carry out serialization for the object. When object is passed by copy, all instance variables in the range possible to attain recursively within the transferring object are copied. For security reason, HORB allows transfer of only "non privates" instance variables. This also means that "protected" field can also be transferred. If a class contains a private instance variable, the HORBC compiler will give warning message when the class is compiled. As in Java RMI, a transient instance field is not transferred.

In HORB, any class transferred remotely must have a Proxy class both on client and on server machine. This means, we need to compile that class with HORBC compiler to generate proxy and skeleton class for it. Also because of the internal mechanism of HORB, the class **must** define a null constructor which is called by the server when it arrives on the server.

If an object was cast to superclass is passed, then the original object itself is passed, not the superclass. For e.g., if we have class B extends class A, and we have

```
A a = new B();
foo.operate(a);
```

then an instance of class B is passed, not A.

Sometimes, we don't want to pass variables in superclasses, this can be done using HORBC compiler with an option of -ignoresuper that will tell HORBC to ignore the superclasses.

In this project, I evaluate the performance of passing a simple object using horb and without using cast operation. What this means is that we call the default constructor of the passing object, but this causes the java.lang.Object's default constructor is called because we didn't define a null constructor for the passing object. What we evaluated here at the moment is just the passing object mechanism, not the object itself. However, evaluation the performance of passing object with casting, passing complex objects will be carried out if time allows.

For the moment, here is the code for our passing object called Data

```
// Data.java
public class Data {
   int a;
}
```

This Data.java class is exactly the same as the Data.java in the Java RMI test above, except we don't explicitly extends the java.io.Serializable. This serialization process is actually taken care by the HORB's proxy object.

Next, use HORBC compiler to compile Data.java to generate Data_Proxy and Data_Skeleton. As before, the proxy object is copied on the client machine and the skeleton object is copied on the server machine.

The client calls the transfer the array of object Data is described in the code bellows:

```
int[] sizes = {1, 10, 20, 30, 40, 50};
for (int s = 0; s < 6; s++) {
    Data[] data = new Data[sizes[s]];
    for (j = 0; j < sizes[s]; j++) {
        data[j] = new Data();
    }
    System.gc();
    double start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        server.methodDA(data);
    }
    double time = System.currentTimeMillis() - start;
    double elapse = (double)(time)/(double)(iter);
    System.out.println("void server.methodDA(Data["+sizes[s]+"]): "+elapse+" msec");
}
```

For complete operation in our Client.java, please refer to the appendix section

Table 7-4: Transfer object by value in HORB

Object Data Transfer with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1, HORB 1.3b1 patch (p) (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: HORB 1.3b1, HORB 1.3b1 patch (p) (Unit time in msec)					
Number of objects	**1**	**10**	**20**	**30**	**40**	**50**	**1**	**10**	**20**	**30**	**40**	**50**
Horb	4.58	4.58	5.63	9.17	11.25	12.71	2.81	5.91	11.11	15.83	21.63	29.34
Horb(p)	2.7	2.75	4.00	5.50	8.28	9.75	2.81	3.21	7.31	6.91	8.71	10.42

As we can see from the table above:

- The time for transferring an array of object is increased with the array size
- The HORB patch version which use Java built in Serialization mechanism shows very good performance compare to the HORB's default serialization process
- For the test on 2 networked machines, as the number of object increase 50 times, the time for the original of HORB increase almost 10 times whereas the HORB patch version increases just only 5 times!

7.6 Numerical Array Transfer

In this test, I evaluate the performance of the HORB with Serialization patch version in transferring the array of primitive data, namely "byte", "int" and "double". The array size is determined at runtime. The test result shows the time in millisecond (msec) corresponding to transferring an array size measured in KB. For reference purpose, I also calculate the throughput, which shows the transfer rate in MB/sec. Tables below show the test on a single machine as well as on two networked machines connected through an idle 100Base-T Ethernet network.

The code for measuring this operation is similar to the code that is used in the Java RMI case, I listed it below just for completeness.

```
// byte[]     bufb; int[]  bufi; double[] bufd;
// int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4, 4096*8, 4096*16, 4096*32, 4096*64 };
        // to send byte array;
          for ( j = 0; j < array_sizes.length; j++) {
            bufb = new byte[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodB(bufb);
            }
            time = System.currentTimeMillis() – start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j])/elapse) + " KB/sec");
          }

          // to send int array
          for ( j = 0; j < array_sizes.length; ; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodI(bufi);
            }
            time = System.currentTimeMillis() – start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
        }

        // to send double array
        for ( j = 0; j < array_sizes.length; ; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodD(bufd);
        }
        time = System.currentTimeMillis() – start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
    }
```

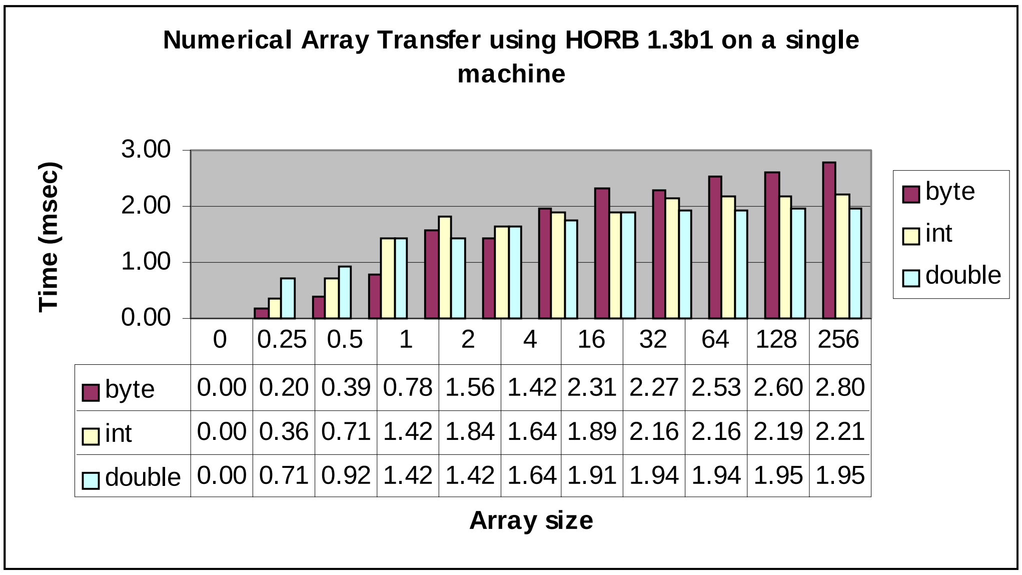
Table 7-5: Numerical Array Transfer in HORB

Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default

Software: Java JDK 1.1.6, HORB 1.3b1

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	9.5	2.75	2.75	0.00	0.00	0.00
0.25	1.25	2.75	2.75	0.20	0.36	0.71
0.5	1.25	2.75	4.25	0.39	0.71	0.92
1	1.25	2.75	5.5	0.78	1.42	1.42
2	1.25	4.25	11	1.56	1.84	1.42
4	2.75	9.5	19	1.42	1.64	1.64
8	4	16.5	36	1.95	1.89	1.74
16	6.75	33	65.5	2.31	1.89	1.91
32	13.75	58	129	2.27	2.16	1.94
64	24.75	116	258	2.53	2.16	1.94
128	48	228	513.5	2.60	2.19	1.95
256	89.25	453	1027	2.80	2.21	1.95

Figure 7-2: Numerical Array Transfer in HORB on the same machine



	0	0.25	0.5	1	2	4	16	32	64	128	256
■ byte	0.00	0.20	0.39	0.78	1.56	1.42	2.31	2.27	2.53	2.60	2.80
□ int	0.00	0.36	0.71	1.42	1.84	1.64	1.89	2.16	2.16	2.19	2.21
□ double	0.00	0.71	0.92	1.42	1.42	1.64	1.91	1.94	1.94	1.95	1.95

The results show that:

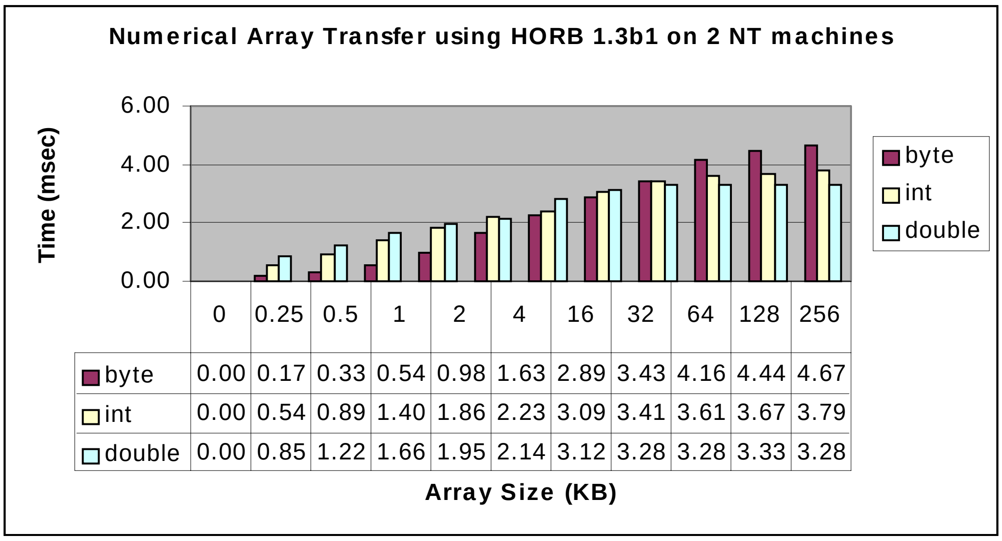
- The transfer rate for "byte" and "int" increase with the array size, which reaches its peak at the array size of 256 KB, at which it has a value of 2.80 MB/sec and 2.21 MB/sec respectively.
- The transfer rate of "double" array reaches becomes saturated at the array size of 128 KB, which has a value of 1.95 MB/sec.
- HORB's proxy and skeleton which perform the marshalling and unmarshalling job is slightly better than Java RMI in this respect.

Table 7-6: Numerical Array Transfer in HORB on 2 NT machines

Two Windows NT Machines
- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6, HORB 1.3b1

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	1.5	2	2.11	0.00	0.00	0.00
0.25	1.4	1.81	2.3	0.17	0.54	0.85
0.5	1.5	2.2	3.21	0.33	0.89	1.22
1	1.81	2.8	4.7	0.54	1.40	1.66
2	2	4.21	8.02	0.98	1.86	1.95
4	2.4	7.01	14.62	1.63	2.23	2.14
8	3.41	13.22	22.24	2.29	2.36	2.81
16	5.41	20.24	40.06	2.89	3.09	3.12
32	9.11	36.64	76.3	3.43	3.41	3.28
64	15.02	69.3	152.22	4.16	3.61	3.28
128	28.14	136.2	300.64	4.44	3.67	3.33
256	53.58	264	610.28	4.67	3.79	3.28

Figure 7-3: Numerical Array Transfer in HORB on 2 NT machines



	0	0.25	0.5	1	2	4	16	32	64	128	256
■ byte	0.00	0.17	0.33	0.54	0.98	1.63	2.89	3.43	4.16	4.44	4.67
□ int	0.00	0.54	0.89	1.40	1.86	2.23	3.09	3.41	3.61	3.67	3.79
□ double	0.00	0.85	1.22	1.66	1.95	2.14	3.12	3.28	3.28	3.33	3.28

The result of the same test over an idle 100Base-T Ethernet shows that:

- The transfer rate increases with the array size, which reaches its peak at the array size of 256 KB in the case of "byte" and "int", at which it has a value of 2.80 MB/sec and 2.21 MB/sec respectively. In the case of "double" array, the peak transfer rate is 3.33 MB/sec at the array size of 128 KB.
- The transfer rate of "double" array reaches becomes saturated at the array size of 128 KB, which is the same for the case in a single machine operation. However, the throughput is about 2 times higher.
- Again, this confirms the superior of marshalling and unmarshalling process in HORB's proxy and skeleton to Java RMI in this respect.

7.7 Garbage Collection

HORB objects (except HORB's Daemon Objects) are automatically deleted from the system once all of the remote object references to the object are no longer used, this is done through the reference counting mechanism.

In HORB, a new TCP/IP connection and a thread of the server object is created when a new remote object reference is created. However, it is possible to explicitly release a connection to a remote object reference using _release() method of proxy objects.

8. Voyager

DOT: Voyager

Developed by: ObjectSpace. Website: http://www.objectspace.com

Version tested: Voyager 2.0.0

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - Sun's JDK 1.1.6 Java Compiler
- Java VM:
  - Sun's JDK 1.1.6 with Symantec's JIT built in

Voyager is a 100% Java distributed computing platform that can be used to produce high-impact distributed systems quickly. Voyager was developed by ObjectSpace, a company that was founded about 6 years ago with the charter of providing support for the development and deployment of distributed object technology solutions. On mid September 1998 this year, ObjectSpace released Voyager 2.0.0. According to ObjectSpace, there are currently about 10,000 companies worldwide using Voyager. Here are the main features that currently supports in Voyager 2.0.0:

- **Remote-Enabling a Class.** Java classes are remote-enabled classes at runtime
- **Dynamic Remote Object Construction.** A remote instance of any class can be created and a proxy to the newly created object is obtained. The proxy class is generated dynamically if it doesn't exist.
- **Dynamic Class Loading.** Classes can be dynamically loaded from one or more locations when necessary.
- **Remote Method Invocation.** Objects can be passed by value (copy) or by reference, which uses the default standard Java serialization mechanism.
- **Exception Handling.** If a remote exception occurs, it is caught at the remote site and rethrown locally.
- **Distributed Garbage Collection.** Reclaims objects that are no longer referred to by clients.

- **Dynamic Aggregation.** Adds secondary objects to a primary object at runtime.
- **Mobile Agent.** Any Serializable objects can move between programs at runtime.
- **Support CORBA.** There is full native support for IDL, IIOP, and bi-directional IDL and Java translation. No stub generators or helper classes are required.
- **Activation.** The activation framework allows objects to be persisted to any kind of database and automatically re-activated in the case that the program is restarted.
- **Naming Service.** The naming service provides a single, simple interface that unifies the many commercially available naming services.
- **Multicast.** Message can be delivered to a distributed group of objects without requiring the sender or receiver to be modified in any way.
- **Publish-Subscribe.** Java event can be published to a distributed group of subscribers.
- **Security.** Hooks can be used to install custom socket such as SSL (Secure Socket Layer)

For more information about Voyager, please visit the web site at:
http://www.objectspace.com/

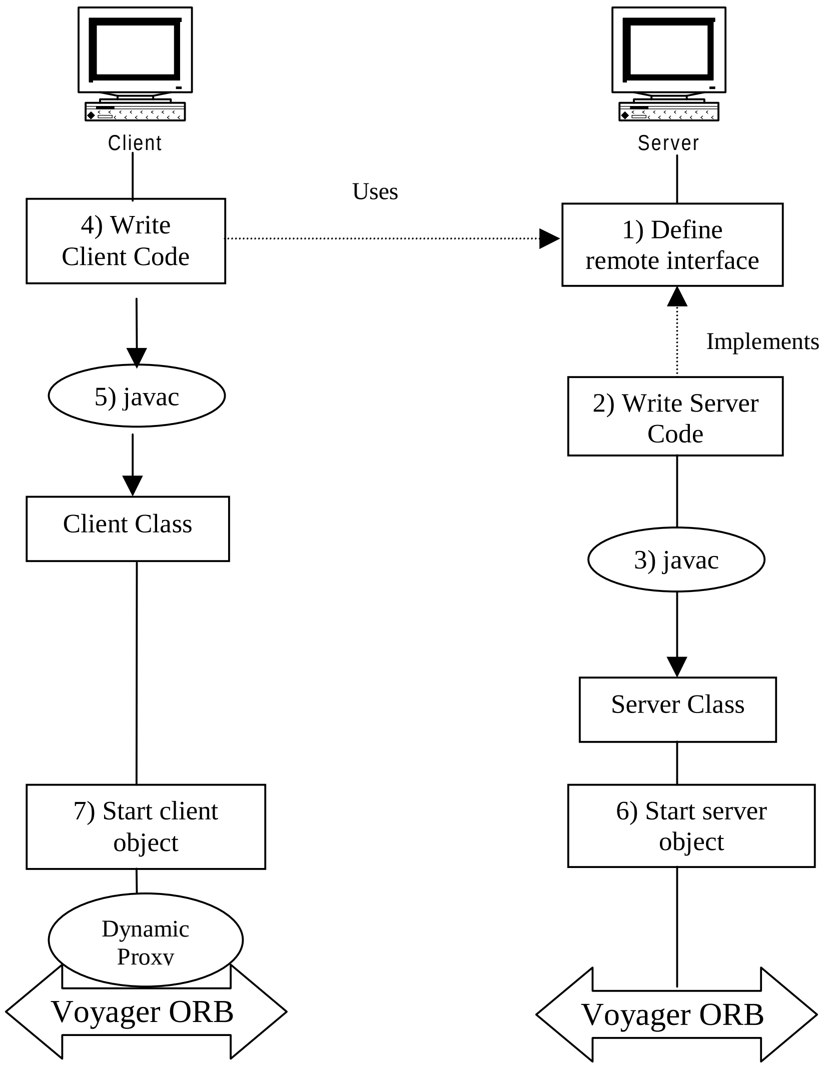
8.1 Development Process

The figure below, I describe steps for developing a distributed object application for the test used in this project. In Voyager, it is possible to turn any classes to be used remotely by running Voyager's **igen** utility on that particular although that class does not implement an appropriate interface. For e.g., to generate the default interface for java.util.Vector, enter this command at the command line:

```
prompt> igen java.util.Vector
```

What is different in Voyager to most DOTs is that Voyager generates client code only. There is no server skeleton/stub on the server side.

Figure 8-1: Distributed object development process in Voyager



1. Write a remote interface. Define a Java interface for remote object. Here's the code:

```
    // IServer.java
    package DOTBenchmark.voy20;

    public interface IServer {
      int methodA1(int a1);
      int methodA2(int a1, int a2);
      int methodA3(int a1, int a2, int a3);
      int methodA4(int a1, int a2, int a3, int a4);
      int methodA5(int a1, int a2, int a3, int a4, int a5);
      int methodA6(int a1, int a2, int a3, int a4, int a5, int a6);
      void methodDA(Data[] dat);
      void methodI(int[] ia);
      void methodD(double[] da);
  }
```

2. Write the Server code. Here I write a server application to start up the voyager server. We can either startup the voyager server from the command line at a specific port or call it from a standalone program. For e.g., enter this command at the dos prompt will startup the voyager at the port 8000

prompt> voyager 8000

Here's the code for our Server.java used for our benchmark. The server program will start up the voyager server when it runs.

```
// Server.java
package DOTBenchmark.voy20;


import com.objectspace.voyager.*;
import java.io.*;

public class Server implements IServer {

  static String serverName;
  static {
      try {
              serverName = java.net.InetAddress.getLocalHost().getHostName();
      } catch (Exception e) {
              serverName = "localhost";
      }
      System.out.println("Server name: "+serverName);
  }

  public Server () {
      super();
  }

  public int methodA1(int a1) { return a1; }
  public int methodA2(int a1, int a2) { return a1; }
  public int methodA3(int a1, int a2, int a3) { return a1; }
  public int methodA4(int a1, int a2, int a3, int a4) { return a1; }
  public int methodA5(int a1, int a2, int a3, int a4, int a5) { return a1; }
  public int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) { return a1; }
  public void methodDA(Data[] dat) { }
  public void methodB(byte[] a) {}
  public void methodI(int[] ia) {}
  public void methodD(double[] da){}

  public static void main(String argv[]) throws Exception {

      Voyager.startup("//" + serverName + ":8000");
                      String serverclass = Server.class.getName(); // get the Server class name
      IServer server = (IServer) Factory.create(serverclass,"//" + serverName + ":8000");
      Namespace.rebind("VServer",server);
      System.out.println("bind done");
      System.out.println("Server is listening at " + serverName + ":8000/VServer");
  }
}
```

As we can see in our server code, it has a static method to locate our server host address, which will be used later on to start up Voyager server services and to register our server with a name. In the static main method, we startup the Voyager services at the server host located previously.

```
Voyager.startup("//" + serverName + ":8000"); // startup Voyager server at port 8000 of this serverName host
```

Then we create a proxy for our Server object by using the voyager Factory.create method. This is the proxy to our local server object which will be used for remote object connection benchmark later on.

Here we call our server object with a name of "Vserver". To bind a name to our Server object, Voyager provides a bind()/rebind() which is somewhat similar in the Java RMI to bind the name to objects for later lookup. In our code, we bind our Server object to a name of "Vserver" as described in the code of our static main method:

```
Namespace.rebind("VServer",server);
```

3. Compile our Server code. We then compile our Server.java using our favorite Java compiler.
   prompt> javac -O Server.java

   As usual, all the java code will be compiled with optimizer option turned on.

4. Write a Client code. We write a client code for our test. As a requirement in Voyager program, Voyager service needs to be running before any services can be used. This can be done by calling Voyager.startup method with a null argument.

```
Voyager.startup(); // startup as client
```

5. Compile Client code. Use Javac to compile our Client.java (with optimizer turned on)

6. Startup Server object. Enter this command from the command line:
   prompt>java DOTBenchmark.voy20.Server

   (Note that our Server is under the DOTBenchmark.voy20 package statement)
7. Startup Client object. Enter this command from the command line:

   prompt> java DOTBenchmark.voy20.Client 100 *serverhostname*

For a complete Voyager's Client and Server program, please refer to the appendix.

8.2 Remote Object Connection

Voyager has its own lookup method just as we use the `Naming.lookup` in Java RMI. To get a proxy object of an existing remote object with a particular name on the server, we use `Namespace.lookup` method as is specified in the codes below:

```
Voyager.startup(); // startup as client
IServer server = null;
System.out.println("looking up...");
server = (IServer)Namespace.lookup("//"+host+":8000/VServer");
System.out.println("done");
```

Here, the client program tries to locate a proxy of the remote object named VServer at the port 8000 of this particular host. The code used for our benchmark in this case is:

```
System.gc();
start = System.currentTimeMillis();
for (i = 0; i < iter; i++) {
    server = (IServer)Namespace.lookup("//"+host+":8000/VServer");
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("Remote object connect: Namespace.lookup(): "+elapse+"msec");
```

The table below shows the performance for remote object connection using Voyager 2.0.0 on a single machine and on 2 NT machines.

Table 8-1: Remote Object Connection in Voyager

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0
12.60 msec	15.02 msec

The result shows that Voyager's lookup mechanism has very good performance. Voyager's locating remote object has better performance than Java RMI and HORB for the operation on the same machine and better than HORB but a little worse than Java RMI for operation on 2 networked NT machines.

8.3 Remote Object Creation

Voyager uses its own method **Factory.create()** to create an remote object at a specified location. This method returns a proxy to the newly created object and creates the proxy class dynamically if it does not already exist. There are several variations of **create(),** depending on whether the object is to be created locally and whether the class's constructor takes arguments. For further details on this method, please refer to the Voyager Core Technology Guide 2.0.

Here's the code that we use for Remote Object Creation operation in Voyager.

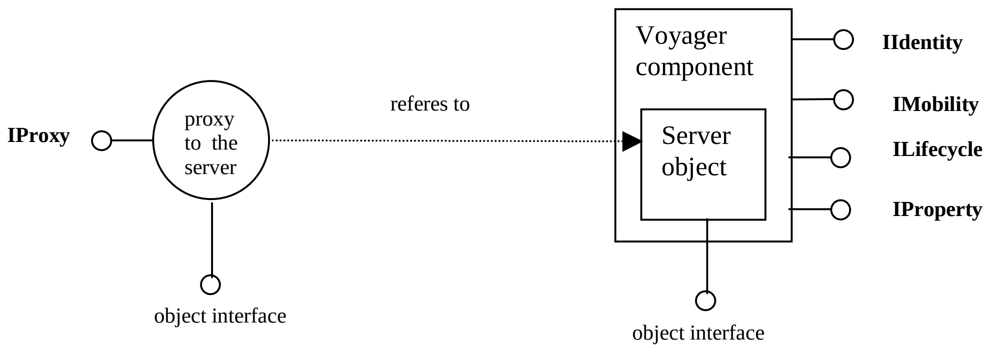
```
// Remote object creation bench
System.gc();
start = System.currentTimeMillis();
for (i = 0; i < iter; i++) {
    // create a remote Server object
    servers[i] = (IServer) Factory.create("DOTBenchmark.voy20.Server", "//"+host+":8000" );
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("create server(): "+elapse+" msec");
```

The name of the class that we want to create must always be fully qualified, i.e. DOTBenchmark.voy20.Server as above. This argument can also be obtained using the code like this:

```
String serverclass = Server.class.getName();
```

So, when the object instance and its proxy are created, Voyager wraps the object in a Voyager component which can be depicted as below:

Figure 8-2: Voyager component model



The value-added interfaces (in boldface) give programmers access to Voyager services to facilitate distributed system developed in Voyager.

Voyager program also provides dynamic class loading. A Voyager program attempts to load new classes according to the following sequence:

1. Search the CLASSPATH
2. Search the installed resource loaders from highest priority to lowest priority.

By default, a Voyager program has a single pre-installed ProxyResourceLoader at priority level 5. This loader can dynamically generate and load a proxy class from its original class.

To enable a program to load resources from remote sources such as web server or a database, we can use the resource loaders by calling:

```
VoyagerClassLoader.addResourceLoader();
```

For e.g. to enable class loading from a specific URL, add an URLResourceLoader constructed on an URL as described below:

```
VoyagerClassLoader.addURLResource(http://localhost:9000/);
```

It is possible to make Voyager acts as a HTTP server class loader, which serve other programs to load any resource that a program needs. To enable HTTP in a Voyager program, invoke the command ClassManager.enableResourceServer(). By default this feature is disabled for security reason.

Now, let's have a look on the performance of Voyager in creating a remote object. The table below shows the result of running the test on a machine and 2 NT machines.

Table 8-2: Remote Object Creation in Voyager

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0
6.60 msec	8.01 msec

The test shows that:

- Create a remote object in Voyager is very fast, in fact it is the fastest in comparing to all other DOT package evaluated in this project.
- Although Voyager dynamically creates proxy and add value-added interfaces to an instance of remote object on the remote machine, it does not slow down it object creation process, i.e. its implementation in creating remote object is very efficient.

8.4 Remote Method Call

In this operation, we evaluate the performance of Remote Method Call with variable number of arguments using Voyager. When a Voyager proxy object receives a message from the client program, it executes according to the following rules:

- If the destination object is in the same program, it message is treated as a normal local call, the argument is not serialized or copied
- If the destination object is in a different program, the arguments and the return value is sent across the network. This can be sent in two different styles:
  - If the argument of the method call implements com.objectspace.voyager.IRemote or java.rmi.Remote, the argument is pass by reference, i.e. a proxy to the argument is sent
  - Otherwise, the argument is pass by value, i.e. a copy of the argument is sent using standard Java Serialization mechanism.

If a remote method throws an exception, it is caught and re-thrown in the local program. Voyager has an exception handling policy that allows us to select between checked and unchecked exception. If we want the checked exception, explicitly add "throws java.rmi.RemoteException" to every method in an interface.

Here's the code that used for testing Remote Method Call with variable arguments with Voyager. After the proxy is created, the operation code is exactly the same as we used for Java RMI and HORB. For a complete version of Voyager's Client program, please refer to the appendix.

```
// newly created server for next bench
Iserver server = (IServer) Namespace.lookup("//"+host+":8000/VServer");

// load class files explicitly
// ignore this just WARM UP
System.gc();
 for (i = iter; i > 0; --i) {
     x = server.methodA1(100);
}

// remote method benchmark
// one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA1(100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA2(100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

//  three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA3(100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");
```

```
// four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA4(100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

// six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
     x = server.methodA6(100, 100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
```

Table 8-3: Remote Method Call in Voyager

Remote Method Call with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0 (Unit time in msec)					
Args	1	2	3	4	5	6	1	2	3	4	5	6
Voyager 2.0.0	1.6	1.6	1.7	1.7	1.7	2.2	1.5	1.6	1.7	2.1	1.7	1.7

The result shows the average of 100 operations on a single machine and on two NT machines. We can see that the performance doesn't change much with the number of argument calls, except at the 6 arguments on local machine and at the 4 arguments on the 2 machine. It is not clear why this is the case, probably this is an outlier. However, comparing to HORB and Java RMI in this operation, Voyager shows better performance than Java RMI but it is slower than HORB. We can see this at the comparison section later on.

8.5 Object Data Transfer

As stated above, in Voyager, arguments of remote message are copied using the standard Java Serialization mechanism before they are sent to a different program. That means the rules applies to transfer objects in Java RMI also applies to the transfer object in Voyager. What is different here is that Voyager adds more services to the object transfer, for e.g. the object transferred can be further move to a new program, this is call object mobility.

The table below shows the performance of transferring array of Data which encapsulates a "int" field on a single machine and on 2 networked machines.

Table 8-4: Transfer object by value in Voyager

Object Data Transfer with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Voyager 2.0.0 (Unit time in msec)					
Number of objects	**1**	**10**	**20**	**30**	**40**	**50**	**1**	**10**	**20**	**30**	**40**	**50**
Voyager 2.0.0	2.80	3.80	5.00	6.60	7.20	8.80	3.01	5.11	6.11	8.71	10.71	12.82

The test shows that:

- Obviously, the time increases with the array size of data transferred.
- Operations on a single machine are faster than operations on 2 networked machines. This indicates that voyager provides a good marshalling and unmarshalling process, the speed depends mostly on the network connection bandwidth.

8.6 Numerical Array Transfer

Just as we did for previous DOTs in evaluating the numerical array transfer operation, we evaluate the operations on a single machine as well as on two NT machines. The table below shows the benchmark for Voyager in this respect.

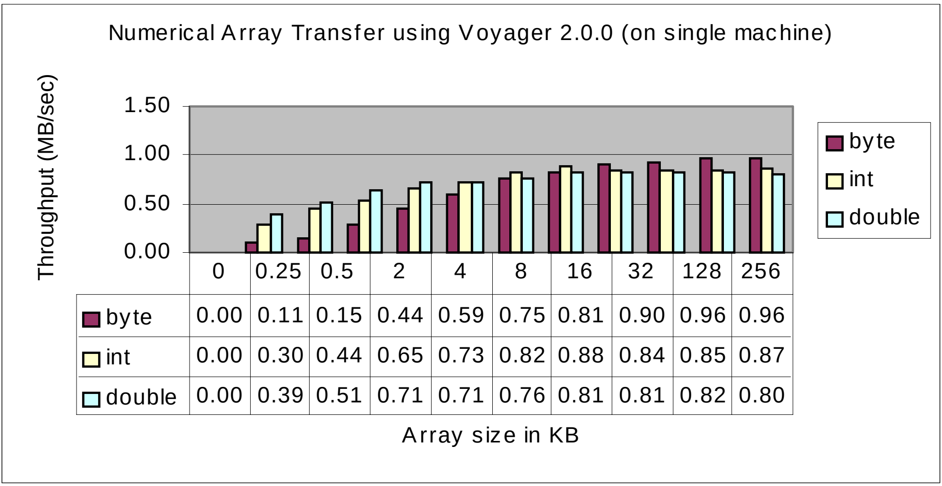
Table 8-5: Numerical Array Transfer in Voyager on the same machine

Same Machine

- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6, Voyager 2.0.0

Array Size in KB	**Time (msec)**			**Throughput (MB/sec)**		
	byte	**int**	**double**	**byte**	**int**	**double**
0	2.7	3.3	2.7	0.00	0.00	0.00
0.25	2.2	3.3	5	0.11	0.30	0.39
0.5	3.3	4.4	7.7	0.15	0.44	0.51
1	3.3	7.2	12.1	0.30	0.54	0.65
2	4.4	12	22	0.44	0.65	0.71
4	6.6	21.5	44	0.59	0.73	0.71
8	10.4	38	82	0.75	0.82	0.76
16	19.2	71	154	0.81	0.88	0.81
32	34.6	149	307	0.90	0.84	0.81
64	67	297	604	0.93	0.84	0.83
128	130	588	1225	0.96	0.85	0.82
256	260	1153	2499	0.96	0.87	0.80

Figure 8-3: Numerical Array Transfer in Voyager on the same machine



	0	0.25	0.5	2	4	8	16	32	128	256
byte	0.00	0.11	0.15	0.44	0.59	0.75	0.81	0.90	0.96	0.96
int	0.00	0.30	0.44	0.65	0.73	0.82	0.88	0.84	0.85	0.87
double	0.00	0.39	0.51	0.71	0.71	0.76	0.81	0.81	0.82	0.80

The test shows that:

- byte array transfer reaches its peak at 0.96 MB/sec for the array size of 256 KB, int array at 0.88 at array size 16 KB, double array at 0.82 at the array size 128 KB.

- Transferring numerical array has roughly the same peak throughput for byte, int and double.

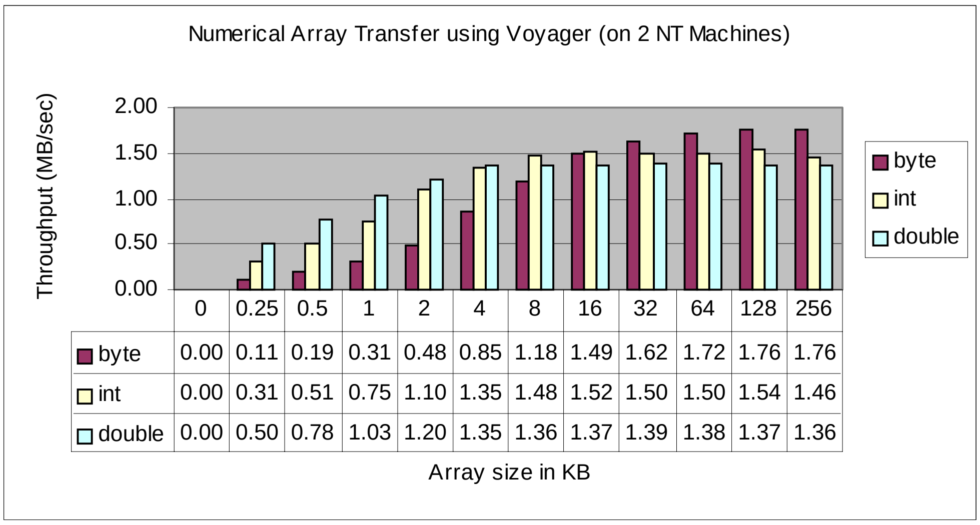
Table 8-6: Numerical Array Transfer in Voyager on 2 NT machines

Two Windows NT Machines
- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6, Voyager 2.0.0

(Unit time in msec)

Array	Time (msec)			Throughput (MB/sec)		
Size in KB	byte	int	double	byte	int	double
0	3.61	3.31	3.01	0.00	0.00	0.00
0.25	2.3	3.2	3.9	0.11	0.31	0.50
0.5	2.61	3.81	5.01	0.19	0.51	0.78
1	3.2	5.21	7.61	0.31	0.75	1.03
2	4.11	7.11	13	0.48	1.10	1.20
4	4.6	11.61	23.1	0.85	1.35	1.35
8	6.6	21.1	46	1.18	1.48	1.36
16	10.51	41	91.2	1.49	1.52	1.37
32	19.33	83.2	180.2	1.62	1.50	1.39
64	36.25	167.2	361.6	1.72	1.50	1.38
128	70.9	325.5	730	1.76	1.54	1.37
256	142.4	685	1469	1.76	1.46	1.36

Figure 8-4: Numerical Array Transfer in Voyager on 2 NT machines



	0	0.25	0.5	1	2	4	8	16	32	64	128	256
byte	0.00	0.11	0.19	0.31	0.48	0.85	1.18	1.49	1.62	1.72	1.76	1.76
int	0.00	0.31	0.51	0.75	1.10	1.35	1.48	1.52	1.50	1.50	1.54	1.46
double	0.00	0.50	0.78	1.03	1.20	1.35	1.36	1.37	1.39	1.38	1.37	1.36

The test shows that:

- Byte array transfer becomes saturated at the array size of 128 KB, at which it has a throughput of 1.76 MB/sec, for int this saturation point occurs at the array size of 16 KB.

- Numerical array transfer throughput on 2 machines is nearly double the numerical array transfer throughput on a single machine
- Numerical array transfer using Voyager is much slower than using Java RMI and HORB in both cases, single machine and 2 networked machines.

8.7 Garbage Collection

Objects that are no longer referred by any local or remote references are garbage-collected by the Voyager's distributed garbage collector. Voyager uses "delta pinging" scheme to check the existence of reference to remote objects. Each program notes when references to remote objects are created and destroyed. Every cycle 2 minutes (default value), the program sends each referenced remote program a single message containing a summary of the references to its objects that were added or removed since the last distributed garbage collector cycle. By tracking this information as it changes over time, each program can tell when no remote references exist to an exported object.

Voyager supports both leased-based as in Java RMI garbage collection mechanism and time-based garbage collection mechanism. With the time-based mechanism, an object's life span can be defined based on specific length of time or a particular point in time. When the object life span is reached, it is garbage-collected by the Voyager's garbage collector. Time-based garbage collection is often used to create roaming agents that automatically self-destruct after it reaches its life span (normally, a few days).

9. VisiBroker

DOT: Inprise's VisiBroker 3.3

Developed by: Inprise Corporation. Website: http://www.inprise.com

Version tested: VisiBroker 3.3 for Java

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - Sun's JDK 1.1.6 Java Compiler
- Java VM:
  - Sun's JDK 1.1.6 with Symantec's JIT built in

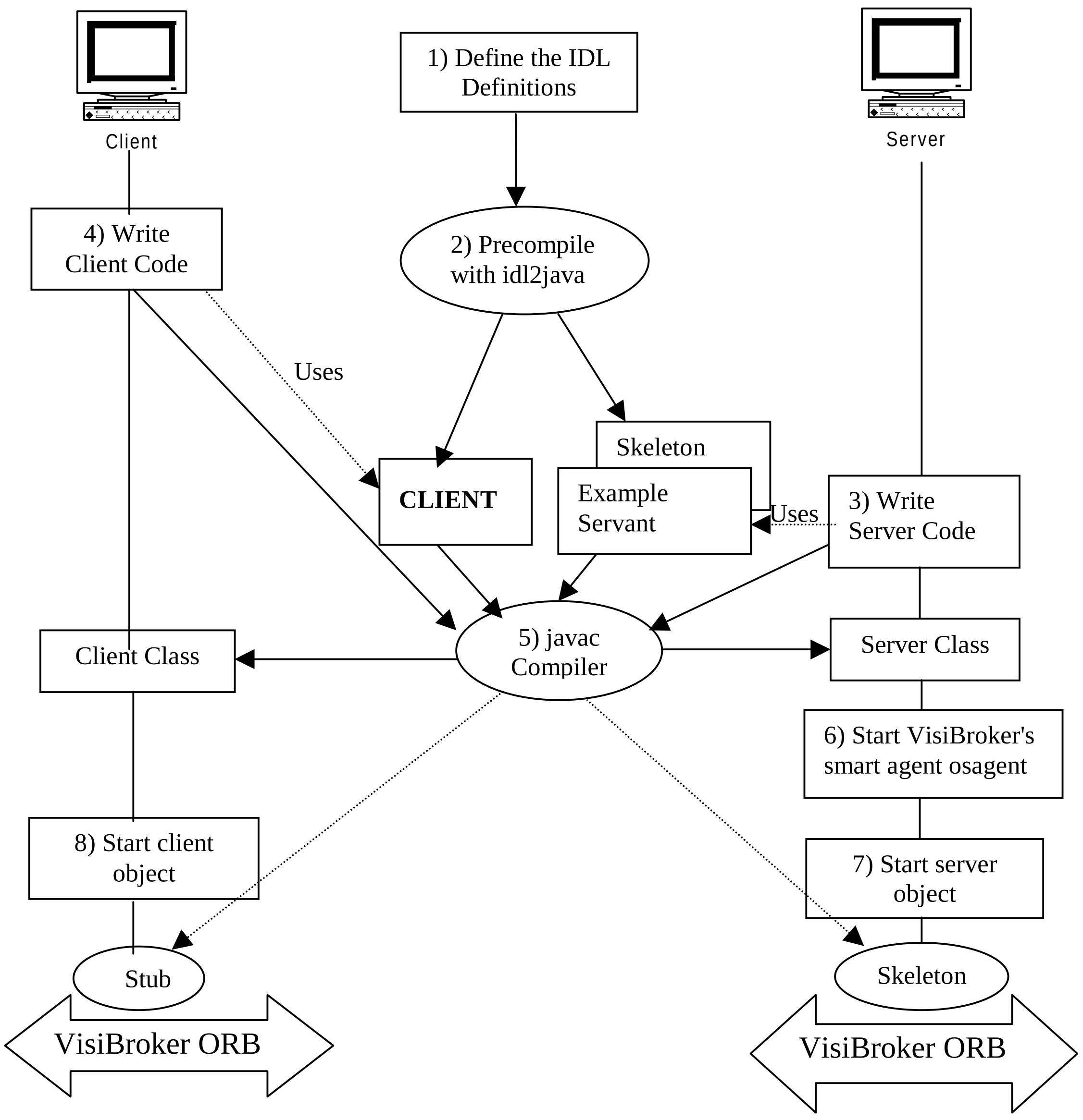
VisiBroker is a complete CORBA 2.2 specification for developing distributed object-based application and is now distributed by the Inprise Corporation. VisiBroker of Java has several key features, the following list shows the main features of VisiBroker for Java 3.3:

- **Interface Repository**. An online database contains meta information about ORB object types.
- **Dynamic Invocation Interface**. Client program can query object repository for an object interface and request a operation on it.
- **Dynamic Skeleton Interface**. Allows server to construct object on the fly to serve the client operation request
- **Smart Binding**. Client performs local method call for object which resides on the same process with the client program, otherwise the client uses IIOP to make the call.
- **Smart Agents**. Smart Agent can automatically reconnect a client program to an appropriate object on the server if the server being used becomes unavailable due to communication failures. It can use VisiBroker's Object Activation Daemon to launch instance of server process on demand.
- **Object Action Daemon**. OAD automatically activates a server when client program requests a bind on a server object.
- **Enhanced Thread and Connection Management**. Provides thread policy for object servers. VisiBroker automatically selects the most efficient way to manage connections between client applications and servers.

- **Location Service**. Working with the Smart Agents on a network, the Location Service can see all the available instances of an object to which a client can bind. The main use of this feature is load balancing.
- **Object Request Debugger**. Incorporation with a Java GUI application to allow programmer to trace the invocation from client to server
- **Web naming**. Associates Uniform Resource Locators (URLs) with objects, allowing an object reference to be obtained and an object to be contacted by specifying a URL.
- **Define interface without IDL**. VisiBroker's java2iiop compiler to use the Java language to define interfaces, instead of using IDL.
- **Smart Stubs**. Provides capability of caching, load balancing, logging
- **Interceptors**. Supports for interceptors is a powerful feature that exposes entry points in the ORB communication between clients and servers. Interceptor code can be inserted at particular points in the ORB message and request processing. Using interceptors, we can view communications between clients and servers; and we can modify these communications if we wish, effectively altering the behavior of the ORB.
- **Communication Event Handlers**. The event handling mechanism notifies clients and object implementations of system events and can be used to implement accounting, tracing, debugging, logging, security and encryption facilities.

9.1 Development Process

Figure 9-1: Distributed object application development in VisiBroker



The graph above shows the steps in development a distributed program in VisiBroker 3.3 for Java. Let's go through each step one-by-one and see what's involved:

1. Define server interfaces using the Interface Definition Language. Here's the Server.idl defined for our benchmark program.

```
module vbroker {
interface Server {
  Server createInstance();
  long methodA1(in long a1);
  long methodA2(in long a1, in long a2);
  long methodA3(in long a1, in long a2, in long a3);
```

```
        long methodA4(in long a1, in long a2, in long a3, in long a4);
        long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
        long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

        typedef sequence<octet> ByteArray;
        typedef sequence<long> IntArray;
        typedef sequence<double> DoubleArray;
        void methodB(in ByteArray ba);
        void methodI(in IntArray ia);
        void methodD(in DoubleArray da);

    };
};
```

The module statement is equivalent to Java package statement, it creates a scoped name that consists of one or more identifiers.

2. Run the idl2java compiler. Compiler the Server.idl using VisiBroker's idl2java compiler. This process produces client stubs for the IDL-define methods, server skeleton and an example class that can be used to start with the implementation of our server. To use idl2java compiler, type the command at the dos prompt as follows:

   prompt> idl2java Server.idl

3. Write the Server code. We need to write the server implementation. Here's the code for our ServerImpl.java. We use the _example_Server.java as the start place to write our ServerImpl.java. (Please refer to appendix for the complete ServerImpl.java code that is used in our benchmark.)

```
// ServerImpl.java
package DOTBenchmark.visibroker;

public class ServerImpl extends vbroker._ServerImplBase {
  /** Construct a persistently named object. */
  public ServerImpl(java.lang.String name) {
      super(name);
  }

  /** Construct a transient object. */
  public ServerImpl() {
      super();
  }

  public static void main(String argv[]) {
                  try {
          // initialize the ORB
          org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
```

```
            // initialize the BOA (Basic Object Adapter)
            org.omg.CORBA.BOA boa = orb.BOA_init();

            String name = "Server";
                    DOTBenchmark.visibroker.ServerImpl server = new
                    DOTBenchmark.visibroker.ServerImpl(name);

            try {
            boa.obj_is_ready(server);
            System.out.println("ServerImpl is ready at: "+name);
            } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
            }
            // wait for incoming requests
            boa.impl_is_ready();

        }
        catch (Exception e) {
                e.printStackTrace();
        }
    }

    public vbroker.Server createInstance() {
        ServerImpl obj = null;
        try {
                obj = new ServerImpl();
                return obj;
        } catch (Exception e) {
                e.printStackTrace();
        }
        return null;
    }

     public int methodA1(int a1) {
        return 0;
    }

    public int methodA2(int a1,int a2) {
        return 0;
    }

    public int methodA3(int a, int a2, int a3) {
        return 0;
    }

    public int methodA4(int a1, int a2, int a3, int a4) {
    // implement operation...
```

```
                return 0;
            }

             public int methodA5(int a1,int a2, int a3, int a4, int a5) {
                return 0;
            }

            public int methodA6(int a1, int a2, int a3, int a4,  int a5, int a6 ) {
                return 0;
            }
             public void methodB( byte[] ba ) {
            }

            public void methodI(int[] ia ) {
            }

            public void methodD(double[] da ) {
            // implement operation...
            }
    }
```

As we can see the bold statement above, the org.omg.CORBA.ORB method returns an object of type org.omg.CORBA.ORB. We can now initialize the BOA. The BOA_init method returns an object reference for BOA. We use this reference to register our ServerImpl object with the BOA. Finally, we tell the BOA that our object is ready to serve the client request.

The BOA.obj_is_ready() call is different from the standard. With VisiBroker, the obj_is_ready() call is used to register individual object implementations on the server.

4. Write the Client code. The Client.java implementation basically will call the methods defined in our Server.idl that has been implemented by the ServerImpl.java. Our client code has most of the code that we have seen in the previous DOT packages. Here's the code that we need to discuss a little bit about it.

   ```
   // Initialize the ORB.
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv,null);
   ```

   This method returns an object of type org.omg.CORBA.ORB. It provides a way to pass in the main program's command line arguments, which let us set certain ORB-related properties at run time.
5. Compile all the code. This step involves using Java compiler to compile all the codes, these include our ServerImp.java, Client.java and its related source files.
6. Start the VisiBroker's OSAgent. The VisiBroker Smart Agent will provide a service to locate instances of server objects. Enter this command at the dos command prompt to run VisiBroker Smart Agent

prompt>osagent

Make sure the system's path can reach the VisiBroker's vbroker\bin folder

7. Start the Server. Start up our Server object using Java VM on the server machine.

```
prompt>java DOTBenchmark.visibroker.ServerImpl
```

Note that our ServerImpl is in the package DOTBenchmark.visibroker

8. Start the Client. Start up our Client object using Java VM on the client machine. Enter the command below at the dos prompt

prompt>java DOTBenchmark.visibroker.Client 100 servername

Note that our Client is also defined in the package DOTBenchmark.visibroker. The number 100 defines the number of iterations and *servername* is the machine name that the server runs on.

9.2 Remote Object Connection

When creating an object, server must specify an object name if the object is to be made available to the client application through the *osagent*. When the server call the method BOA.boa_is_ready, the object interface name will only be registered with the VisiBroker osagent if the object is named. If an object name is not specified when bind is called, the VisiBroker osagent will return any suitable object with the specified interface.

Remote object connection benchmark measures how fast the client can locate the server object. To locate object implementations, VisiBroker uses its own Location Service mechanism, which is a specialized daemon program (osagent or Smart Agent as it is named). The Location Service can be used to query all available instances of an object based on the interface repository ID or with named object.

In this benchmark, our client program uses the bind method of the ServerHelper class to connect to the server object named "Server". The bind methods require that an org.omg.CORBA.ORB object be supplied as the first parameter. Here's part of the code for this benchmark.

```
        start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
            vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("bind to server(): "+elapse+" msec");
```

The table below shows VisiBroker Remote Object Connection benchmark on a single machine as well as on 2 networked machines. The result is an average of 100 iteration call "bind" method by the client program.

Table 9-1: Remote Object Connection in VisiBroker

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java
12.10 msec	11.11 msec

The test shows that client can locate the server object with a reasonable fast operation. In fact, it is the second fastest among DOT packages we have tested in this project so far.

9.3 Remote Object Creation

VisiBroker does not provide instantiating an object implementation from outside of the server address space. In order to perform this operation, we need to follow these steps:

1. create a new object, we can pass it a name (e.g. ServerImpl obj = new ServerImpl("server2"));
2. return the newly created object (e.g. return obj)

These steps are implemented in the method createInstance() in the ServerImpl.java. After the client program obtains the object reference to the remote server object, it uses this to invoke the method createInstance() to create a remote server object. Here is the code to measure this benchmark:

```
// In the ServerImpl.java
public vbroker.Server createInstance() {
    ServerImpl obj = null;
    try {
        obj = new ServerImpl();
        return obj;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

```
    // In the Client.java
    vbroker.Server server2 = null;
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
        server2 = server.createInstance();
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("object creation: "+elapse+" msec");
```

Table below shows the Remote Object Creation test using VisiBroker 3.3 for Java. The result on 2-networked machine is much faster than the result on a single machine. These results show of speed penalty for using the object creation simulation.

Table 9-2: Remote Object Creation in VisiBroker

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java
15.90 msec	10.41 msec

9.4 Remote Method Call

The code for Remote Method Call benchmark is the same as we used to test this operation for previous DOT packages. First of all the client obtained the server object reference, then it uses this reference to make method call. I listed the code here just for completeness.

```
// prepare server for next benchmark tests
vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");

// load class files explicitly
 // ignore this just WARM UP
System.gc();
 for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}

// remote method benchmark
// one argument
```

```
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA2(100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

//  three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA3(100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

// four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA4(100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");
```

```
// six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA6(100, 100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
```

Table 9-3: Remote Method Call in VisiBroker

Remote Method Call with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java (Unit time in msec)					
Args	1	2	3	4	5	6	1	2	3	4	5	6
VisiBroker 3.3 for Java	2.2	2.7	2.8	2.7	2.8	2.7	1.7	1.7	1.7	1.7	1.7	1.8

The results of the Remote Method Call with variable arguments benchmark shows that there is no significant difference between (small) number of argument call. It suggests that the marshalling, unmarshalling and the transport handling mechanism by VisiBroker's Stub and Skeleton is efficient. It is as fast as Remote Method Call in RMI and Voyager.

9.5 Object Data Transfer

All the previous DOT we have seen so far allow us to pass objects by value, i.e. the actual state of the object is copied or returned in a method call, Java program can invoke methods on the cloned object that has the same state as the original object. This is done with the help of object serialization process, which turns object into an array stream of byte. In contrast, CORBA only supports pass-by-value for non-object types, such as all the primitive data type.

CORBA copies the values of "in" parameters from the client to the server and "out" parameters from the server to the client or "inout" parameters in both directions during the invocation and reply of a method. However, CORBA only passes an object's reference; it does not copy the state of the object, this means that if an object which inherits methods from its ancestor is passed across client and server process, then those methods can't be used after object is received because information of how we implemented it couldn't be found. In essence, passing a subtype object could result in receiving an unknown object.

CORBA assumes that all objects are remote, i.e. they can be accessed via their reference, and so passing an object reference in a call is sufficient. If we create remote applications starting from CORBA IDL, we won't have any problems because IDL won't let us pass objects by value. To pass an object, we have to make it a remote object.

However, VisiBroker for Java provides passing object by value using a non-standard extension of IIOP; it is the basis for new OMG standard, but with some modifications. VisiBroker pass objects by value by using its own data type called "***extensible structs***", which are upwardly compatible extensions of CORBA ***structs***. When we use ***extensible structs***, objects are passed by value.

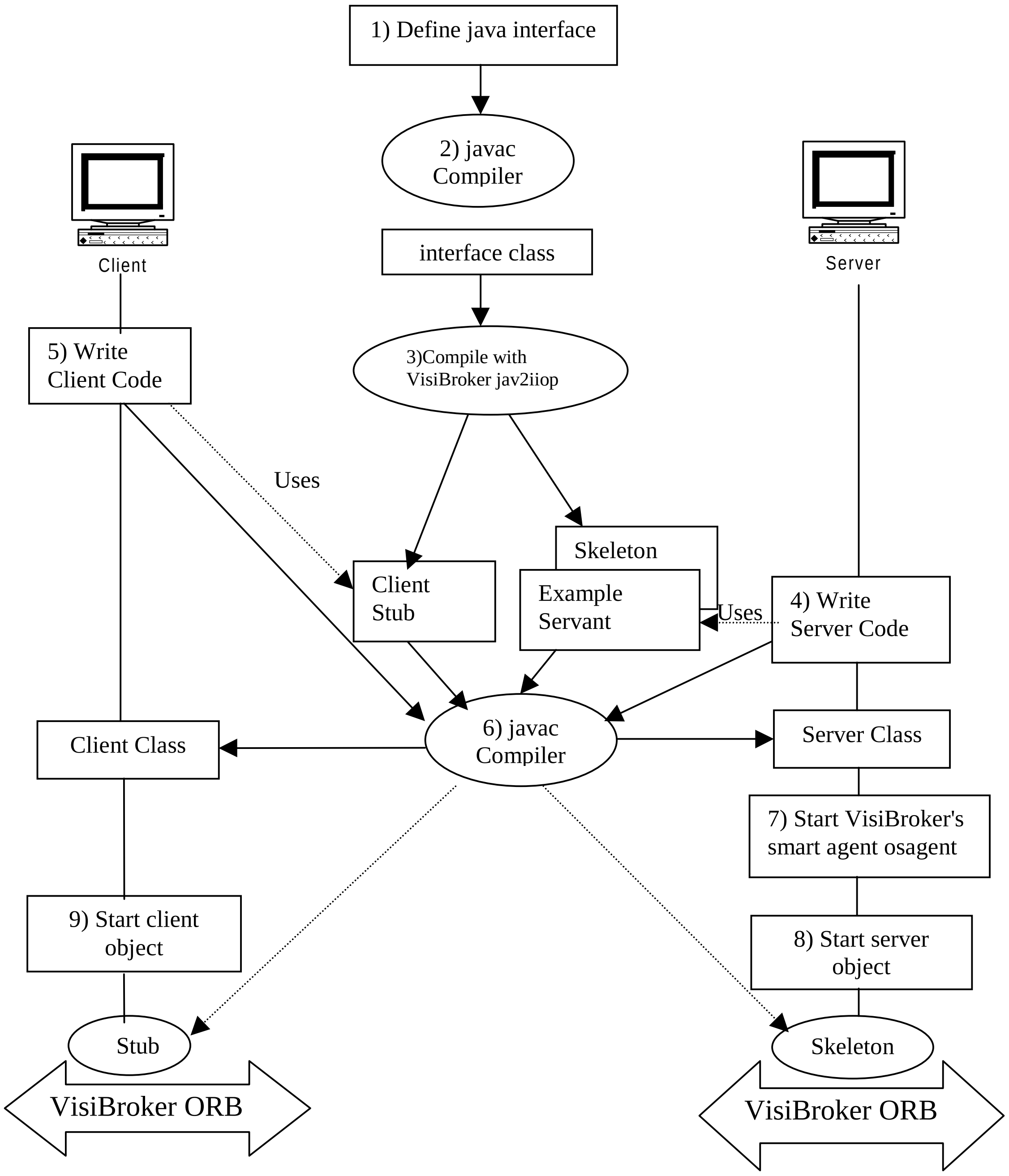
We can use ***extensible structs*** to pass objects by value with all CORBA language mapping, not just only Java. In theory, we could pass an object's state across languages such as C++, Smalltalk, OO-COBOL and Java. The use of extensible structs is a VisiBroker extension to the OMG IDL with an additional keyword "extensible". Because it is an non standard feature, we can't use our code to other ORBs vendor. Extensible structs allow us to uses classes that can be defined in Java but cannot be defined in IDL because of CORBA limitations.

VisiBroker uses Java serialization to pass classes in the form of extensible structs. ***Extensible structs*** is simply a way to flatten the state of an object and all its parent objects that can be passed on-the-wire using any IIOP ORB. Here is the main functions added by the extensible structs to an ordinary CORBA struct:

- supports for arbitrary recursive definitions that let us pass graphs of objects
- supports nested encoding
- supports an outer encoding that can represent the extensible struct as an opaque sequence of octets- this makes the struct transportable on IIOP ORBs that are not struct-aware.

The figure below shows the steps required building an application that performs object passing by value. Let's go through these steps one-by-one and see what's involved:

Figure 9-2: Caffein application development in VisiBroker



1. Define a remote interface. The server object declares its service via a remote Java interface. It does this by extending org.omg.CORBA.Object. Here's the code for our benchmark purpose.

```
package DOTBenchmark.visibroker;

// ServerOT.java
public interface ServerOT extends org.omg.CORBA.Object {
   public void methodDA(DataOT[] data);
}
```

2. Compile the interface. We must compile this remote interface using javac compiler.

```
prompt>javac –O ServerOT.java
```

3. Create IIOP client stubs and server skeletons. Run the **java2iiop** compiler against the class files to generate client stubs and server skeletons for our remote classes.

   prompt> java2iiop ServerOT

4. Write Server code. Here's the code for our ServerOTImpl.java

```
package DOTBenchmark.visibroker;

    public class ServerOTImpl extends {

   /** Construct a transient object. */
   public ServerOTImpl() {
      super();
   }

   public static void main(String argv[]) {
   try {
         // initialize the ORB
         org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

         // initialize the BOA (Basic Object Adapter)
         org.omg.CORBA.BOA boa = orb.BOA_init();
         String host = null, name = null;

         name = "Server";
         _example_ServerOT server = new _example_ServerOT(name);
         System.out.println("server was made: "+server);
         try {
         boa.obj_is_ready(server);
         System.out.println("ServerImpl is ready at: "+name);
         } catch (Exception e) {
         e.printStackTrace();
         System.exit(1);
         }
         // wait for incoming requests
         boa.impl_is_ready();
   }
   catch (Exception e) {
         e.printStackTrace();
   }
 }

}
```

5. Write the client code. The client code is just an ordinary Java code, it use a Naming Service to locate a remote object, it then invokes its method via a stub that serves as a proxy for the remote object. Here's the code for our ClientOT.java

```
// ClientOT.java
package DOTBenchmark.visibroker;
import java.io.*;
import java.util.*;
class ClientOT {
   public static void main(String argv[]) {
   int iter = 100;
   int i, x;
   long start, time;
   double elapse;
```

```
        //    sequence_of_DataHolder data = new sequence_of_DataHolder();
        String host = "127.0.0.1";
        String host2;
        if (argv.length == 2) {
            iter = Integer.parseInt(argv[0]);
            host = argv[1];
        } else if (argv.length == 1) {
            iter = Integer.parseInt(argv[0]);
        } else {
            System.err.println("java horb.examples.bench.Caffeine.Client [iteration]");
            System.exit(1);
        }
        // Object send in array without cast benchmark
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv,null);
        // Object send benchmark
        DOTBenchmark.visibroker.ServerOT server = DOTBenchmark.visibroker.ServerOTHelper.bind(orb, "Server");
        int[] sizes = {1, 10, 20, 30, 40, 50};
        for (int s = 0; s < 6; s++) {
            DOTBenchmark.visibroker.DataOT[] data = new DOTBenchmark.visibroker.DataOT[sizes[s]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                server.methodDA(data);
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodDA(Data["+sizes[s]+"]): "+ elapse+" msec");
        }
    }
}
```

6. Compile all the codes. We need to compile ServerOTImpl.java and ClientOT.java
7. Start the VisiBroker's Naming Service. Run the Osagent on the server machine

   prompt>osagent
8. Start the server object. Run ServerOTImpl using Java VM on the server machine

   prompt>java ServerOTImpl
9. Start the client object. Run ClientOT using Java VM on the client machine.

   prompt>java ClientOT 100 serverlocation

As usual, we specify the location that the server is running on and the iteration for the test is 100.

Table 9-4: Transfer object by value using java2iiop in VisiBroker

Object Data Transfer with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: VisiBroker 3.3 for Java (Unit time in msec)					
Number of objects	**1**	**10**	**20**	**30**	**40**	**50**	**1**	**10**	**20**	**30**	**40**	**50**
VisiBroker 3.3 for Java	7.2	4.4	6.6	7.7	9.9	11	7.42	4.21	5.9	8.52	10.41	12.52

The table above shows the test of object data transfer using VisiBroker's java2iiop. The result shows that:

- Time to pass an array of object (the basic Data as defined for previous DOTs) depends mostly on the length of data.
- Encapsulating a field into object adds more overheads for the process of marshalling and unmarshalling. Hence, when passing array of object, the invocation time mostly depends on the number of array.
- Performance on single machine is faster than performance over 2 networked machines, this hold for any number of array lengths.

9.6 Numerical Array Transfer

This benchmark tests the speed of transferring the array of primitive data using VisiBroker. The array length is determined at runtime. Operations were evaluated on a single machine and on 2 networked machines.

Table 9-5: Numerical Array Transfer in VisiBroker on the same machine

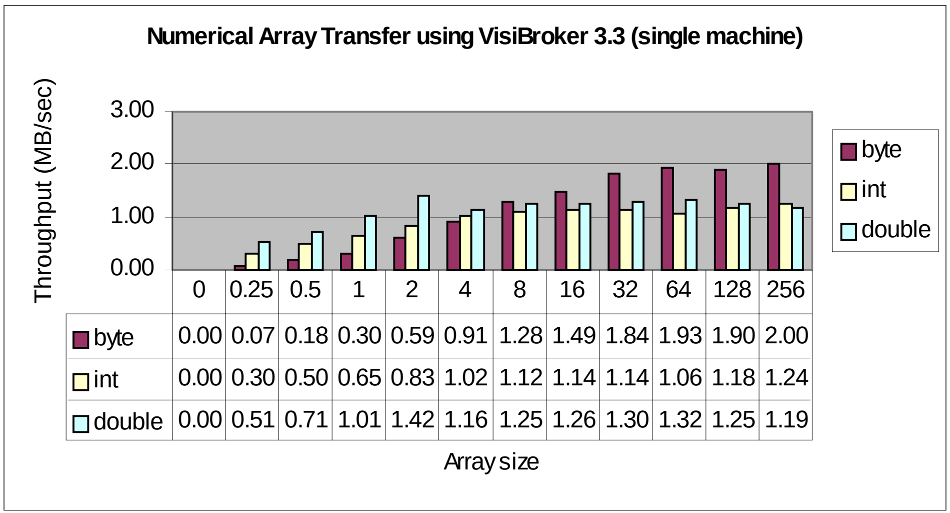
Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default

Software: Java JDK 1.1.6, VisiBroker for Java 3.3

(Unit time in msec)

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	3.3	2.8	2.8	0.00	0.00	0.00
0.25	3.3	3.3	3.8	0.07	0.30	0.51
0.5	2.7	3.9	5.5	0.18	0.50	0.71
1	3.3	6	7.7	0.30	0.65	1.01
2	3.3	9.4	11	0.59	0.83	1.42
4	4.3	15.3	27	0.91	1.02	1.16
8	6.1	28	50	1.28	1.12	1.25
16	10.5	55	99	1.49	1.14	1.26
32	17	110	192	1.84	1.14	1.30
64	32.4	236	379	1.93	1.06	1.32
128	65.9	423	797	1.90	1.18	1.25
256	125	808	1675	2.00	1.24	1.19

Figure 9-3: Numerical Array Transfer in VisiBroker on the same machine



	0	0.25	0.5	1	2	4	8	16	32	64	128	256
■ byte	0.00	0.07	0.18	0.30	0.59	0.91	1.28	1.49	1.84	1.93	1.90	2.00
□ int	0.00	0.30	0.50	0.65	0.83	1.02	1.12	1.14	1.14	1.06	1.18	1.24
□ double	0.00	0.51	0.71	1.01	1.42	1.16	1.25	1.26	1.30	1.32	1.25	1.19

The test shows that:

- Transferring array of byte and int has throughput increases with the array size, which reaches its peak at about 2 MB/sec at the array size of 256 KB for transferring array of byte and 1.24 MB/sec at the array size of 128 KB for transferring array of int
- Transferring array of double becomes saturated at 1.42 MB/sec at the array size of 2 KB
- Transferring longer array with byte and int is better than transferring longer array with double.

Table 9-6: Numerical Array Transfer in VisiBroker on 2 NT machines

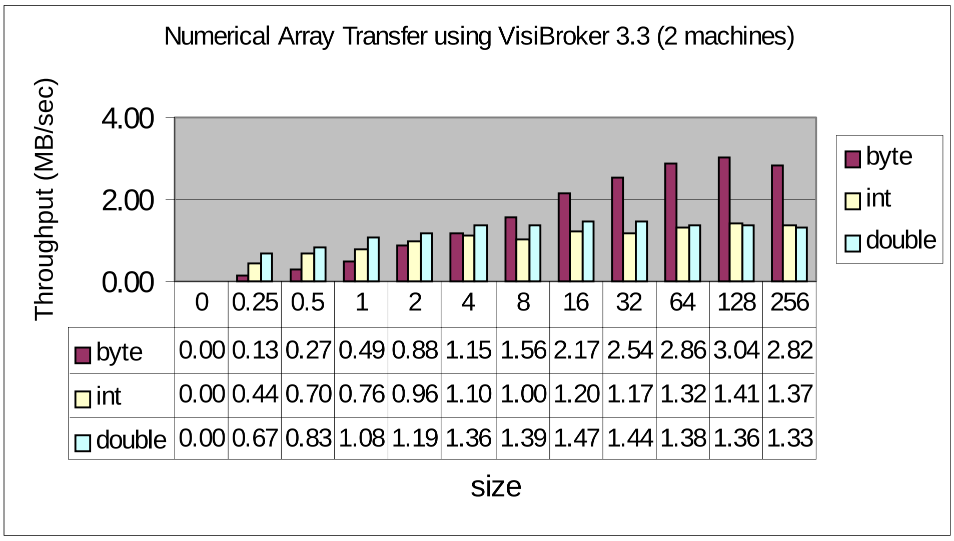
Two Windows NT Machines

- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6, VisiBroker for Java 3.3

(Unit time in msec)

Array	Time (msec)			Throughput (MB/sec)		
Size in KB	byte	int	double	byte	int	double
0	1.7	1.8	2.1	0.00	0.00	0.00
0.25	1.81	2.21	2.91	0.13	0.44	0.67
0.5	1.8	2.8	4.7	0.27	0.70	0.83
1	2	5.11	7.21	0.49	0.76	1.08
2	2.21	8.11	13.1	0.88	0.96	1.19
4	3.4	14.22	23	1.15	1.10	1.36
8	5.01	31.1	45.1	1.56	1.00	1.39
16	7.21	52.1	85.1	2.17	1.20	1.47
32	12.32	107.1	173.3	2.54	1.17	1.44
64	21.83	189.3	363.5	2.86	1.32	1.38
128	41.16	354.5	735.1	3.04	1.41	1.36
256	88.73	730.1	1499	2.82	1.37	1.33

Figure 9-4: Numerical Array Transfer in VisiBroker on 2 NT machines



size	0	0.25	0.5	1	2	4	8	16	32	64	128	256
byte	0.00	0.13	0.27	0.49	0.88	1.15	1.56	2.17	2.54	2.86	3.04	2.82
int	0.00	0.44	0.70	0.76	0.96	1.10	1.00	1.20	1.17	1.32	1.41	1.37
double	0.00	0.67	0.83	1.08	1.19	1.36	1.39	1.47	1.44	1.38	1.36	1.33

The result on 2 networked machines indicates that:

- Throughput for transferring array of byte reaches its peak at 3.04 MB/sec at the array size of 128 KB, for int 1.37 MB/sec at 256 KB and double 1.47 MB/sec at 16 KB
- Throughput of transferring array of byte is much higher than int and double starting from the array size of 8 KB.
- Throughput of transferring array of primitive data on 2 machines is much higher throughput on a single machine

9.7 Garbage Collection

VisiBroker does not support built-in distributed garbage collection. It relies on the client and server machine VM in doing the garbage collection. However, I heard that many other companies who are using the product have used **interceptors** in conjunction with various models (e.g. reference counting, pinging, etc) to come up with their own distributed garbage collection.

10. Iona OrbixWeb

DOT: Iona's OrbixWeb for Java

Developed by: Inprise. Website: http://www.iona.com

Version tested: OrbixWeb 3.0

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - Sun's JDK 1.1.6 Java Compiler
- Java VM:
  - Sun's JDK 1.1.6 with Symantec's JIT built in

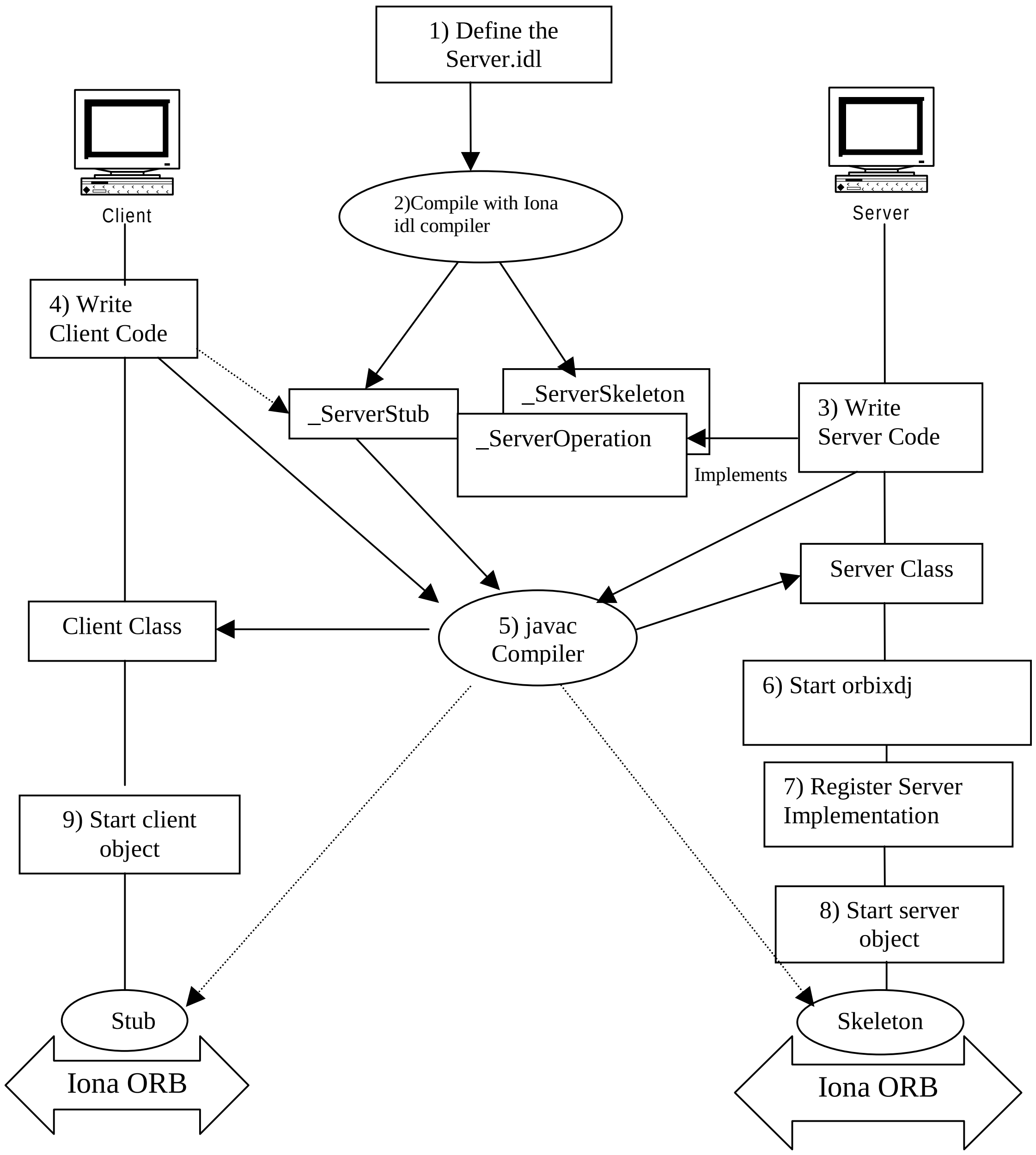
IONA Technologies is the world’s leading provider of CORBA technology. The Orbix product was launched in June 1993 as the first full and complete implementation of the CORBA standard and is the market-leading implementation of the CORBA standard. With Orbix, programmers can develop distributed, object-oriented C++ applications following a consistent and straightforward, standards-based model. IONA also provides CORBA technology for the Ada 95 and Smalltalk languages.

OrbixWeb 3 is a fully OMG's CORBA 2.0 -compliant client/server development and runtime environment for Java. Therefore, it can be used in conjunction with stub and skeleton code generated by third party Java ORBs which comply with the OMG mapping specification.

10.1 Development process

Just as we did in developing distributed applications in CORBA, the first step to developing OrbixWeb Application is to define an IDL interface, then compiler the IDL file to generate stub and skeleton code so that client and server can communicate through the CORBA bus. The figure below shows the steps required in developing the benchmark test for OrbixWeb. Let's go through step-by-step to see what's involved:

Figure 10-1: Distributed object application development in OrbixWeb



1. Define server interfaces using the Interface Definition Language. Here's the Server.idl defined for our benchmark program.

```
interface Server {
    Server createInstance();
    ong methodA1(in long a1);
    long methodA2(in long a1, in long a2);
    long methodA3(in long a1, in long a2, in long a3);
    long methodA4(in long a1, in long a2, in long a3, in long a4);
    long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
    long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

    typedef sequence<octet> ByteArray;
    typedef sequence<long> IntArray;
    typedef sequence<double> DoubleArray;
    void methodB(in ByteArray ba);
    void methodI(in IntArray ia);
    void methodD(in DoubleArray da);
};
```

As we can see, this Server.idl file is exactly the same as we defined in VisiBroker benchmark.

1. Run the Iona's idl compiler. Compiler the Server.idl using Iona's idl compiler. This process produces client stubs for the IDL-define methods, server skeleton and a couple of helper classes. To run idl compiler, type the command at the dos prompt as follows:

   prompt> idl Server.idl

2. Write the Server code. We need to write the server implementation. Here's the code for our ServerImpl.java. We use the _example_Server.java as the start place to write our ServerImpl.java. (Please refer to appendix for the complete ServerImpl.java code that is used in our benchmark.)

```
package DOTBenchmark.orbix;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb.Features.Config;
import org.omg.CORBA.ORB;
import ServerPackage.*;

import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.SystemException;

public class ServerImpl implements _ServerOperations {
    public static void main(String argv[]) throws Exception {
            ORB.init();

            String host = null, name = null;
```

```
          try {
               host = java.net.InetAddress.getLocalHost().getHostName();
          } catch (Exception e) {
               e.printStackTrace();
               System.exit(1);
          }
          ServerImpl server;
          Server serverTie;
          server = new ServerImpl();
          serverTie = new _tie_Server(server);
          System.out.println("server was made");
          _CORBA.Orbix.impl_is_ready("Server");
 }

 public Server createInstance() {
     ServerImpl obj = null;
     try {
          obj = new ServerImpl();
          Server serverTie = new _tie_Server(obj);
          return serverTie;
     } catch (Exception e) {
          e.printStackTrace();
     }
     return null;
}
public int methodA1(int a1) {
     return 0;
}
public int methodA2(int a1,int a2) {
     return 0;
 }
 public int methodA3(int a, int a2, int a3) {
     return 0;
 }
public int methodA4(int a1, int a2, int a3, int a4) {
 // implement operation...
     return 0;
}
public int methodA5(int a1,int a2, int a3, int a4, int a5) {
     return 0;
}
public int methodA6(int a1, int a2, int a3, int a4,  int a5, int a6 ) {
     return 0;
}
public void methodB( byte[] ba ) {
}

 public void methodI(int[] ia ) {
        }
         public void methodD(double[] da ) {
        // implement operation...
        }
    }
```

As we can see the bold statement above, the method of org.omg.CORBA.ORB.init() must be called by the server (as well as client as we will see later on). Here we use a new style of programming in CORBA, which use the _tie approach. There are two styles of programming in CORBA: **inheritance-based** and **delegate-based**. The inheritance-based we'd already seen in the VisiBroker benchmark program. The delegate-based is often used in integrating legacy code written in non-OO languages. As we can see, the _tie_Server is a delegator class for the Server interface; it delegates every call to the real implementation class, i.e. the ServerImpl. We must pass an instance of the real implementation class to the Tie object via its constructor when we first initialize it.

3. Write the Client code. The Client.java implementation basically will call the methods defined in our Server.idl that has been implemented by the ServerImpl.java. Our client code has most of the code that we have seen in the previous DOT packages. As a requirement, our client code must call the org.omg.CORBA.ORB.int() to get a object reference to the ORB.
4. Compile all the code. This step involves using Java compiler to compile all the codes. Note that all the source codes will be compiled with optimizer option turned on.
5. Start up the orbixdj. The OrbixWeb daemon process must be running on the server machine. Enter this command at the dos command prompt

   prompt>orbixdj
6. Bind the interface definition to the Implementation Repository. We need to load the Server interface information into the Implementation Repository using Iona's **putit** command. This will allow the server to be launched automatically. The Implementation Repository is a server 'database' which maintains a mapping from the server name to the name of the Java bytecode that implements that server. If the server is registered, it is automatically run through the Java interpreter when a client binds to our Server object.

   To register the server, use the Iona's **putit** command. Enter the command below at the command prompt

   prompt>putit Server -java ServerImpl

Note: Before registering the server (or running the client), make sure that an OrbixWeb daemon process (orbixd or orbixdj) is running on the server machine.

The first parameter to **putit** is the server's name which we have chosen as Server. This is also the name of the server passed to impl_is_ready().
The <class name> parameter is the name of the class which contains the server main()

method, that is the name of the class which should be interpreted by the Java interpreter. In our case, this is the ServerImpl.

7. Start the Server. Start up our Server object using Java VM on the server machine.

   prompt>java DOTBenchmark.orbix.ServerImpl

   Note that our ServerImpl is in the package DOTBenchmark.orbix
8. Start the Client. Start up our Client object using Java VM on the client machine. Enter the command below at the dos prompt

   prompt>java DOTBenchmark.orbix.Client 100 servername

Note that our Client is also defined in the package DOTBenchmark.orbix. The number 100 defines the number of iterations and servername is the machine name that the server runs on.

10.2 Remote Object Connection

Our client program uses the ***bind*** method to get a reference of the remote server object via the ORB. The test for this operation measures how long it takes the client program to connect to the Server object which is already running on a server host. Here's the code for this:

```
// remote object connection benchmark
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    server = ServerHelper.bind(":Server", host);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("bind to server(): "+elapse+" msec");
```

The table shows the benchmark result for Remote Object Connection in OrbixWeb on a single machine and on 2 networked machines.

Table 10-1: Remote Object Connection in OrbixWeb

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0
14.30 msec	23.63 msec

The result shows that:

- Binding to an implementation remote object on two networked machines is much slower than binding on a single machine. This suggests that connect and disconnect to the server object through a distributed environment should be carefully done. It's better to maintain the connection to last in a longer period than to disconnect and reconnect in a short period.
- Compare to other DOTs so far, OrbixWeb is the slowest of all in the benchmark test for this operation.

10.3 Remote Object Creation

OrbixWeb does not have a function to create Server object outside of the server address space, we use the same technique to simulate the creation of object implementation accessible from outside as we did for VisiBroker. Here's the code just for completeness:

```
// In the ServerImpl.java
public Server createInstance() {
  ServerImpl obj = null;
  try {
      obj = new ServerImpl();
      Server serverTie = new _tie_Server(obj);
      return serverTie;
  } catch (Exception e) {
      e.printStackTrace();
  }
  return null;
  }
  // In the Client.java
  Server server2 = null;
  start = System.currentTimeMillis();
  for (i = iter; i > 0; --i) {
      server = ServerHelper.bind(":Server", host);
      server2 = (Server)server.createInstance();
  }
  time = System.currentTimeMillis() - start;
  elapse = (double)(time)/(double)(iter);
  System.out.println("create remote object: "+elapse+" msec");
```

Here is the result for Remote Object Creation benchmark in OrbixWeb

Table 10-2: Remote Object Creation in OrbixWeb

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0
16.50 msec	20.53 msec

The test shows that:

- Creating new server object on a single machine is faster than the creating the new server object on 2 networked machines.
- In a situation that we need to have many server objects running at the same time, it is better to connection to an existing server object and creates new server objects on demand than to start up many servers before hand and let the client programs to connect to it.

10.4 Remote Method Call

The Remote Method Call benchmark measures the invocation time for method with variable argument calls. The code for this operation is exactly the same as the use for the VisiBroker benchmark, the only difference is how to connect to the server object as is described in the above remote object connection benchmark.

Table 10-3: Remote Method Call in OrbixWeb

Remote Method Call with variable arguments												
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: OrbixWeb 3.0 (Unit time in msec)					
Args	1	2	3	4	5	6	1	2	3	4	5	6
OrbixWeb 3.0	4.40	3.90	3.80	4.40	3.90	4.40	3.81	3.30	3.71	3.10	3.20	3.61

The test shows that:

- Invocation time is roughly the same regardless the number of arguments call, with the exception of the first call which always takes longer time than subsequent calls although the argument call increased.
- Paradoxically, local invocation times may sometimes be longer than network invocation time. This indicates that the marshalling and unmarshalling process in IONA is a CPU intensive operation.

10.5 Object Data Transfer

OrbixWeb does not support object transfer by value.

10.6 Numerical Array Transfer

In this benchmark, we measure the performance of OrbixWeb in transferring primitive array of "byte", "int" and "double". The array is determined at runtime and the result shows the average of 100 times operation call.

Table 10-4: Numerical Array Transfer in OrbixWeb on the same machine

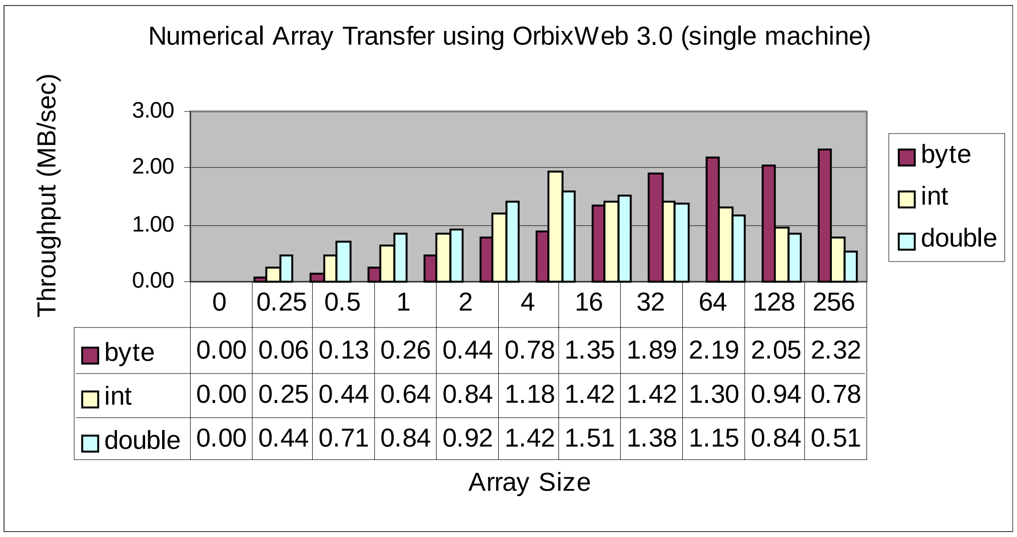
Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default

Software: Java JDK 1.1.6, OrbixWeb 3.0

(Unit time in msec)

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	3.8	3.3	3.3	0.00	0.00	0.00
0.25	3.9	3.9	4.4	0.06	0.25	0.44
0.5	3.8	4.4	5.5	0.13	0.44	0.71
1	3.8	6.1	9.3	0.26	0.64	0.84
2	4.4	9.3	17	0.44	0.84	0.92
4	5	13.2	22	0.78	1.18	1.42
8	8.8	16	39	0.89	1.95	1.60
16	11.6	44	83	1.35	1.42	1.51
32	16.5	88	181	1.89	1.42	1.38
64	28.6	192	434	2.19	1.30	1.15
128	61	533	1186	2.05	0.94	0.84
256	108	1274	3894	2.32	0.78	0.51

Figure 10-2: Numerical Array Transfer in OrbixWeb



	0	0.25	0.5	1	2	4	16	32	64	128	256
byte	0.00	0.06	0.13	0.26	0.44	0.78	1.35	1.89	2.19	2.05	2.32
int	0.00	0.25	0.44	0.64	0.84	1.18	1.42	1.42	1.30	0.94	0.78
double	0.00	0.44	0.71	0.84	0.92	1.42	1.51	1.38	1.15	0.84	0.51

The test shows that:

- Transferring byte array reaches its peak rate at 2.32 MB/sec at the array size of 256 KB
- Transferring int array reaches its peak rate at 1.95 MB/sec at the array size of 8 KB, which it decreases the throughput from 1.95 to 0.78 MB/sec at the array size increase from 8KB to 16, 32, 128 and 256 KB
- Transferring double array also reaches it peak throughput at the array size of 8 KB at which it has a rate of 1.60 MB/sec, then it decreases dramatically from 1.60 to 0.51 MB/sec. It indicates that marshalling and unmarshalling for large double array size is very inefficient in OrbixWeb.

Table 10-5: Numerical Array Transfer on 2 NT machines

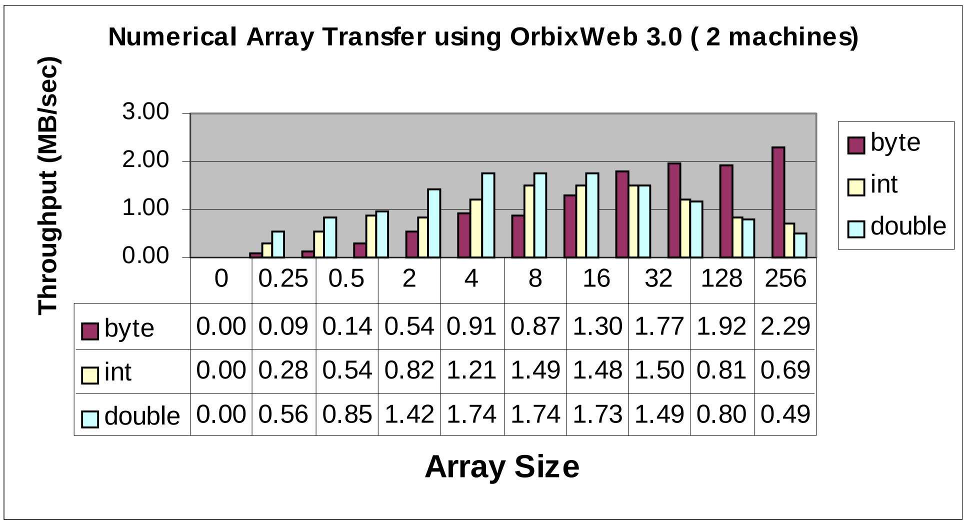
Two Windows NT Machines

- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6, OrbixWeb 3.0

(Unit time in msec)

Array	Time (msec)			Throughput (MB/sec)		
Size in KB	byte	int	double	byte	int	double
0	3.6	3.31	3.41	0.00	0.00	0.00
0.25	2.81	3.51	3.5	0.09	0.28	0.56
0.5	3.4	3.6	4.61	0.14	0.54	0.85
1	3.31	4.51	8.21	0.30	0.87	0.95
2	3.6	9.51	11	0.54	0.82	1.42
4	4.31	12.92	18	0.91	1.21	1.74
8	9.01	21	36	0.87	1.49	1.74
16	12.02	42.1	72.1	1.30	1.48	1.73
32	17.63	83.1	168.3	1.77	1.50	1.49
64	31.74	206.3	434.6	1.97	1.21	1.15
128	65	613.9	1251	1.92	0.81	0.80
256	109.3	1446	4063	2.29	0.69	0.49

Figure 10-3: Numerical Transfer on 2 NT machines using OrbixWeb



	0	0.25	0.5	2	4	8	16	32	128	256
byte	0.00	0.09	0.14	0.54	0.91	0.87	1.30	1.77	1.92	2.29
int	0.00	0.28	0.54	0.82	1.21	1.49	1.48	1.50	0.81	0.69
double	0.00	0.56	0.85	1.42	1.74	1.74	1.73	1.49	0.80	0.49

The result shows that:

- Transferring byte array reaches its peak at the array size of 256 KB, at which the peak rate is 2.29 MB/sec.
- Transferring int array reaches its peak at the array size of 32 KB, at which the peak rate is 1.50 MB/sec

- Transferring double array reaches its peak at the array size of 8 KB, which is the same size as in the case of single machine but it has the peak rate a little higher of 1.74 MB/sec
- Transferring array size larger than 8 KB for int and double would result in lower throughput than making two separated round-trip transfers.
- Again, transferring large array size for double is slow in OrbixWeb.

10.7 Garbage Collection

Orbix or the CORBA architecture does not provide a means of distributed garbage collection. There is no concept of **refCount** in OrbixWeb. The way to explicitly delete an object in OrbixWeb is to call the method _dispose(), passing the object reference of the object. Otherwise, the garbage collector deletes the objects when the server exits, or when there are no more references to the object. However, if there is an absolutely need to implement remote garbage collection, a reference counting mechanism can be implemented using Orbix Smart Proxies. For more information of how to implement this, please refer to the article ID: 141.261 at IONA webs site or follow this link:

http://www.iona.com/support/kb/Orbix_C++/articles/141.261.html

11. DCOM

DOT: Microsoft SDK 3.1 for DCOM

Developed by: Microsoft. Website: http://www.microsoft.com/java/sdk/default.htm

Version tested: DCOM95

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - Microsoft's jvc Java Compiler
- Java VM:
  - Microsoft's jview Java Virtual Machine

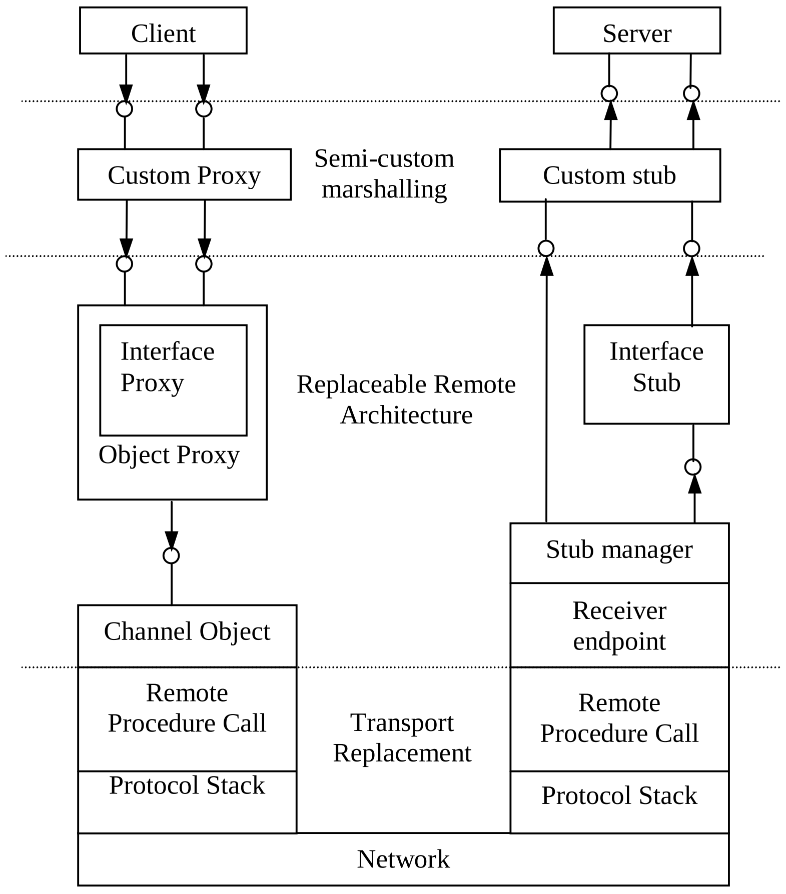
Microsoft's Distributed COM (DCOM) is an extension of the Component Object Model (COM) to support communication among objects which resides on different computers, which can be a LAN, WAN or the Internet. Because DCOM is a seamless evolution of COM, we can use existing COM-based applications, component, tools and knowledge and make them usable in a distributed computing. DCOM handles the low-level details of network protocols to facilitate the high-level communication, it frees developers from handling low-level details and therefore, they can concentrate on the high-level business logic in building business solution.

Here's a brief feature that supported in DCOM:

- **Transparency**. Supports both *nontransparent* and *transparent* modes. In nontransparent mode, a client can explicitly specify whether the server object class resides in an in process (DLL), local EXE or located on some other machine. In transparent mode, the client lets COM consult the registry to determine such attributes. Once the machine name is determined, COM will try to attach to a running server instance hosting the requested object class on that machine. If none exists, COM will automatically locate the server implementation file, start a server process and create an instance.
- **Extensibility**. Allows applications to customize the client-server connection that might use add-on features such as multicast service, asynchronous transport, compression or encryption packages.

There are three categories of extensibility: below, above and within. "Below" category extends COM at the Remote Procedure Call layer and below as shown in the figure below**:**

Figure 11-1: COM Remote Architecture and Extensibility



Notes on terminology

Object proxies. Acts as the client-side representatives of server objects and connects directly to the client.

Interface proxies. Performs client-side data marshalling and are aggregated into object proxies

Client-side channel objects. Uses RPCs to forward marshalled calls

Server-side endpoints. Receives RPC requests.

Server-side stub manager. Dispatches calls to the appropriate interface stubs

Interface stubs. Performs server-side data marshalling and make the actual calls on the objects

Standard marshaller. Marshals interface pointers into object references on the server side and unmarshals the object reference on the client side.

Currently, DCOM can be configured to run on TCP, UDP, IPX or NetBIOS.

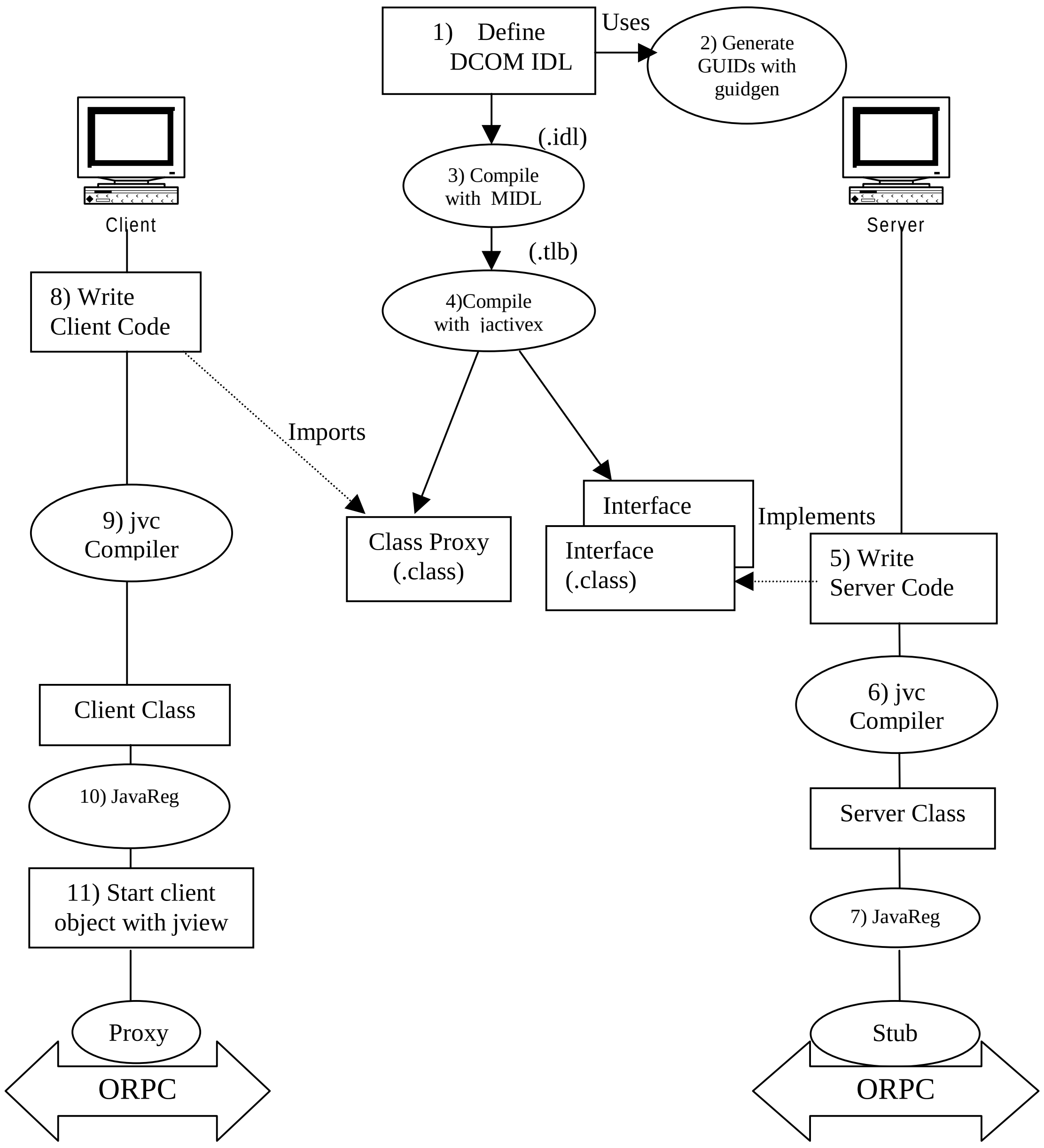
"Above" category allows inserting a handler layer above the standard remote architecture and below the user application. It is often called semi-custom marshaling because most of the tasks are delegated to the standard remote architecture. "Within" category is the most general extensibility type. For e.g. an application wants to replace the channel object and the receiver endpoint as shown in the figure above, while reusing object proxies and stub managers for object-identity maintenance and call dispatching. However, current COM architecture provides only limited support for this type of extensibility. A new component architecture call COMERA (COM Extensible Remote Architecture) has been proposed to promote binary reuse at the infrastructure level, thereby achieving this type of extensibility. For more information about this, please refer to the research paper by Y.M. Wang and W.J.Lee, "COMERA: COM Extensible Remote Architecture," which can also be found at the web site: http://www.research.microsoft.com/~ymwang/papers/COOTS98CR.htm

- **Indirection**. This is a special form of providing extensibility. COM's architecture supports "activation indirection" and "call indirection". "Activation indirection" occurs when a client requests the activation of a server object by specifying its CLSID. "Call Indirection" provides a way to indirectly forward a call to a new machine that can provide the same service to the client. This feature creates the basis for client transparent object migration and fault tolerance. The next generation of COM, the COM+, will provide a mechanism called interceptors, which support indirection through receiving events related to object creation as well as method invocation.
- **Versioning**. Interface identified by IID (Interface Identifier) is immutable, it cannot be changed in any way. A new implementation of the same CLSID must support an existing interface.
- **Server-lifetime management**. COM supports a rich set of server styles that require different server-lifetime management. COM uses reference counting to manage server lifetime; COM uses automatic "pinging" to solve the problem of an abnormally terminated client holding a reference indefinitely**.**
- **Multiple interfaces.** COM objects can support multiple interfaces.

For more information about DCOM, please visit the site:
http://www.microsoft.com/com/dcom.asp

11.1 Development Process

Figure 11-2: Distributed object application development in DCOM



The above figure shows the development process that was used in the project to evaluate DCOM performance. Here's the step-by-step involved:

1. Define DCOM IDL. We must define all the interfaces for our class. Here is the interface used in our project.

```
//dcoms.odl

[ uuid (29EE83A0-37AE-11d2-9539-F7F021215302) ]
library Ldcoms
{
   importlib("stdole32.tlb");

   [    uuid(29EE83A1-37AE-11d2-9539-F7F021215302),
        helpstring("IDConnect Interface")]
   dispinterface IDConnect
   {
   properties:
   methods:
        [id(1)]int dserv_1( [in] int p1 );
        [id(2)]int dserv_2( [in] int p1, [in] int p2 );
        [id(3)]int dserv_3( [in] int p1, [in] int p2, [in] int p3 );
        [id(4)]int dserv_4( [in] int p1, [in] int p2, [in] int p3, [in] int p4 );
        [id(5)]int dserv_5( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5 );
        [id(6)]int dserv_6( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6 );
        [id(7)]int dserv_7( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6, [in] int p7 );
        [id(8)]int dserv_8( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6, [in] int p7, [in] int
p8 );
        [id(9)]void dserv_iarray( [in] int * p1 );
        [id(10)]void dserv_barray( [in] unsigned char * p1 );
        [id(11)]void dserv_darray( [in] double * p1 );

   }

   [    uuid(29EE83A2-37AE-11d2-9539-F7F021215302),
        helpstring("CDConnect Class")]
   coclass CDConnect
   {
        dispinterface IDConnect;
   };
};
```

2. Generate GUIDs for our IDL interface. We can create the GUIDs and UUIDs for each class and interface defined in the IDL by using the guidgen utility in the SDK-Java.31.
3. Compile the IDL file to generate type library file. Use Microsoft interface compiler to generate type library file. Enter the command below at the dos command prompt

   prompt>mktyplib dcoms.odl /nocpp

Note: I have to installed VJ++ 1.1 in order to get mktyplib utility to compile the IDL.

4. Create Java wrappers for DCOM class. We can generate these wrappers for our DCOM class using Jactivex that included in the SDK-Java.31. Enter the command below at the DOS command prompt:

   prompt>Jactivex dcoms.tlb

5. Write the server code. We must write a Java file to implement all the interfaces defined in the IDL. We should use the summary.txt, which was generated by the type library compiler about as a template to start with. Here is the code CDConnect.java for this:

```
  // CDConnect.java
  import dcoms.*;
  import com.ms.com.*;
  import java.net.*;

  public class CDConnect implements IDConnect
  {
public int dserv_1( int p1 ) throws ComException{
    int x1 = p1;
    return 100; }
public int dserv_2( int p1, int p2 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
    return 100; }
public int dserv_3( int p1, int p2, int p3 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
    int x3 = p3;
    return 100; }
public int dserv_4( int p1, int p2, int p3, int p4 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
    int x3 = p3;
    int x4 = p4;
    return 100; }
public int dserv_5( int p1, int p2, int p3, int p4, int p5 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
    int x3 = p3;
    int x4 = p4;
    int x5 = p5;
    return 100; }
            public int dserv_6( int p1, int p2, int p3, int p4, int p5, int p6 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
    int x3 = p3;
    int x4 = p4;
    int x5 = p5;
    int x6 = p6;
    return 100; }
            public int dserv_7( int p1, int p2, int p3, int p4, int p5, int p6, int p7 ) throws ComException{
    int x1 = p1;
    int x2 = p2;
```

```
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        int x6 = p6;
        int x7 = p7;
        return 100; }
            public int dserv_8( int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8 ) throws
            ComException{
        int x1 = p1;
        int x2 = p2;
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        int x6 = p6;
        int x7 = p7;
        int x8 = p8;
        return 100; }
    public void dserv_iarray( int [] p1 ){
        int x,i;
        for( i = 0; i < p1.length; i++ )
        x = p1[i];

        return;
    }
    public void dserv_barray( byte [] p1 ){
        int i;
        byte x;
        for( i = 0; i < p1.length; i++ )
            x = p1[i];

        return;
    }
    public void dserv_darray( double [] p1 ){
        int i;
        double x;
        for( i = 0; i < p1.length; i++ )
            x = p1[i];

        return;
    }

     }
```

6. Compile the CDConnect.java with Microsoft's Jvc Java Compiler.
7. Register our server class. Use the Javareg to register our CDConnect as a server class into the registry. Enter this command at the command line.

prompt> javareg /register /surrogate /class:CDConnect /clsid:{29EE83A2-37AE-11d2-9539-F7F021215302}

8. Write the client code. Here we write the client implementation that is used for testing DCOM operation in the project. We cannot instantiate a DCOM class directly, we must use it interface and cast it to the interface and use the class.

   IDConnect cdc = (IDConnect)new dcoms.CDConnect();

   See appendix for complete client code.
9. Compile the client.java. We need to compile the client.java into class using Microsoft's Jvc compiler
10. Register the Java class. Use JavaReg to specify the remote machine that the client will run. Enter this command at the command line as follows:

    prompt> javareg /register /class:CDConnect /clsid:{29EE83A2-37AE-11d2-9539-F7F021215302} /remote:servername
11. Start the client. Start the client class using Microsoft Java VM.

    prompt>jview client 100 servername

Note. We don’t have to start up server class, the DCOM object factory knows how to start it up for us when the object is requested by client.
(I still have some problems to get DCOM going.)

11.2 Remote Object Connection

In DCOM, to allocate DCOM server object, we must create an instance of the class through interface, the code looks like this:

IDConnect cdc = (IDConnect) new dcoms.CDConnect();

This returns an interface point which refers to an object instance on a particular server that it is running; however this reference may or may not represent to the same object instance created on the server machine. Therefore, we couldn't measure the performance of DCOM in this respect.

11.3 Remote Object Creation

DCOM dynamically loads and creates the interface stubs and proxies that responsible for data marshalling. The test shows how fast DCOM can create a remote server object on the remote machine through the Ethernet LAN. I don't know how to get the client as a Local Server on the same machine. As usual, our client performs 1000 method invocation. Here is the part of code to measure this operation.

```
//Object Creation Benchmark
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc = ( IDConnect )new dcoms.CDConnect();
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("Object creation bench:"+elapse+"msec");
```

Table 11-1: Remote Object Creation in DCOM

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • SDK-Java.31 for DCOM95	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • SDK-Java.31 for DCOM95
Not available	8.12 msec

The test shows that Object Creation operation using DCOM is very fast, it is the second fastest after voyager we have seen above.

11.4 Remote Method Call

Table 11-2: Remote Method Call in DCOM

Remote Method Call with variable arguments							
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • SDK-Java.31 for DCOM95 (Unit time in msec)	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • SDK-Java.31 for DCOM95 (Unit time in msec)					
Args	NOT AVAILABLE	1	2	3	4	5	6
DCOM95		3.9	2.21	2.2	2.2	2.3	2.4

The test for DCOM in remote method call shows that:

- The invocation time does not vary much for method call with small variable argument, with the exception for the first call. This would suggest that the subsequent call after the first take advantage of the proxy marshalling and unmarshalling process of the first call.
- The operation of DCOM in this respect is slower than other DOTs compared in the project, it is the second slowest after OrbixWeb.

11.5 Object Data Transfer

DCOM does not support transfer object by copy

11.6 Numerical Array Transfer

I can't measure the performance of DCOM in transferring primitive data transfer, it seems to me that the call for transferring large numerical array data was not directed to the server, the time stays the same for all operations of calling.

11.7 Garbage Collection

DCOM supports distributed garbage collection. COM uses reference counting to keep track of server object lifetime. The server increases the reference count to every new reference to an object, i.e. upon returning the interface pointer. When the counts of a server object drops to zero, the server object realizes that it is no longer serving any client and can therefore be destroyed.

In the Java version, DCOM depends on the Java Virtual Machine's garbage collector to automatically keep track of references to an object. DCOM uses a pinging protocol to check if a connection is still alive. The client machines send a periodic message to the server. It stops sending this when the client process terminates. Upon detecting that the number of missing pinging periods has exceeded a threshold (default is more than 3 pinging periods of 2 minutes), the server-side assumes the client has died and so decrements the server object's reference count on the behalf of client.

12. Java IDL

DOT: JDK 1.2

Developed by: Sun Microsystems. Website: http://www.javasoft.com/

Version tested: JDK 1.2 beta4

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Compiler:
  - JDK 1.2 's javac Java Compiler
- Java VM:
  - JDK 1.2's Java JVM

Java IDL is the JavaSoft version of Java ORB which includes a CORBA/IIOP ORB as part of the JDK 1.2 core. Java IDL is free. It provides a development environment for generating CORBA stubs and skeletons from OMG's IDL. It also includes a CORBA Naming Service. With this Java IDL in hand, the Enterprise JavaBeans can use it to deliver a distributed component model. In addition, Enterprise JavaBeans is built on top of the Java Transaction Service (JTS), which is the Java implementation of the CORBA Object Transaction Service (OTS).

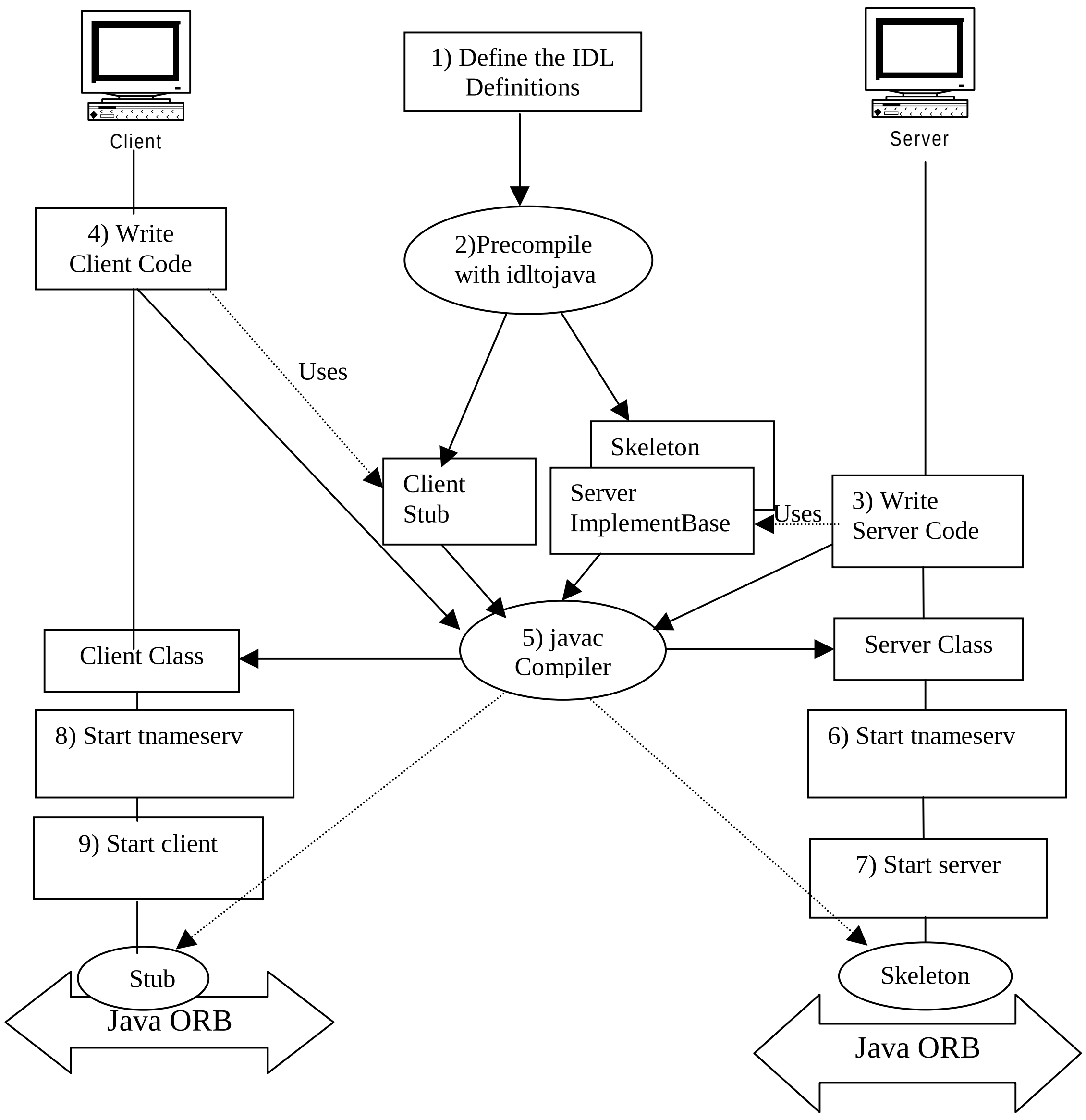
At the current version Java IDL with JDK 1.2 beta4, it supports:

- Supports Static and Dynamic CORBA invocations
- Supports Java IDL Naming Service- a CORBA compliant Name Service. Java IDL also provides a transient *nameserver* to organize objects into a tree-directory structure. Although a transient object disappears when its server process stops running, the object may be implemented to store its state in a file and to re-initialize itself from this file at creation time.
- Support IIOP Client and Server ORB functions.
- Supports transient CORBA objects - objects whose lifetimes are limited by their server process's lifetime

For more information about Java IDL, please visit the web site at: http://www.javasoft.com/

12.1 Development Process

Figure 12-1: Distributed object application development in Java IDL



The graph above shows the steps involved in writing a client and server test program for Java IDL.

1. Define IDL. As usual in the first step for a CORBA application development, we have to define all the service available to be invoked remotely in an OMG's IDL file. Here is the code for it:

```
// Server.idl
module JavaIDLServer {
    interface Server {
    Server createInstance();
    long methodA1(in long a1);
    long methodA2(in long a1, in long a2);
    long methodA3(in long a1, in long a2, in long a3);
    long methodA4(in long a1, in long a2, in long a3, in long a4);
    long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
    long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

    typedef sequence<octet> ByteArray;
    typedef sequence<long> IntArray;
    typedef sequence<double> DoubleArray;

    void methodB(in ByteArray ba);
    void methodI(in IntArray ia);
    void methodD(in DoubleArray da);

    };
};
```

2. Compile the IDL. We need to generate client stub and server skeleton from the IDL file. Enter this command at the command line prompt.

   prompt>idltojava –fno-cpp Server.idl

   By default, the idltojava using C/C++ preprocessor, we can turn it off by adding –fno-cpp option to the command line. The idltojava can be download at this address:
   http://developer.javasoft.com/developer/earlyAccess/jdk12/idltojava.html

3. Write Server code. We use inheritance style to write our server implementation. Here is the code.

```
package DOTBenchmark.javaidl;
import JavaIDLServer.*;  // The package containing our stubs.

import org.omg.CosNaming.*;
// The package containing special exceptions thrown by the name service.
import org.omg.CosNaming.NamingContextPackage.*;

// All CORBA applications need these classes.
import org.omg.CORBA.*;


public class ServerImpl extends _ServerImplBase {

  /** Construct a transient object. */
  public ServerImpl() {
      super();
```

```
}

public static void main(String args[]) {
     try {
           // Create and initialize the ORB
          ORB orb = ORB.init(args, null);

          // Create the servant and register it with the ORB
          ServerImpl serverRef = new ServerImpl();
          orb.connect(serverRef);

          // Get the root naming context
          org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
          NamingContext ncRef = NamingContextHelper.narrow(objRef);



          // Bind the object reference in naming
          NameComponent nc = new NameComponent("Server", " ");
          NameComponent path[] = {nc};
          ncRef.rebind(path, serverRef);

          System.out.println("server was made: "+serverRef);

          // Wait for invocations from clients
          java.lang.Object sync = new java.lang.Object();
          synchronized(sync){
          sync.wait();
          }
     }
     catch (Exception e) {
          e.printStackTrace();
     }
 }

public Server createInstance() {
     ServerImpl obj = null;
     try {
          obj = new ServerImpl();
          return obj;
     } catch (Exception e) {
     e.printStackTrace();
     }
     return null;
}
public int methodA1(int a1) {
  return 0;
}
public int methodA2(int a1,int a2) {
```

```
            return 0;
          }
          public int methodA3(int a1, int a2, int a3) {
            return 0;
          }
          public int methodA4(int a1,int a2,int a3,int a4) {
            // implement operation...
            return 0;
          }
          public int methodA5(int a1,int a2,int a3,int a4,int a5) {
            return 0;
          }
          public int methodA6(int a1,int a2,int a3,int a4,int a5,int a6) {
            return 0;
          }
          public void methodB(byte[] ba ) {
          }
          public void methodI(int[] ia) {
          }
          public void methodD(double[] da) {
          }
}
```

As we can see, the code is very similar to the ServerImpl.java we wrote for VisiBroker test. The only different is in the main method where VisiBroker uses the BOA class to bind Server object to the ORB and wait for connection from client, here we use the Java IDL Naming Service to bind the Server object. We can do this by first retrieving the root naming context which actually very similar to retrieving the root directory on a file system), then by adding the NameComponent to it to bind our ServerImpl object. Once this is done, our server application simply goes into a "wait" mode in order to handle client request.

4. Write the client application. We write the client program to run our test benchmark. Here is the code described the client resolves the object reference using the naming service, the rest of the code after this is exactly the same as for all the DOTs we wrote and therefore is not presented here.

```
// Create and initialize the ORB
    ORB orb = ORB.init(args, null);

// Get the root naming context
    org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
    NamingContext ncRef = NamingContextHelper.narrow(objRef);


/* remote object connection benchmark */
// Resolve the object reference in naming
    NameComponent nc = new NameComponent("Server", " ");
NameComponent path[] = {nc};
Server server = ServerHelper.narrow(ncRef.resolve(path));
```

Here the server object reference is returned to the client. The client can use it to do whatever services which have been honored in the IDL.

5. Compile all the source code. Run Javac to compile ServerImpl.java and Client.java (with option optimization turned on)
6. Start the Java IDL transient Naming service. We need to run the *tnameserv* on the server machine

   prompt>tnameserv
7. Start the Server.

   prompt>java ServerImpl.java
8. Start the Java IDL transient Naming service. We also need to run the tnameserv on the client machine as well.

   prompt>tnameserv
9. Start the client.

   prompt>java Client 100 *servername*

Note that *servername* is the machine name that the server is running on and 100 is the iteration for the test operation.

12.2 Remote Object Connection

The test for the Remote Object Connection measures the time it take the client program to locate the server object which is running on a server machine. Unfortunately, I couldn't run the test on 2 networked machines because of the flaw in resolving object reference for the server object in Java IDL. The table below shows the result of the test on a single machine. Here is the part of code for this test.

```
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    Server server = ServerHelper.narrow(ncRef.resolve(path));
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("bind to server(): "+elapse+" msec");
```

Table 12-1: Remote Object Connection in Java IDL

Remote Object Connection	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java IDL 1.2	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java IDL 1.2
35.1 msec	Not available

The result shows that, the performance of Java IDL in the remote object connection is the slowest of all the DOTs that we compared in the project.

12.3 Remote Object Creation

In this operation, we also simulate the remote object creation as we did for VisiBroker and OrbixWeb. We let the client connect to an existing server object and then call the method createInstance() to return a new server object, as is specified in the code below:

```
/* remote object creation benchmark */
Server server2 = null;
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
        Server server = ServerHelper.narrow(ncRef.resolve(path));
        server2 = server.createInstance();
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("object creation: "+elapse+" msec");
```

The table shows the result of average of 100 time invoke the createInstance() method to create remote server object .

Table 12-2: Remote Object Creation in Java IDL

Remote Object Creation	
Same Machine • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java IDL 1.2	**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java IDL 1.2
12.7 msec	Not available

The result shows us that Java IDL has better performance than VisiBroker and OrbixWeb which has a creation time of 15.9 and 16.5 msec respectively. However, it is a bit slower than HORB and Voyager, which has a creation time of 11.0, and 6.60 msec respectively.

12.4 Remote Method Call

The remote method call evaluates the performance of Java IDL stub and skeleton in marshalling and unmarshalling the argument of the method call. Here is the code for testing this operation in Java IDL. Actually, it is exactly the same for all the DOTs we have written the code for them so far, I listed here just for completeness.

```
Server server = ServerHelper.narrow(ncRef.resolve(path));
//
// remote method call with variable parameters benchmark
//

// one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA1(100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// two arguments
```

```
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA2(100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

// three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA3(100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

// four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA4(100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
    x = server.methodA5(100, 100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

// six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
    x = server.methodA6(100, 100, 100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
```

Table 12-3: Remote Method Call in Java IDL

Remote Method Call with variable arguments							
Environment	**Same Machine** • CPU: 200MHz • RAM: 64 MB SDRAM • OS: Windows 98 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with Symantec's JIT compiler turned on as default • Software: Java IDL 1.2 (Unit time in msec)						**Two Windows NT Machines** • CPU: 2 Pentium 200MHz computers • Network: 100Mbps and TCP/IP (100Base-T) • RAM: 96 MB SDRAM • OS: Windows NT SP3 • Java Compiler: JDK 1.1.6 • Java VM: Sun Java VM with symantec's JIT compiler turned on as default • Software: Java IDL 1.2 (Unit time in msec)
Args	1	2	3	4	5	6	NOT AVAILABLE
Java IDL 1.2	6.1	5.4	6.1	5.5	5.5	5.5	

Looks at this result, we can see that Java the invocation time does not vary much with variable number of argument calls. However, the marshalling and unmarshalling process of the Stub and Skeleton of Java IDL is much slower than the previous DOT we had tested so far.

12.5 Object Data Transfer

This feature of object transfer by value has not yet implemented in Java IDL

12.6 Numerical Array Transfer

As before, we also test the operation of transferring primitive "byte", "int" and "double" using Java IDL. Here is the result for the operation on a single machine

Table 12-4: Numerical Array Transfer in Java IDL on the same machine

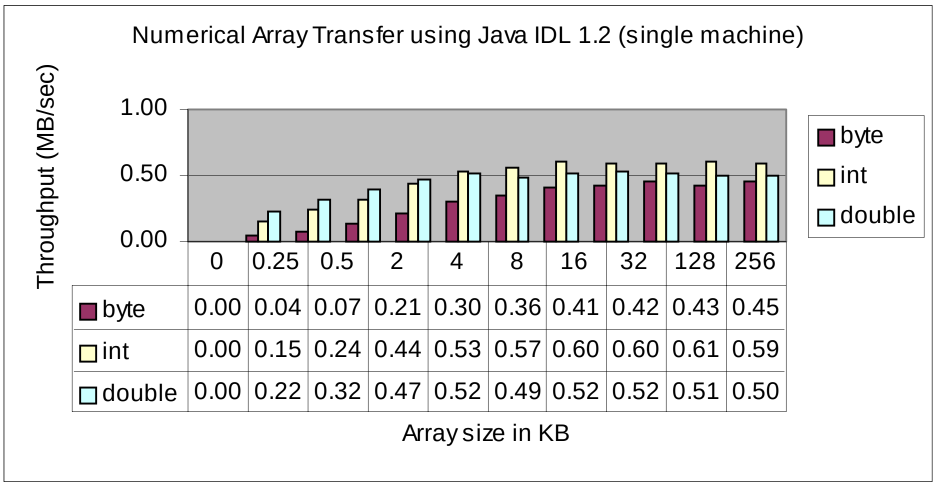
Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default

Software: Java JDK 1.2 beta4

(Unit time in msec)

Size(in byte)	1	256	512.00	1024	2048	4096	8192	16384	32768	65536	131072	262144
byte	3.6	2.81	3.4	3.31	3.6	4.31	9.01	12.02	17.63	31.74	65	109.25
int	3.31	3.51	3.6	4.51	9.51	12.92	21	42.1	83.1	206.3	613.9	1446.1
double	3.41	3.5	4.61	8.21	11	18	36	72.1	168.3	434.6	1250.8	4062.9

Figure 12-2: Numerical Array Transfer using Java IDL on the same machine



	0	0.25	0.5	2	4	8	16	32	128	256
byte	0.00	0.04	0.07	0.21	0.30	0.36	0.41	0.42	0.43	0.45
int	0.00	0.15	0.24	0.44	0.53	0.57	0.60	0.60	0.61	0.59
double	0.00	0.22	0.32	0.47	0.52	0.49	0.52	0.52	0.51	0.50

The result shows that:

- The peak throughput is low for this operation. For transferring "byte", it reaches a peak of 0.45 MB/sec at the array size of 256 KB, for "int" of 0.61 at the array size of 128 KB and "double" of 0.52 MB/sec at the array sizes of 16 and 32 KB.
- Transferring "int" and "double" array shows higher throughput than transferring "byte" array, this has not been seen for all previous DOTs we have done so far where throughput for "byte" array transferring is always higher than "int" and "double" array.
- Marshalling and Unmarshalling process in Java IDL is much slower than the other DOTs we have tested, in fact it is the slowest of all.

12.7 Garbage Collection

I don't know whether Java IDL supports Distributed Garbage Collection or not, I suspect it also uses the distributed garbage collection mechanism just as in the Java RMI.

13. Socket Implementation

13.1 *Java Socket*

In order to see how good the performance of DOTs in transferring the array of primitive comparing to the plain Socket version counterpart, a Socket version for Java and C is developed to test this feature. For a complete code of Client/Server Java Socket version, please refer to the appendix.

Table 13-1: Numerical Array Transfer Using Socket Java on the same machine

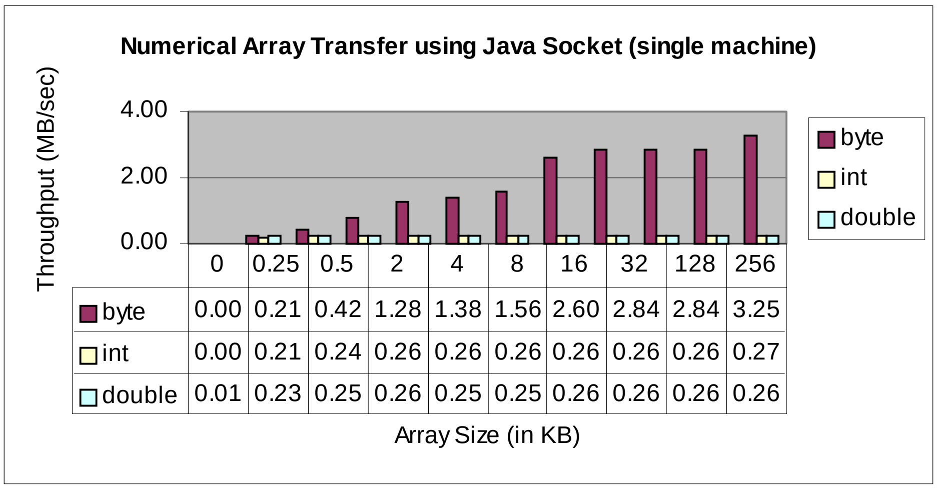
Same Machine

- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6

(Unit time in msec)

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	0.94	0.94	1.04	0.00	0.00	0.01
0.25	1.15	4.61	8.35	0.21	0.21	0.23
0.5	1.15	8.18	15.7	0.42	0.24	0.25
1	1.27	15.3	30.7	0.77	0.26	0.25
2	1.53	30	60.9	1.28	0.26	0.26
4	2.83	61	127	1.38	0.26	0.25
8	5	120	247	1.56	0.26	0.25
16	6	236	489	2.60	0.26	0.26
32	11	478	966	2.84	0.26	0.26
64	22	945	1934	2.84	0.26	0.26
128	44	1889	3861	2.84	0.26	0.26
256	77	3757	7711	3.25	0.27	0.26

Figure 13-1: Numerical Array Transfer using Socket Java on the same machine



	0	0.25	0.5	2	4	8	16	32	128	256
byte	0.00	0.21	0.42	1.28	1.38	1.56	2.60	2.84	2.84	3.25
int	0.00	0.21	0.24	0.26	0.26	0.26	0.26	0.26	0.26	0.27
double	0.01	0.23	0.25	0.26	0.25	0.25	0.26	0.26	0.26	0.26

The result shows that:

- Transferring "byte" array has throughput increased gradually, it reaches its peak 3.25 MB/sec at the array size of 256 KB
- Transferring "int" and "double" become saturated at the throughput of 0.26 MB/sec, which corresponds to the array size of 2 KB
- The throughput for transferring "byte" array is much higher than the throughput for transferring "int" and "double" array. This indicates than Socket Java handles very bad in transferring "int" and "double" array.

Table 13-2: Numerical Array Transfer Using Socket Java on 2 NT machines

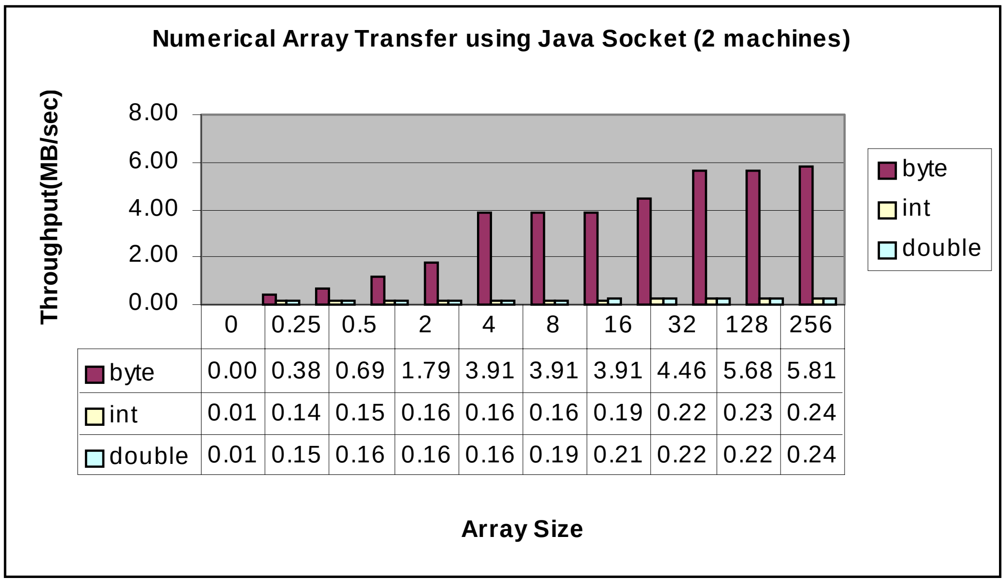
Two Windows NT Machines

- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Java JDK 1.1.6

(Unit time in msec)

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	0.611	0.591	0.641	0.00	0.01	0.01
0.25	0.641	6.86	13.199	0.38	0.14	0.15
0.5	0.711	12.798	25.136	0.69	0.15	0.16
1	0.841	24.746	51.604	1.16	0.16	0.15
2	1.092	48.87	98.802	1.79	0.16	0.16
4	1	95.2	197.3	3.91	0.16	0.16
8	2	194.3	328.5	3.91	0.16	0.19
16	4	322.4	591.9	3.91	0.19	0.21
32	7	579.9	1112.6	4.46	0.22	0.22
64	11	1089.5	2203.1	5.68	0.23	0.23
128	22	2213.2	4541.6	5.68	0.23	0.22
256	43	4162	8433.1	5.81	0.24	0.24

Figure 13-2: Numerical Array Transfer Using Socket Java on 2 NT machines



	0	0.25	0.5	2	4	8	16	32	128	256
■ byte	0.00	0.38	0.69	1.79	3.91	3.91	3.91	4.46	5.68	5.81
□ int	0.01	0.14	0.15	0.16	0.16	0.16	0.19	0.22	0.23	0.24
□ double	0.01	0.15	0.16	0.16	0.16	0.19	0.21	0.22	0.22	0.24

The result shows that:

- Throughput for transferring "byte" array on 2 NT machines reaches its peak at 5.81 MB/sec, which is nearly double the throughput for transferring "byte" array on the same machine.
- Throughput for transferring "byte" array is much higher than throughput for transferring "int" and "double" array, this confirmed the inefficient handling the "int" and "double" marshalling and unmarshalling in Socket Java.

13.2 C Socket

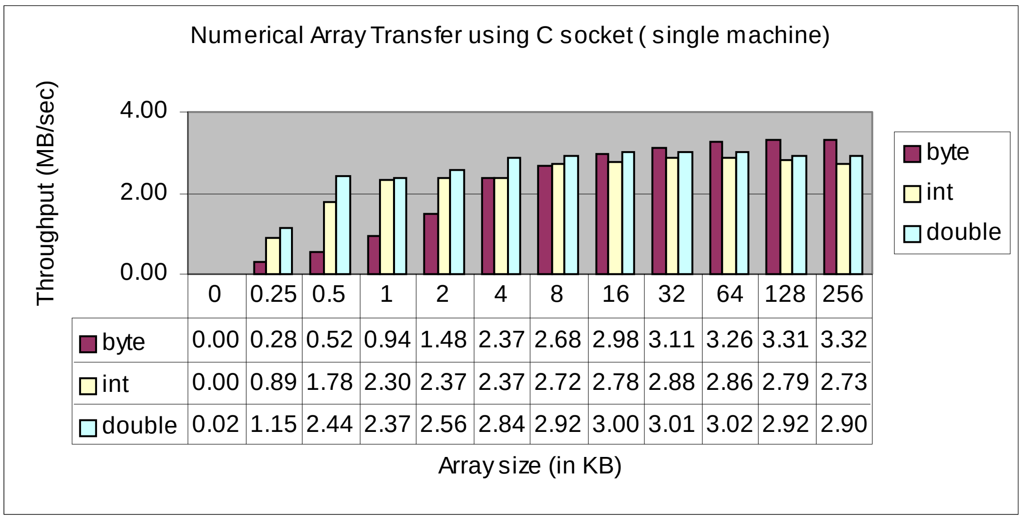
Here we test the numerical array transfers using Socket C version. For complete code detail, please refer to the appendix section of this report.

Table 13-3: Numerical Array Transfer using Socket C on the same machine

Same Machine
- CPU: 200MHz
- RAM: 64 MB SDRAM
- OS: Windows 98
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Borland C++ 5.02 compiler

Array Size in KB	Time (msec)			Throughput (MB/sec)		
	byte	int	double	byte	int	double
0	0.71	1.1	0.5	0.00	0.00	0.02
0.25	0.88	1.1	1.7	0.28	0.89	1.15
0.5	0.94	1.1	1.6	0.52	1.78	2.44
1	1.04	1.7	3.3	0.94	2.30	2.37
2	1.32	3.3	6.1	1.48	2.37	2.56
4	1.65	6.6	11	2.37	2.37	2.84
8	2.91	11.5	21.4	2.68	2.72	2.92
16	12.63	22.5	41.7	1.24	2.78	3.00
32	10.05	43.4	83	3.11	2.88	3.01
64	19.17	87.3	165.3	3.26	2.86	3.02
128	37.79	179.1	342.2	3.31	2.79	2.92
256	75.41	366.9	688.7	3.32	2.73	2.90

Figure 13-3: Numerical Array Transfer using Socket C on the same machine



	0	0.25	0.5	1	2	4	8	16	32	64	128	256
byte	0.00	0.28	0.52	0.94	1.48	2.37	2.68	2.98	3.11	3.26	3.31	3.32
int	0.00	0.89	1.78	2.30	2.37	2.37	2.72	2.78	2.88	2.86	2.79	2.73
double	0.02	1.15	2.44	2.37	2.56	2.84	2.92	3.00	3.01	3.02	2.92	2.90

The result shows that:

- Throughput for transferring numerical array increases with the array size from 0 to 256 KB
- Throughput for transferring of "int" and "double" is higher than throughput of transferring "byte" array for the array size from 0 to 8 KB; after that throughput of "byte" array transfer is higher.
- Throughput for "byte" array transfer reaches it peak of 3.32 MB/sec at the array size of 256 KB, "int" of 2.88 at the array size of 32 KB and "double" of 3.02 at the array size of 64 KB
- Socket C can handle marshalling and unmarshalling process equally well for "byte", "int" and "double" array transfer.

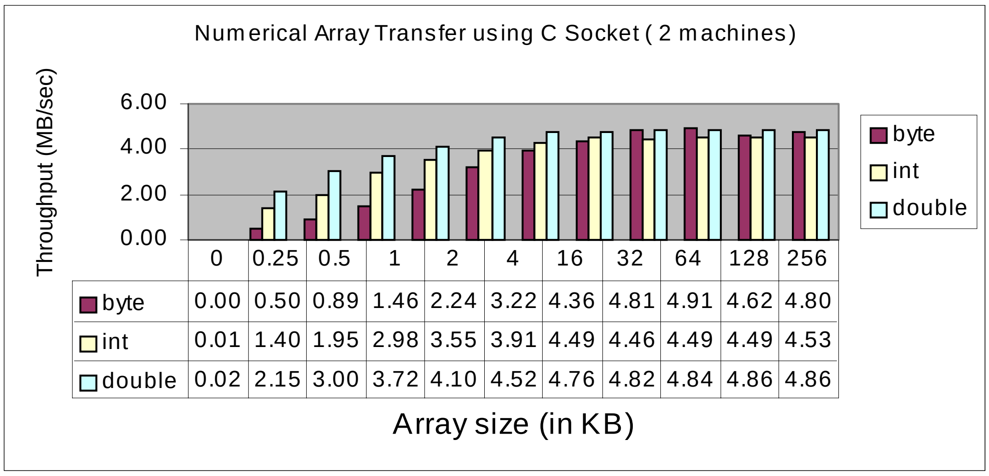
Table 13-4: Numerical Array Transfer using Socket C on 2 NT machines

Two Windows NT Machines

- CPU: 2 Pentium 200MHz computers
- Network: 100Mbps and TCP/IP (100Base-T)
- RAM: 96 MB SDRAM
- OS: Windows NT SP3
- Java Compiler: JDK 1.1.6
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default
- Software: Borland C++ 5.02 compiler

Array Size in KB	**Time (msec)**			**Throughput (MB/sec)**		
	byte	**int**	**double**	**byte**	**int**	**double**
0	0.451	0.5	0.4	0.00	0.01	0.02
0.25	0.491	0.7	0.91	0.50	1.40	2.15
0.5	0.551	1	1.3	0.89	1.95	3.00
1	0.671	1.31	2.1	1.46	2.98	3.72
2	0.871	2.2	3.81	2.24	3.55	4.10
4	1.212	4	6.91	3.22	3.91	4.52
8	1.992	7.32	13.01	3.92	4.27	4.80
16	3.586	13.92	26.24	4.36	4.49	4.76
32	6.499	28.04	51.88	4.81	4.46	4.82
64	12.73	55.68	103.25	4.91	4.49	4.84
128	27.08	111.5	205.69	4.62	4.49	4.86
256	52.09	220.6	411.19	4.80	4.53	4.86

Figure 13-4: Numerical Array Transfer using Socket C on 2 NT machines



	0	0.25	0.5	1	2	4	16	32	64	128	256
byte	0.00	0.50	0.89	1.46	2.24	3.22	4.36	4.81	4.91	4.62	4.80
int	0.01	1.40	1.95	2.98	3.55	3.91	4.49	4.46	4.49	4.49	4.53
double	0.02	2.15	3.00	3.72	4.10	4.52	4.76	4.82	4.84	4.86	4.86

The result shows that:

- The throughput patterns for transferring numerical array on 2 networked machines does not change comparing to the single machine test.
- Throughput for transferring numerical array on 2 machines across the network is higher than throughput for transferring numerical array on the same machine. This indicates that the marshalling and unmarshalling process is a CPU intensive operation.

14. Comparison

In this section, I compare the benchmark of all DOT packages based on the evaluation criteria I have discussed on the section 5. The comparison is for operation on a single machine as well as on 2 NT machines. The specific operation of each DOT is also listed.

Platforms used:

- Windows 98:
  - CPU: 200 MHz Intel Pentium
  - RAM: 64 MB SD RAM
- Windows NT 4.0 SP3 Workstation
  - CPU: 200 MHz Intel Pentium
  - RAM: 96 MB SD RAM
- Network:
  - 100 Base-T Ethernet LAN
- Java Compiler: JDK 1.1.6 javac compiler, MS-SDK 3.1jvc compiler
- Java VM: Sun Java VM with Symantec's JIT compiler turned on as default, MS-SDK jview
- DOT Packages:
  - Sun's JDK 1.1.6 for RMI
  - HORB version 1.3b1 (with serialization patch)
  - Voyager 2.0.0 Production Release
  - Inprise's VisiBroker 3.3 for Java
  - Iona's OrbixWeb 3.0 for Java
  - Microsoft's Java SDK 3.1 for DCOM
  - Sun's JDK 1.2 for Java IDL

14.1 Remote Object Connection

Remote Object Connection benchmark tests the time it takes a client program to get a reference to an existing object which is running on the server a priori. The vertical axis shows the time that a DOT executes in msec. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-1: Remote Object Connection for DOTs on the same machine

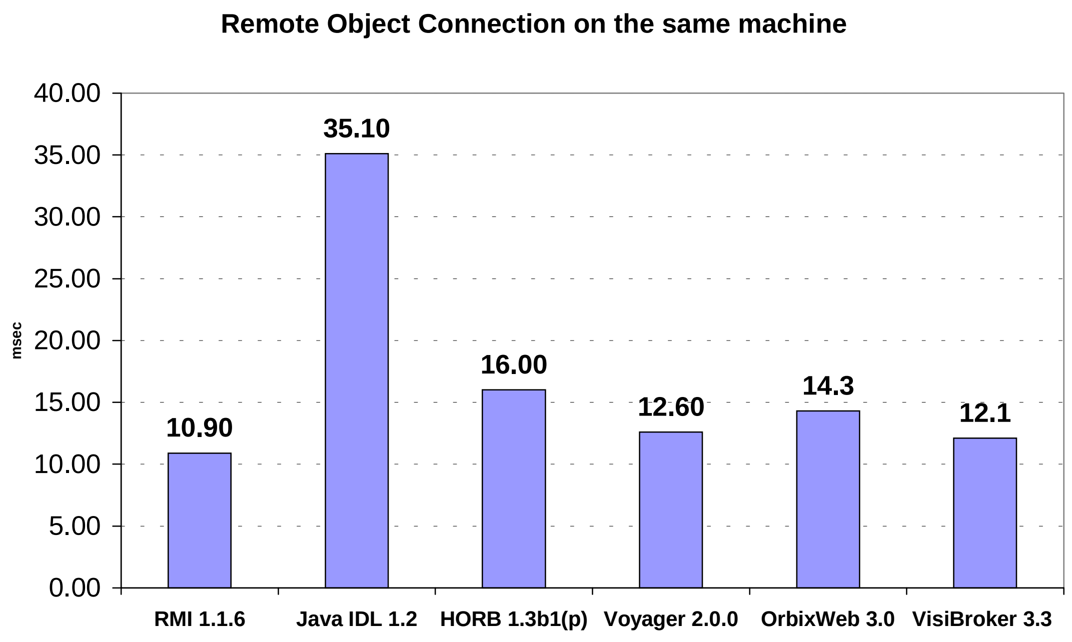
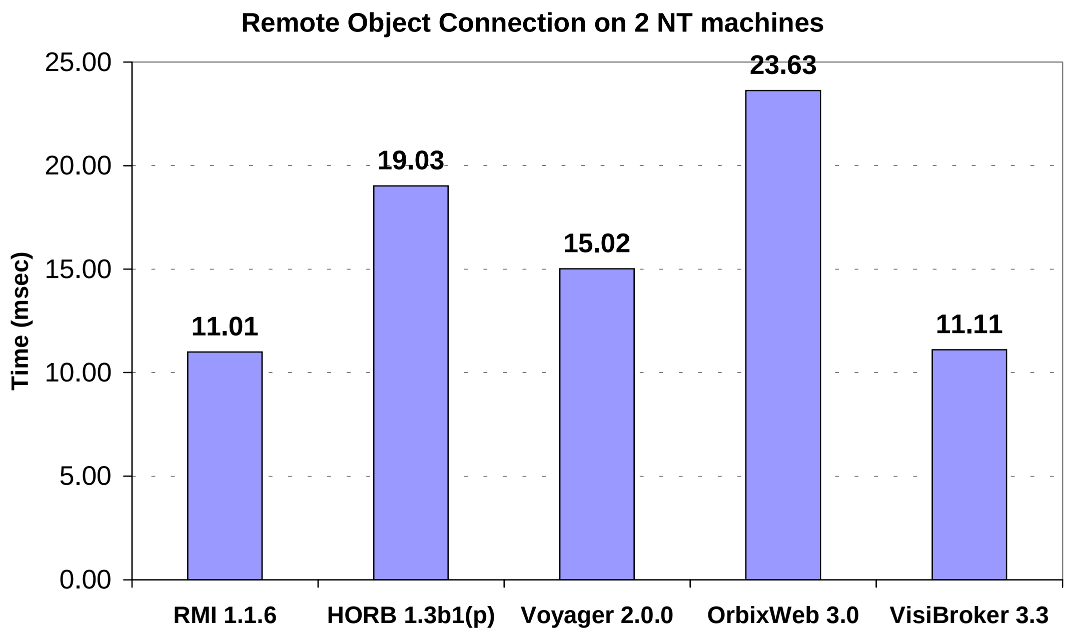


Figure 14-2: Remote Object Connection for DOTs on 2 NT machines



The figures show that:

- On a local machine, all DOTs have nearly the same performance except for Java IDL which shows the longest time. Java RMI won the benchmark for this case
- For a test over the network, OrbixWeb is the slowest. I can't get Java IDL run over the network. Java RMI still won the benchmark in this case.
- Most of the results collected over the network is slower than result collected on the local machine for this operation and this can be understood because it requires more effort to locate an object over the network than on the same machine. However, be warned that this is not always true for e.g. in the case of VisiBroker, with the help of a daemon service called osagent which helps to efficiently locate objects anywhere on the network, can indeed show better performance on over the network than on a local machine
- All DOT except OrbixWeb shows only less than 5% amount of time penalty for locating a server object over the network to a local machine, for OrbixWeb it takes more than 50% amount of time to locate a server object over the network to a local machine.
- All DOTs except HORB and Java RMI employs connection multiplexing; i.e. one TCP/IP connection is used to carry more than one connect message requested from the client object. Connection multiplexing saves the startup cost for creating each TCP/IP for each connection, however if there are two many connections, the connection will become bottleneck and slow down the speed of accessing remote object from the client. The result above does show that the time for creating a new TCP/IP connection is negligible. Therefore, connection multiplexing is a good choice. It is particularly important in distributed system that requires dedicated bandwidth as much as possible.
- DCOM not included in this test because it is not appropriate (please refer to the DCOM to find out why)

Each DOT has each own way to return an proxy (stub) to the existing object on the server, below is the specific operation in which a client object tries to connect to an object name “Server” which located on the server machine.

Table 14-1: Remote Object Connection Method used by DOTs

Remote Object Connection	
DOT	Implementation
Java RMI	Naming.lookup(“hostname/Server”)
HORB	new Server_Proxy(“hostname”, “Server”);
Voyager	Namespace.lookup(“hostname/Server”)
OrbixWeb	ServerHelper.bind(“:Server”, hostname);
VisiBroker	ServerHelper.bind(orb, “Server”)
Java IDL1.2	

Note*: orb is the Object Request Broker which can be obtained by calling this method:org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv, null);*

14.2 Remote Object Creation

Remote Object Creation benchmark in this project is the time it takes a client to create a new remote server objects from a existing one. Not all DOTs have equipped this kind of operation, therefore a simulating Remote Object Creation for them is used, these DOTs include: Java RMI, Java IDL, OrbixWeb and VisiBroker. The vertical axis shows the time that a DOT executes in msec. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-3: Remote Object Creation on the same machine

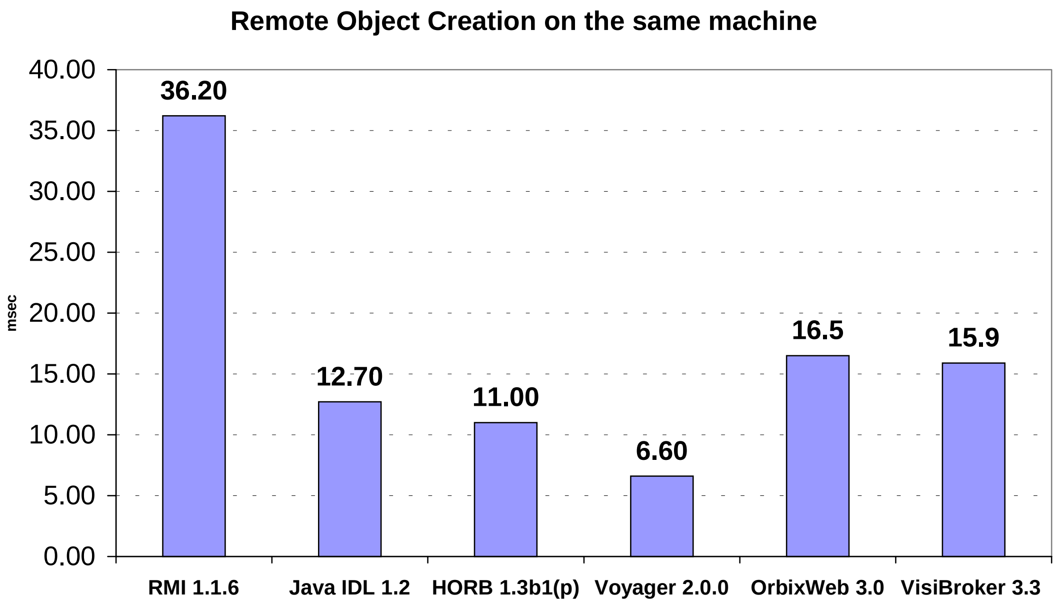
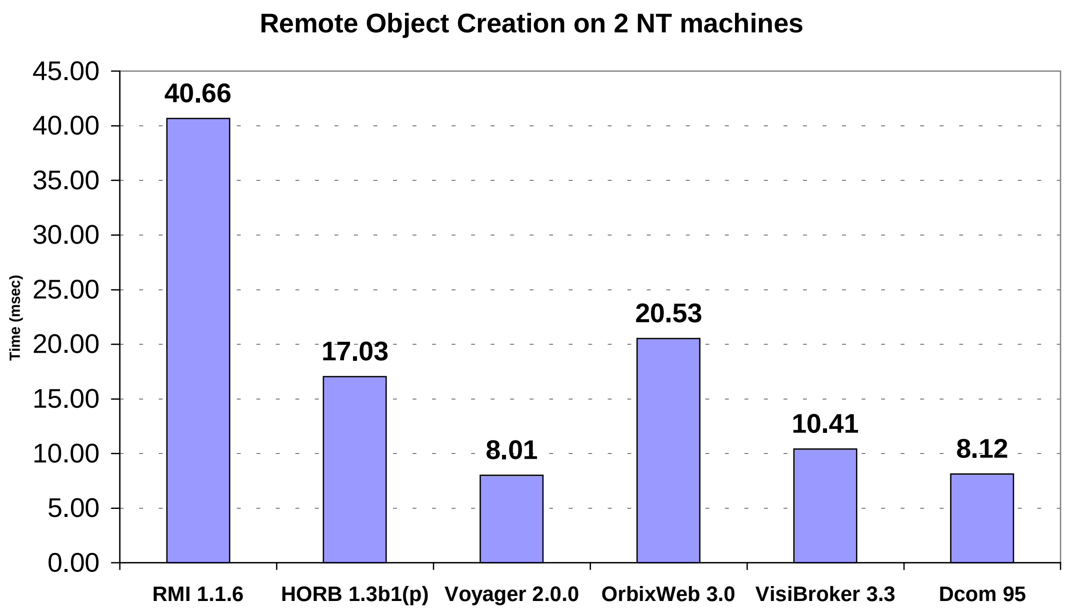


Figure 14-4: Remote object creation on 2 NT machines



The figures show that:

- Time to create new remote server objects varies different for each DOT, this reflects different mechanism used by each DOT to carry out this operation. DOT which is equipped its own method to create remote object shows better performance than those DOTs don't have, which we use simulating method to test them out.
- On a local machine, Java RMI has the longest time and Voyager has the shortest time. Voyager is about 6 times faster than Java RMI. Interestingly, the Java IDL CORBA ORB version shows better performance than two big commercial CORBA ORB vendors do, VisiBroker and OrbixWeb.
- Over the network, Java RMI is still the slowest. DCOM is almost as fast as Voyager.
- Creation benchmark for DOTs over the network is generally slower than on a local machine. This fact again does not apply to VisiBroker where benchmark over the network is better than benchmark on a local machine in the object creation operation.
- I can't get Java IDL run over the network and I cant' get DCOM run out-of-process on a local machine, that why they are missing in the figure above.
- Although Voyager uses Object Reflection to create object dynamically, this does not degrades its performance. In fact it won benchmark for this operation for both cases, over the network and on a local machine.

HORB, Voyager and DCOM has a its own function to create a remote object on the server, which is listed in the table below. For those DOTs do not implement a function to dynamically create remote server object, it call the method createInstance() instead.

Table 14-2: Remote Object Creation methods used by DOTs

Remote Object Creation	
DOT	**Implementation**
Java RMI 1.1.6	server.createInstance()
HORB 1.3b1	server.Server()
Voyager 2.0.0	Factory.create(“Server”)
OrbixWeb 3.0	server.createInstance()
VisiBroker 3.3 for Java	server.createInstance()
DCOM 95	new ServerConnect()
Java IDL 1.2	server.createInstance();

14.3 Remote Method Call

Remote Method Call benchmark measures the time it takes the DOT in marshalling and unmarshalling a variable number of argument calls of the method invocation. The argument has an "int" type. There are six operations, with the first method corresponds to one argument and the sixth method corresponds to 6 arguments. The vertical axis shows the time that a DOT executes in msec. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-5: Remote Method Call on the same machine

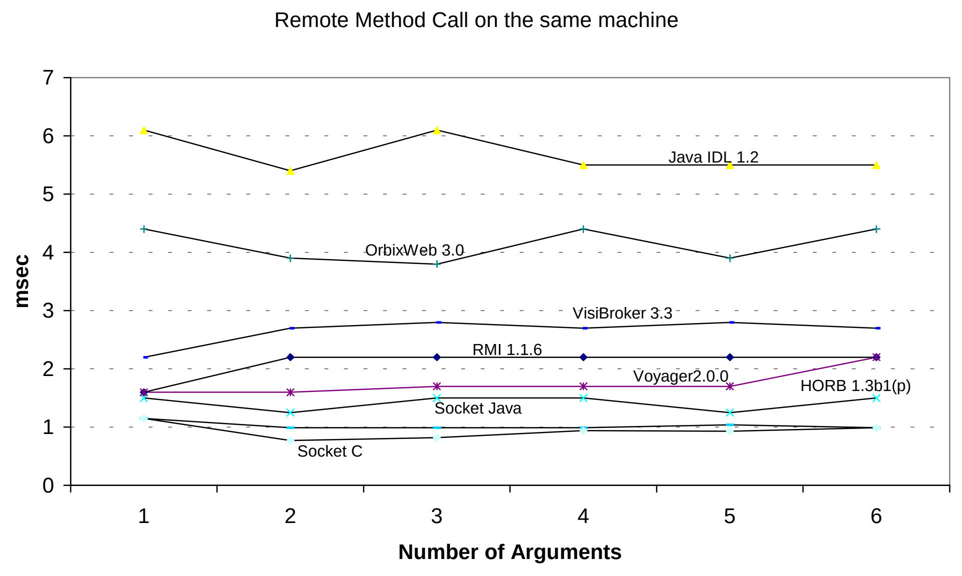
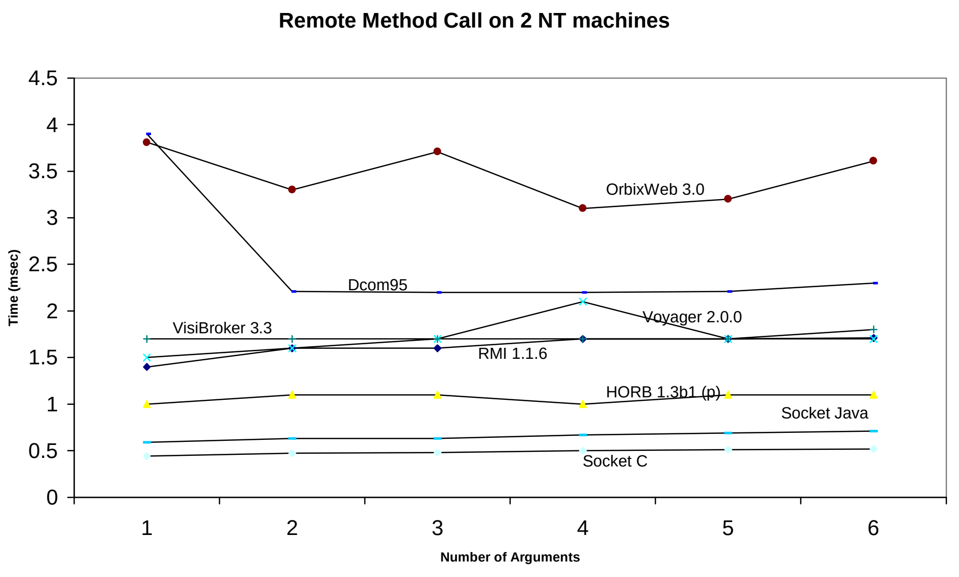


Figure 14-6: Remote Method Call on 2 NT machines



Remote Method Call with 3 arguments:

Figure 14-7: Remote Method Call with 3 arguments on the same machine

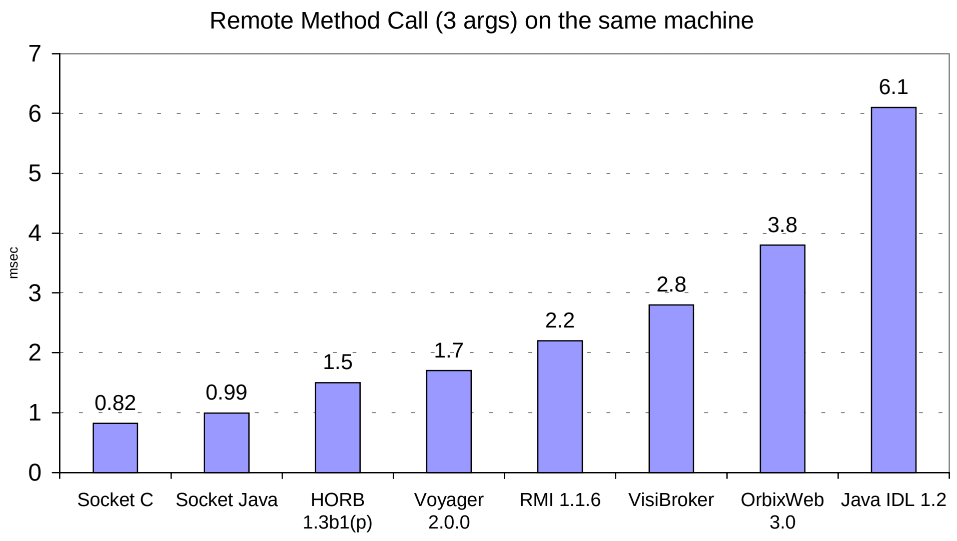
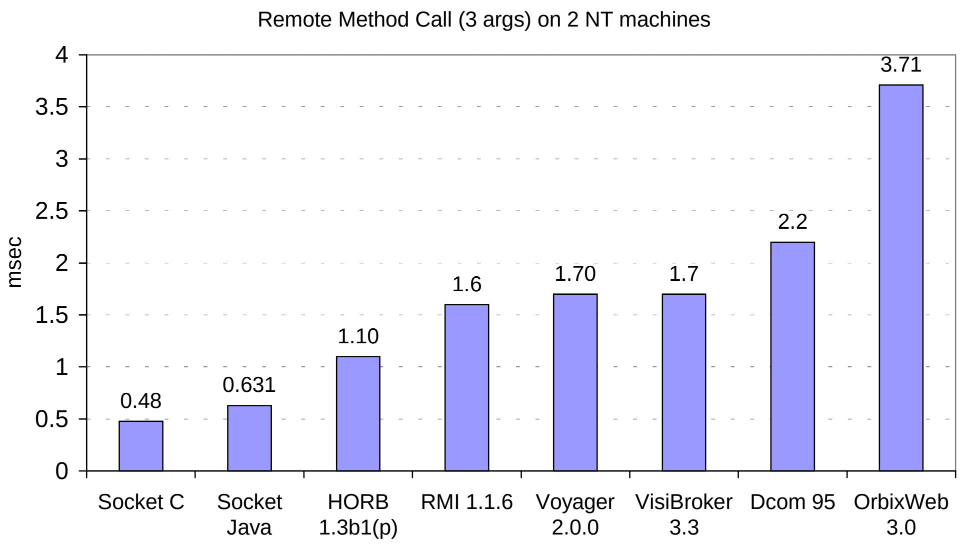


Figure 14-8: Remote Method Call with 3 arguments on 2 NT machines



The figures and the graphs show that:

- Amount of time does not change much with variable argument calls; however, it does show the fact that calling method with more arguments takes a little bit longer because of the marshalling and unmarshalling the method calls and its arguments.
- On the same machine, Java IDL is the slowest of all. On 2 NT machines, OrbixWeb is the slowest. If the Java IDL can run over the network, the result of this may give a new ordering
- Local machine operations may not always faster than network machines operation. This can be seen in this benchmark. It indicates that marshalling and unmarshalling is a CPU extensive process, more CPU resources give better performance.
- No doubt that DOT adds overhead in making method call. This is confirmed that Socket version (C or Java) is much faster than method call in DOT
- In this benchmark, HORB gives the fast performance of all DOTs but it's still about 60% slower than Socket C version or 40% slower than Socket Java version

Here are the list of methods called by our DOTs in this project

Table 14-3: Remote Method Call methods used by DOTs

Remote Method Call	
DOT	**Implementation**
Java RMI 1.1.6	The method call is the same for all DOTs
HORB 1.3b1	server.methodA1(100);
Voyager 2.0.0	server.methodA2(100,100);
OrbixWeb 3.0	server.methodA3(100,100,100);
VisiBroker 3.3 for Java	server.methodA4(100,100,100,100);
DCOM 95	server.methodA5(100,100,100,100,100);
Java IDL 1.2	server.methodA6(100,100,100,100,100,100);

14.4 Object Data Transfer

Object Data Transfer benchmark measures the time it takes a client program to transfer array of object by value (or by copy as another name for this operation). As the moment, the object data for transferring is very simple. This data object just encapsulates a non-private data field of type int. The main point affected this operation is the process of handling of object serialization by DOTs. Although some DOTs can use their own version of Serialization or Java Serialization process, the benchmark for object transfer by value evaluated in this project measures **only** how well each DOT performs the serialization process which all relies on the Java RMI serialization system. As we realized that, transferring object by value preserves object state, i.e. the shape of object graphs are kept when transferring. This concept is very important because it gives programmers more flexible way to handle object polymorphism in OO feature. Polymorphism means that we can use a variable of a superclass type to hold a reference to an object whose class is the superclass or any of its subclasses. Therefore, it would be an interesting test for transferring complex objects to see how efficiently and correctly the DOT handles this kind of operation; however, I haven't tested this feature yet.

These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-9: Object Data Array Transfer on the same machine

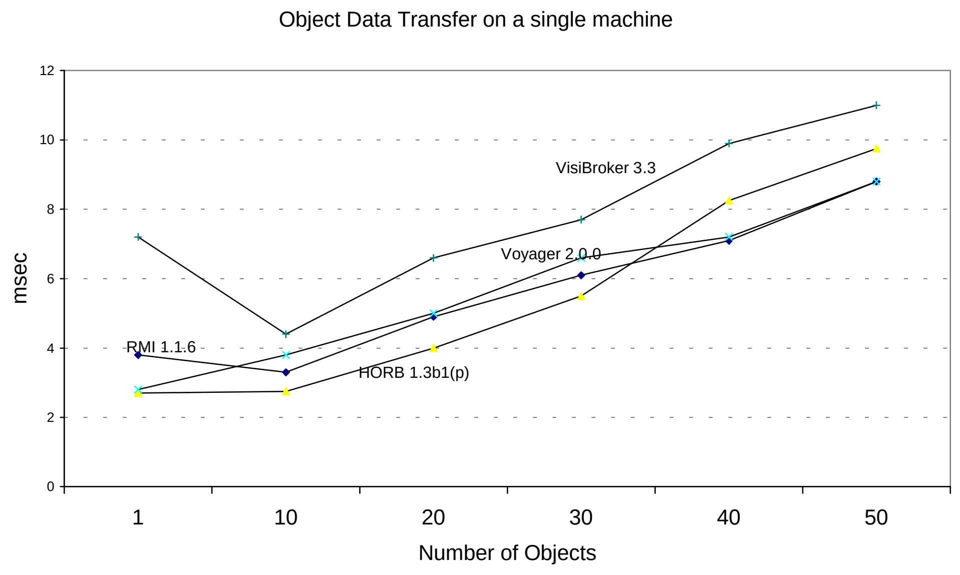
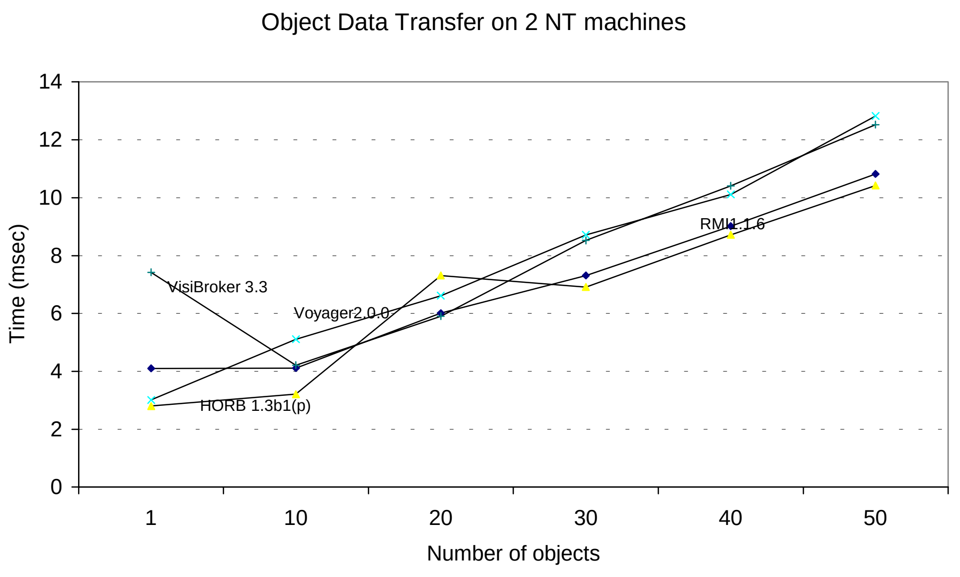


Figure 14-10: Object Data Array Transfer on 2 NT machines



The figures show that:

- There is a dependency between the number of objects and the time for creating the object from the client program and transferring to the server. This indicates the proxy object at the client side needs to perform more work in marshalling and unmarshalling the data for transferring.
- It takes longer time period to invoke the operation for the first time to transfer one object and all the operation for the subsequent times when more objects are transferred. This shows that the cache mechanism is used in all DOTs for transferring data objects, it is clearly shown in VisiBroker case; however, how it is implemented is not clear.
- On a local machine, HORB shows very good performance than the other DOTs. Of all these DOTs, VisiBroker is the slowest one. For transferring more than 30 data objects, Voyager and RMI shows a little bit better performance than HORB
- Over the network, HORB shows a better performance than the other DOTs in handling bigger number of objects transferred. However, at the point of 20 data objects is transferred, it performs worse than the other DOTs
- Performance of Voyager and Java RMI are very close together in this respect. This suggests that Voyager serialization process is tightly integrated with Java Serialization System. For HORB, it shows a smarter way to use Java RMI Serialization process.

Here I list the methods that was used by the DOTs for transferring object data in this project

Table 14-4: Object Data Array Transfer method used by DOTs

Object Transfer by value	
DOT	**Implementation**
Java RMI 1.1.6	server.methodDA(data);
HORB 1.3b1	server.methodDA(data);
Voyager 2.0.0	server.methodDA(data);
OrbixWeb 3.0	Not Support object transfer by value
VisiBroker 3.3 for Java	server.methodDA(data) but need to use javaiiop
DCOM 95	Not Support
Java IDL 1.2	Not Support

14.5 Numerical Array Transfer

Here I compare the benchmark for operations of transferring array of primitive data type, these are "byte", "int" and "double" using DOTs and plain socket version (C and Java socket). The vertical axis represents the throughput in MB/sec, the horizontal axis represents the array size in KB. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

14.4.1 Primitive byte array transfer

Figure 14-11: Primitive byte array transfer on the same machine

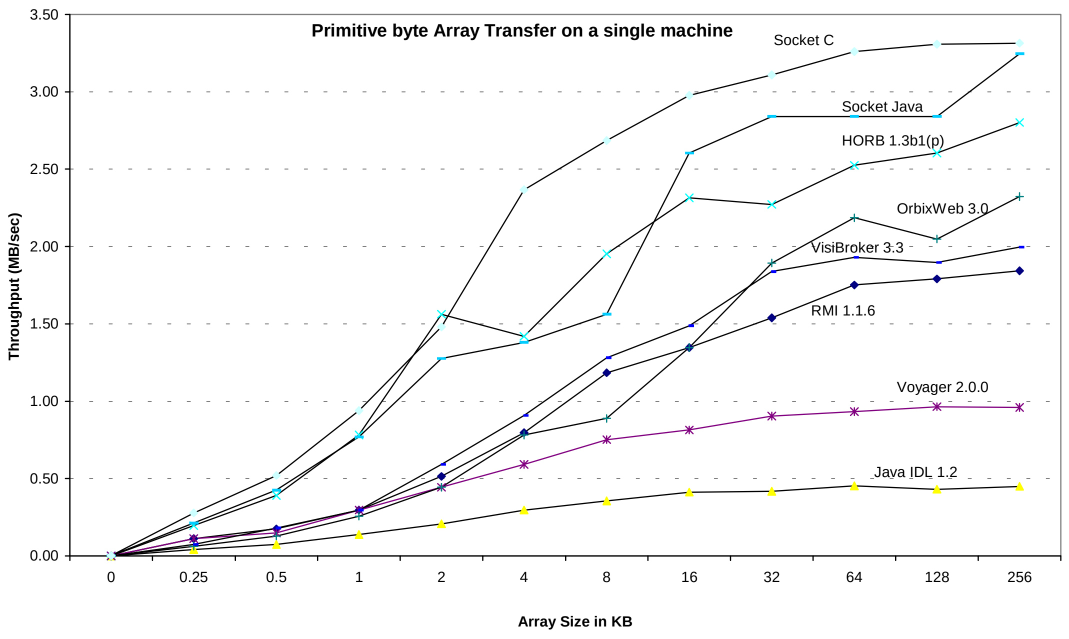
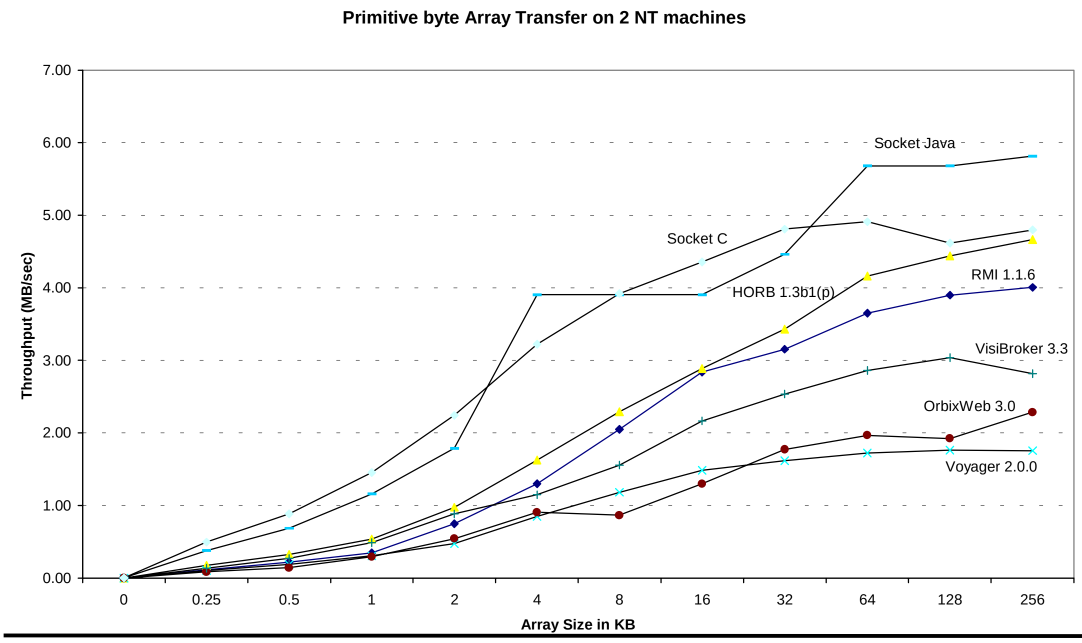


Figure 14-12: Primitive byte array transfer on the same machine



The figures show that:

- For operation on local machine, socket C and socket Java show very good performance in which socket C is the fastest. HORB is the only DOT that can compare to the socket version in "byte" array transfer. The throughput for other DOTs is very low compare to socket version. The lowest is Java IDL which has a peak throughput about 1/5 of the peak throughput of socket C. The DOT version of transferring numerical is not efficient because of the overhead involved, such as the wrapping primitive data to the corresponding class.
- For operation over network machines, transferring "byte" with socket version also shows very good performance. Socket Java shows better performance than the socket C counterpart. HORB again has the highest throughput comparing to the other DOTs.
- Throughput over the network is higher than throughput on a local machine. This indicates that the marshalling and unmarshalling of data is a very intensive CPU operation.

Table 14-5: Primitive byte transfer method used by DOTs

Byte Array Transfer Method Call	
DOT	**Implementation**
Java RMI 1.1.6	The method call is the same for all DOTs server.methodB(*bytearray*)
HORB 1.3b1	
Voyager 2.0.0	
OrbixWeb 3.0	
VisiBroker 3.3 for Java	
DCOM 95	
Java IDL 1.2	

14.4.2 Primitive int array transfer

Here I compare the benchmark for operations of transferring array of "int" using DOTs and plain socket version (C and Java socket). The vertical axis represents the throughput in MB/sec, the horizontal axis represents the array size in KB. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-13: Primitive "int" array transfer on the same machine

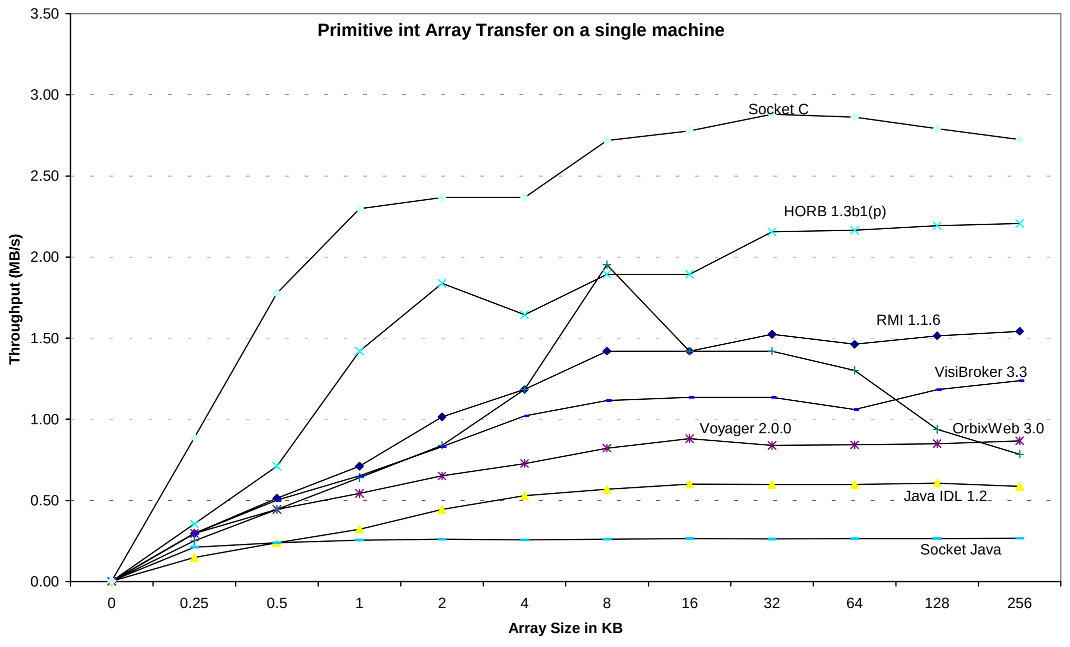
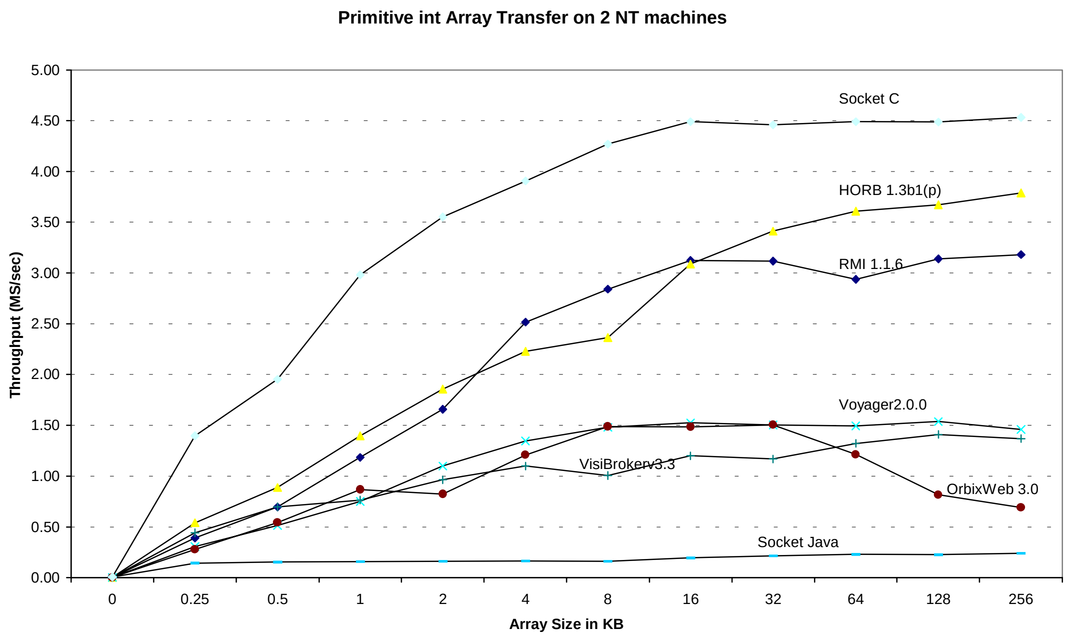


Figure 14-14: Primitive "int" array transfer on 2 NT machines



The figures show that:

- The socket C has very good throughput, it is still the fastest of all, both on the same machine as well as over the network. The socket Java shows the worst of all in this respect. Among the DOTs, HORB still is the fastest.
- Although HORB uses Java serialization in this test to perform marshalling and unmarshalling data, it shows better performance than Java RMI for both operation on the same machine as well as over the network.
- Transfer rate over the network is higher than transfer rate on a local machine, this again indicates that intensive CPU involved in marshalling and unmarshalling data transfer
- Socket C reaches its saturated transfer at about 20% higher than the saturated transfer rate using HORB for both operations on the same machine as well as over the network. This indicates HORB has an efficient marshalling and unmarshalling data, the different of the transfer rate between HORB in Socket C is the amount of overhead that HORB added to the operations.

Here I listed the method that was called by each DOT for transferring "int" array

Table 14-6: Primitive "int" array transfer method called by DOTs

Primitive "int" array transfer method	
DOT	**Implementation**
Java RMI 1.1.6	The method call is the same for all DOTs server.methodI(*intarray*)
HORB 1.3b1	
Voyager 2.0.0	
OrbixWeb 3.0	
VisiBroker 3.3 for Java	
DCOM 95	
Java IDL 1.2	

14.4.3 Primitive double array transfer

Here I compare the benchmark for operations of transferring array of "double" using DOTs and plain socket version (C and Java socket). The vertical axis represents the throughput in MB/sec, the horizontal axis represents the array size in KB. These operations are repeated for 100 iterations and the average elapse time is collected. The test is carried out for client and server hosted on the same machine as well as on two machines connected via an Ethernet LAN network.

Figure 14-15: Primitive "double" array transfer on the same machine

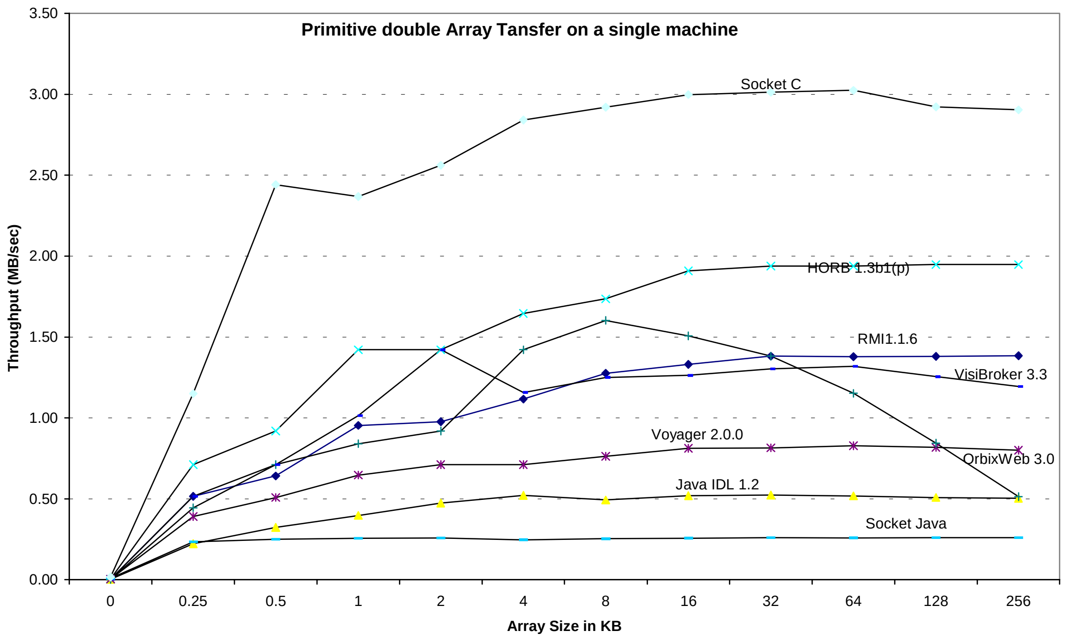
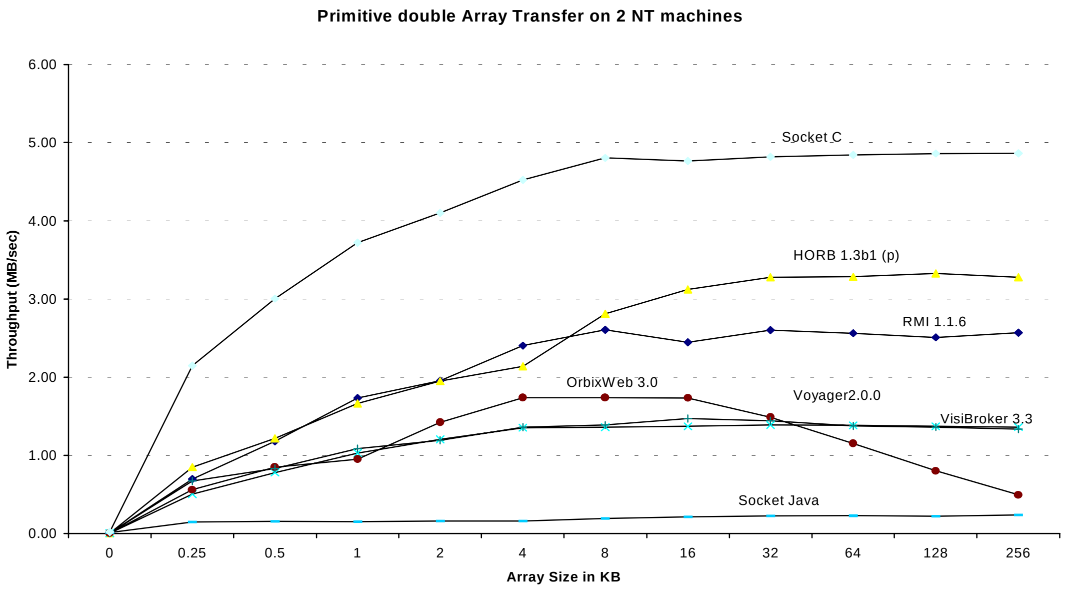


Figure 14-16: Primitive "double" array transfer on 2 NT machines



The figures show that:

- Socket C still shows its superiority in the transferring numerical data. HORB shows very good throughput for operation on a local machine as well as over the network
- For transferring "double" array, HORB has a transfer rate of about 35% lower than the socket C. This indicates that "double" data requires more processing than "int", "byte"
- On a local machine, Voyager has much lower throughput than VisiBroker, but over the network the transfer rate of both two is almost the same. This indicates that Voyager requires more CPU resource than VisiBroker in transferring double array.

Here is the method called by each DOT for "double" array transfer benchmark.

Table 14-7: Primitive "double" array transfer method called by DOTs

Remote Method Call	
DOT	**Implementation**
Java RMI 1.1.6	The method call is the same for all DOTs server.methodD(*doublearray*)
HORB 1.3b1	
Voyager 2.0.0	
OrbixWeb 3.0	
VisiBroker 3.3 for Java	
DCOM 95	
Java IDL 1.2	

14.6 Garbage Collection Mechanism

In this section, I just listed the mechanism used in the distributed garbage collection for each DOT. To compare which distributed garbage collection method is more efficient than the other and whether it affects the benchmark operations that we discussed in this project is outside the scope of this project. Distributed garbage collection is still an active research topics in academic in search for a better way to provide better handling system resource in distributed computing.

Table 14-8: Distributed Garbage Collection mechanism used in DOTs

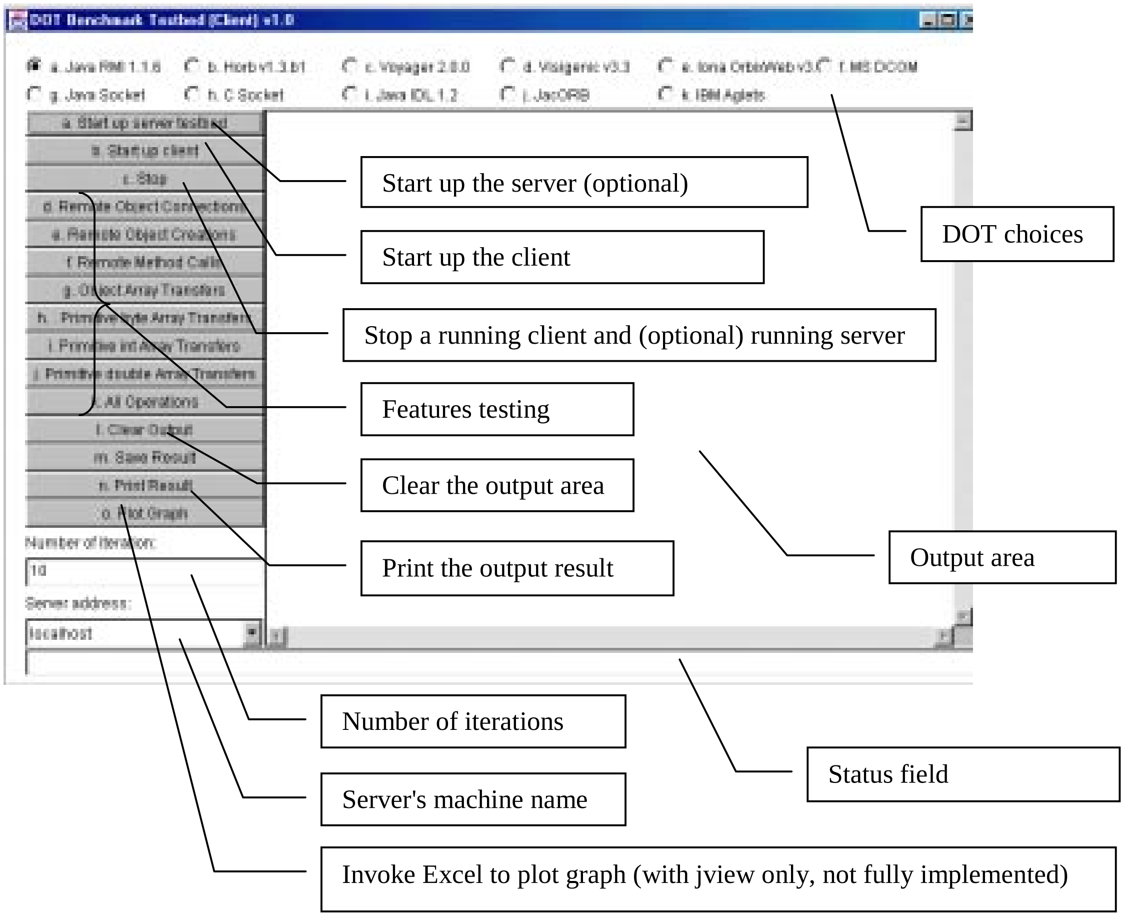
Distributed Garbage Collection	
DOT	**Implementation**
Java RMI 1.1.6	Reference counting
HORB 1.3b1	Reference counting
Voyager 2.0.0	Delta pinging (every 2 minutes)
OrbixWeb 3.0	No support, custom built
VisiBroker 3.3 for Java	No support, custom built
DCOM 95	Pinging (every 2 minutes)
Java IDL 1.2	Reference counting???

15. The GUI Test bed version for DOT Benchmark

In this section, I describe the GUI version that I wrote for our benchmark test. The GUI gives user a flexible way to test out the performance of the DOTs described in this project. The user can control each feature tests for each DOT separately and as a whole. The running client can start and stop a running server on a specified machine. Of course, this GUI program needs to be installed and configured appropriately on those machines. The GUI testbed was written using HORB. The reason I choose HORB to develop this GUI program because it is much easier than the other DOT and we only test on a small set of operations. HORB is good enough for doing this. The GUI uses the ETL's microbenchmark suite as a main tool to measure the performance of DOTs in this project. The GUI has 2 component, a client GUI and a server GUI which I described them in the section followed up.

15.1 The Client

<u>Figure 15-1</u>: The client GUI



First of all, make sure we have correctly installed all the DOT packages. One of the problems that had caused me a lot of pain is the system environment, the path and the classpath. It seems to me that VisiBroker and OrbixWeb do not co-accessible through the system classpath on the Windows NT machine, either one of the other must be turned off in the system classpath. However on the same machine which run Windows 98 OS, it doesn't have any problem with this. So, if you are running this application on NT machines, you need to take a good care of this classpath environment variable. This would sound a bit silly because we try to test all DOTs together but have to change the classpath manually.

Anyway, to run this application, make sure the dotclasses is installed on the two machines and the systems classpath can access to it. From the dos command prompt, enter this command line:

Prompt>java DOTBenchmark.gui.Rclient

This command will start up our client GUI program. Also make sure there is a horb daemon running on the server machine. Now we can ready for the test.

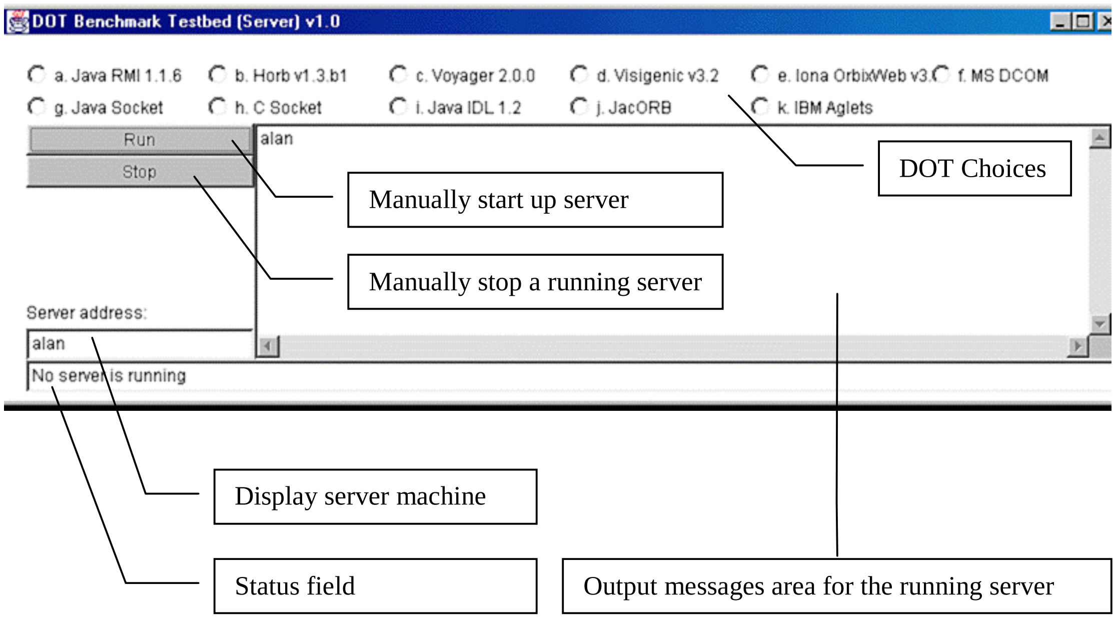
Click on the start up server test bed on the client GUI will automatically create the server GUI on the server machine. Now we can have either choice of starting up a particular server manually to serve for a particular client or let the client control the startup and shutdown the server program. After the client/server pairs have been up and running, all we have to do is to click on the button which specifies the test features for that DOT.

The result produced by the test is displayed on the client window GUI. We can now save it into a text file for processing later on. However, there is a button called "Plot graph" that will plot the graph from the generated results. This button will call up the Microsoft Excel to run and plot the graph. This feature is implemented by using Java calling COM in Microsoft Java technology. Therefore, it only runs with the Microsoft JVM, not Sun JVM. At the moment, I have just invoked Excel to run and have not actually implemented the plot graph. I intend to do it for the next phase of the project if I have a chance.

One of the big problem I have encountered is the result of the benchmark is less accurate than in the case I tested on the dos command line. This is because I couldn't isolate the disruption by the running of the GUI process itself. Sometimes the running client or server of the previous test couldn't be destroyed completely, then I have to manually kill it myself.

15.2 The Server

Figure 15-2: The Server GUI



alan

alan

No server is running

The server GUI is very easy to understand, it is just used for display the message at the status field saying that whether it is running or not. The output area is used to display any related message such as there is a exception occurred or prompt message to tell user to clean up any previous running client/server so that a new test can be started. The radio button is used to indicate the current running server. Other fields are obvious as described by the text box pointers.

By the way, we can also start up the server GUI from the dos command line by entering this command at the dos command prompt as follows:

Prompt>java DOTBenchmark.gui.RemoteController_Impl

16. Discussion and Further Research

We have discussed the three main distributed object technologies: Java RMI, CORBA DCOM with some other DOTs such as HORB, Voyager, Java IDL. We also outlined the steps that are needed to develop a basic distributed client and server application using these technologies. In order to realize the basic characteristics of each DOT and how the perform, we wrote the same client/server application for each DOT to perform the most basic operations such as remote object connection, remote object creation, remote method call, object transfer by value and large numerical array transfer. In the numerical array transfer, we also developed a socket version to see how fast each DOT in comparing with the plain socket when they are used to transferring large numerical array size. From the benchmark above, we can see that the cost of overhead added by the DOT involved in marshalling and unmarshalling process is very expensive.

In the project, we benchmarked each of these operations using the latency factor as the main criteria to justify how fast each DOT can perform. This means that we try to measure how fast the marshalling and unmarshalling process can be done by each DOT. There are lots of other features that can be tested for the next phase of the project. For example, transferring objects is one of the interesting feature can be tested. In the project, we have already tested the transferring of simple objects with only a small number of objects and we did not perform the object casting. How will the benchmark change if the object we want to transfer is a complex one, e.g. a complex object tree? And what about object that are casted further down from the top of the object hierarchy? If we try to increase the number of objects transfer, what is the limit that a DOT can handle for a method call. How efficient can a stub/proxy and skeleton of each DOT handle this kind of operation? Will they still keep its speed? Other possible tests can be the patterns we used to make the method call. Which way is more efficient if we make less message call with more arguments and if we make more message calls with less arguments, how can this be handled in each DOTs?

In distributed computing, we need to concern the partial failure factors and resource management factors as well. Partial failure problem is a non-trivial task to solve in distributed computing. The chance of communication failure is more likely to happen in a distributed application environment than in a local application environment. So we might interest to test on how reliability a DOT can provide. In resource management, we might interest in measuring how cooperative each DOT can be, how efficient each DOT can provide in load balancing. These and some more other features might be a very interesting test that can be carried out for the next phase of this benchmark project.

17. Future of Distributed Object Technology

17.1 The CORBA 3.0

The OMG will announce a pre-production release of the CORBA 3.0 specification before year-end 1998. A CORBA 3.0's final release is expected in the first half of 1999. This latest revision builds on the success of CORBA to date and incorporates the new technologies that organizations are using today. The result is an industry standard that supports and promotes the use of all enterprise technologies in building new powerful applications. CORBA 3.0 will achieve a new level of capabilities for both experienced distributed computing programmers and less technical business programmers/analysts. The new version of CORBA will simplify the use of CORBA ORBs for the development of distributed object applications and include features that will facilitate CORBA's adoption in the enterprise.

CORBA 3.0 capabilities fall into several categories, these include support for distributed components, quality of service features, new messaging support, and other technologies that will enable complete Internet integration and support for legacy environment.

Three core aspects added to the OMG's CORBA 3.0 specification;

- **Java/Internet Integration and Legacy Support Features:**
  - Completed suite of support includes IDL to Java mapping to integrate Java programs into CORBA applications
  - Java to IDL mapping to support automated IDL generation from existing Java programs
  - Binary portable Java stubs for executing any Java/CORBA application in any Java VM.
    Interim specification can be found at: http://www.omg.org./corba/c2indx.htm
  - DCE/CORBA Internetworking specification provide a roadmap for integrating legacy DCE applications into CORBA environments. It is available at ftp://ftp.omg.org/pub/docs/orbos/98-06-01.pdf)
  - Firewall specification that enables IIOP to securely traverse corporate firewalls
    Final portion proxy support specification can be found at: ftp://ftp.omg.org/pub/docs/orbos/98-07-03.pdf

- **Quality of Service Management**
  - Quality of Service encompasses a number of service-level selection capabilities and legacy integration
  - Support message-oriented middleware (MOM) integration. Specification is available at ftp://ftp.omg.org/pub/docs/orbos/98-05-06.pdf

- Minimum CORBA addresses for a CORBA-compliant system that can operate in an embedded environment. The specification identifies a minimum compliance for supporting CORBA ORBs. The draft specification can be found at: ftp://ftp.omg.org/pub/docs/orbos/98-08-04.pdf
- Real-time CORBA 1.0 extends the CORBA specification for a new type of ORB called the Real-time ORB. A Real-time ORBs might be comprised of fixed priority scheduling, control over ORB resources for end-to-end predictability, and flexible communications. Its draft specification is available at http://www.omg.org/library/schedule/Realtime_CORBA_1.0_RFP.htm
- The CORBA Component Model is the most significant addition to CORBA. This component architecture is a superset of Enterprise Java Bean. It will specify a framework for the development of "plug-and-play" CORBA objects. The specification specifies a Component Model and Container that can be mapped onto various programming languages. The CORBA Component Model draft specification can be found at: http://www.omg.org/library/schedule/CORBA_Component_Model_RFP_htm
- The CORB Scripting Language specification will make composition of CORBA components easier using scripts. Its draft specification is available at: http://www.omg.org/library/schedule/CORBA_Scripting_Language_RFP.htm
- Objects-by-Value provides a straightforward and efficient method of passing information in an object-oriented form across a network. Support for Objects-by-Value passing will help programmers integrate CORBA more seamlessly into modern programming languages. Extending IIOP support to value objects passing is an enabler for CORBA 3.0's new Java-to-IDL mapping. The final specification for support passing objects-by-value is available at ftp://ftp.omg.org/pub/docs/orbos/98-01-18.pdf
- Multiple Interfaces will allow a single object to present multiple views of itself through an interface selection mechanism. This allows programs to automatically control access to an object's functions based on interface definitions, operations, or other criteria.

17.2 Microsoft COM+

Microsoft next generation for Distributed Computing is the COM+. COM+ includes the MTS (Microsoft Transaction Server) development and deployment tool as well as a series of services, including events, publish-and-subscribe event model, dynamic load balancing and queued components via Microsoft message queue server.

At the TechEd Microsoft developers' conference in New Orleans on Jun 1998, Microsoft showed the world that it is offering attributed-based programming, or the capability to easily build a server-side component and tag it as transactional object or a queued object.

MTS provides context objects for these server objects so that they can participate in transactions. By the way MTS objects must be implemented in the form of DLLs that will be hosted by MTS surrogate processes. A server object instance notifies MTS when it has finished processing the current task and no longer needs the volatile state associated with the instance. MTS can then reuse that instance and its associated resources to serve other clients without incurring object-activation and resource initialized overhead.

18. Conclusion

Distributed object technology is the major breakthrough in modern distributed computing. It is dramatically changing the ways in which software systems evolve over time. It is the cornerstone for building the next generation of business applications, the modern client/server model that uses a middle ware as the main communication mechanism. This middle ware as we have seen is called the Object Request Broker (ORB) and have standardized by OMG with a Common Object Request Broker Architecture model. Given so may DOTs available on the market nowadays, which one will we consider to be a candidate for building our client/server application for the next ten years? Which is the winning technology that will shape the way we develop and deploy our major software applications?

In order to leverage this task, we have come up with a small set of benchmark operations which tests on the most common operations of all DOTs. What we were trying to do in this project is to discover the relative speeds (or latency) for each DOTs because it is considered to be an important factor in distributed computing to the application developers point of view. We let all the DOTs run on the same platforms with the same environment configuration. Based on the result of this benchmark, we can say that no DOT won all our benchmark tests. However, the only DOT that shows very promising performance is HORB. Its performance in remote method calls is very good and consistent. Providing a good performance in method call is a critical tool in building and maintaining large distributed object system. Because Object Oriented programming encourages programmers to use more data encapsulation, the result will increase the number of method calls. Faster method calls allow programmer to make more frequently call with shorter message call than making fewer call with longer message call. This leads to design applications with smaller parts that can be easily maintained. No doubt that there is a small amount of overhead that DOTs added into the process of marshalling and unmarshalling the method call when comparing with the plain socket versions. Although HORB is not as fast as socket C or socket Java in transferring primitive array, it is the fastest among other DOTs.

Based on the results I found in doing this DOT benchmark project, I would like to make the following points in this conclusion section as follows:

- Remote Object Connection time is reduced for those DOTs that have equipped with connection multiplexing feature and have a good locating service implementation. Connection multiplexing saves DOTs the time to create each new TCP/IP connection in obtaining each new remote object reference. All DOTs except HORB can support connection multiplexing. However, the overhead to create each new connection in HORB is small and efficient comparing to the other DOTs. Java IDL although have implemented connection multiplexing, its tnameserv locating service performs very bad, which results it to be the slowest of all DOTs in the remote object connection. A flexible connection scheme for DOT will be a good choice.

- Remote Object Creation saves system resources because objects will only be created when they are needed. The Remote Object Creation time is quite more expensive in Java RMI than it is in Voyager. DOTs that have equipped with its own method to create remote object shows better performance than those DOTs don't have, which we use a custom process to simulate the remote object creation.

- Remote Method Call time varies considerably among the DOTs. Remote Method Call is a bit more expensive in DOTs than it is in Socket C and Socket Java. It also shows that, adding one more parameter to the method call does not increase the time required in the marshalling and unmarshalling process of the method call. And regardless of the number of parameters, there is an overhead in setting up a method call. Once the method call has been set up, adding a few parameters will have little impact on the time it takes to make the call.

- Object Transfer by value (without casting) time basically increases with the number of object transferred. This indicates more processing are involved to serialize the object data into byte stream. Although HORB uses Java serialization system, it handles object transferred more efficient than Java RMI in this respect.

- Large numerical array transfer requires intensive CPU resource, all DOTs show higher cost than the plain socket version. Socket Java shows very high performance in transferring large array of byte but it degrades dramatically when it is used to transfer large array of int or double, this is because of the default of small buffer size of the BufferOutputStream and the BufferInputStream. Those DOTs support connection multiplexing are more expensive than HORB in this operation. This is because HORB has a dedicated connection for each data path where as the others have to pay the cost of sharing bandwidth of the data path, they also have to pay the cost of copying data and buffer management to ensure the integrity of the shared data resource.

In term of development process for distributed applications using this DOTs, HORB and Voyager are the most straightforward and the easiest to use and configure. They also come with an easy to understand user's guide which provides step-by-step examples to illustrator the program development. Next is Java RMI, which is rated to be an medium-easy. Then VisiBroker, OrbixWeb and Java IDL which have about the same requirements for developing a CORBA program. They also have a comprehensive programmer's guide and user's guide; however it's not very easy to understand. The most difficult to develop and less user friendly is DCOM although it has been improved since its first release.

This project has just tested only on the latency factor. In the large scale deployment, we also need to consider other important factors such as partial failure, concurrency control, error control logging, central monitoring that are handled by DOTs so that a comprehensive features can be used to select a DOT for deployment.

19. References

[1] Orfali, R., Harkey, D., Client/Server Programming with Java and CORBA, 2nd Ed, John Wiley & Sons, Inc., 1998

[2] Harold, E., R., Java Network Programming, O'Reilly & Associates, Inc., 1st Ed, February, 1997

[3] Jain, R., The Art of Computer Systems Performance Analysis, John Wiley & Sons, Inc., 1991

[4] Flanagan, D., Java In A Nutshell, A Desktop Quick Reference, 2nd Ed, O'Reilly & Associates, Inc., 1997

[5] Horstmann, C., S., Cornell, G., Core Java 1.1, Volume 1-Fundamental, The SunSoft Press, Java Series, 1997

[6] Horstmann, C., S., Cornell, G., Core Java 1.1, Volume 2-Advanced Features, The SunSoft Press, Java Series, 1998

[7] Chan, P., Lee, R., The Java Class Libraries, 2nd Ed, Volume 2, Addison Wesley Longman, 1998

[8] Sun Microsystems, Remote Method Invocation Specification, 1998

[9] Sun Microsystems, Object Serialization Specification, 1998

[10] Sun Microsystems, Java Remote Method Invocation-Distributed Computing for Java, White Paper, 1998

[11] Gosling, J., McGilton, H., Java Language Environment, White Paper

[12] Gosling, J., Joy, B., Steele, G., The Java Language Specification, Addison-Wesley, 1996

[13] Birrell, A., Evers, D., Nelson, G., Owicki, S., Wobber, E., Distributed Garbage Collection for Network Objects, Digital, SRC, 1993

[14] Sun Microsystems, Java IDL, 1998

[15] Morgan, B., Java 1.2 extends Java's Distributed Object Capabilities, JavaWorld, April 1998

[16] Morgan, B., CORBA meets Java, JavaWorld, October 1997

[17] Object Management Group, The Common Object Request Broker: Architecture and Specification, 2.2, 1998

[18] Vinoski, S., CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, 1997

[19] Waldo, J., Wyant, G., Wollrath, A., Kendall, S., A Note on Distributed Computing, Sun Microsystems Laboratory, 1994

[20] Vinoski, S., Distributed Object Computing With CORBA, C++ Report, 1993

[21] Krieger, D., Adler, R., The Emergence of Distributed Component Platforms, IEEE Computer, 1998

[22] Inprise Corporation, VisiBroker 3.3 for Java Programmer Guide, 1998

[23] Inprise Corporation, VisiBroker 3.3 for Java Programmer Reference, 1998

[24] Iona Technology, OrbixWeb 3.0 Programmer's Guide, 1997

[25] Iona Technology, OrbixWeb 3.0 Programmer's Reference, 1997

[26] Microsoft Corp., DCOM Technical Overview, White Paper, 1996

[27] Microsoft Corp., DCOM Architecture, White Paper, 1997

[28] Wang, Y., Chung, P., Customization of Distributed Systems Using COM, IEEE Concurrency, 1998

[29] Chung, P., Huang, Y., Yajnik, S., Liang, D., Shih, J., Wang, C., Wang, Y., DCOM and CORBA Side by Side, Step by Step, and Layer by Layer, C++ Report, 1997

[30] Albertson, T., Best Practices in Distributed Object Application Development: RMI, CORBA and DCOM, JavaWorld, 1998

[31] HIRANO, S., Yasu, Y., Igarashi, H., Performance Evaluation of Popular Distributed Object Technologies for Java, HORB Open and ETL, 1997

[32] HORB, HORB's User Guide, 1996

[33] HIRANO, S., Distributed Execution of Java Programs, ETL, 1997

[34] Hagimoto, J., Construct Java Applications Through Distributed Object Technology, JavaWorld, 1997

[35] Sarmenta, L., HORB Serialization Patch 1.1a, 1998

[36] ObjectSpace, VOYAGER Core Technology 2.0 User Guide, 1998

[37] Microsoft Tech-Ed, Introduction to COM+, 1998

[38] Fingar, P., Stikeleather, J., Distributed Object for Business, 1996

[39] Wallnau, K., Weiderman, N., Northrop, L., Distributed Object Technology With CORBA and Java: Key Concepts and Implications, Technical Report, 1997

[40] Weiderman, N., Northrop, L., Smith, D., Tilley, S., Wallnau, K., Implications of Distributed Object Technology for Reenginerring, 1997

20. Appendix I-Client/Sever Code Base

20.1 The Client GUI

The Client GUI Testbed

```
// The Rclient.java
//
// Main program for Benchmark interface
// Written for DOT project 490.450DT 1998
// @version v1.0
// Last modified: September 7th, 1998
//
//
// by Michael Ta
//
package DOTBenchmark.gui;


import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb.Features.Config;
import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

import DOTBenchmark.voy20.*;
import com.objectspace.voyager.*;
import DOTBenchmark.visibroker.*;
import DOTBenchmark.orbix.*;


import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Properties;

import java.net.*;
import java.rmi.*;
import horb.orb.*;

//import org.omg.CosNaming.*;  // JavaIDL will use the naming service.
//import excel8.*; // for using to
//import com.ms.com.*; // invoke Excel


public class Rclient extends Frame implements ActionListener {
    private Panel checkBoxGroupPanel = new Panel();
    private Panel buttonPanel = new Panel();
    private TextArea outputArea = new TextArea();
    private TextField statusField = new TextField();
    private Choice serverName = new Choice();
    private TextField iterationField;
    private int iterationNumber = 10;
    private java.awt.Button creationButton, connectionButton, callButton, objectArrayButton, byteArrayButton;
    private java.awt.Button intArrayButton, doubleArrayButton, allOperationButton, clearButton, plotGraphButton;
    private java.awt.Button startServerButton, startClientButton, stopButton, printButton, saveButton;
    private String serveraddress;
    private CheckboxGroup chboxg = new CheckboxGroup();
    Properties p = new Properties();
    DOTBenchmark.rmi.Server rmiServer = null;
    DOTBenchmark.horb.Server_Proxy horbServer = null;
    DOTBenchmark.voy20.IServer voyagerServer = null;
    vbroker.Server visibrokerServer = null;
    DOTBenchmark.orbix.Server orbixWebServer = null;
    //DOTBenchmark.javaidl.Server javaIDLServer = null;
    RemoteController rcserver = null; // remote controller server
    Process child;
    private boolean openxls = false;
    private boolean running = false;
private StringBuffer temp = new StringBuffer();
int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
          4096*8, 4096*16, 4096*32, 4096*64 };
long startTime, time;
double elapse;
HorbURL url;
org.omg.CORBA.ORB orb;

/*_Global globXL = null;
_Application appXL = null;
Workbooks books = null;
_Workbook book = null;*/

public static void main(String args[]) {
     Rclient project = new Rclient();

}

public Rclient() {
     setTitle("DOT Benchmark Testbed (Client) v1.0");
     setSize(800,600);
     /** Register to handle window events */
     addWindowListener(new WindowAdapter() {
     public void windowClosing( WindowEvent evt ) {
          if (openxls) {
               //closeExcel();
          }
          dispose();
          System.exit( 0 );
          }
     } );
     setLayout(new BorderLayout());
     Panel westPanel = new Panel();
     westPanel.setLayout (new BorderLayout());
     buttonPanel.setLayout(new GridLayout(15,1));
     westPanel.add("North", buttonPanel);
     startServerButton = makeButton(buttonPanel,"a. Start up server testbed");
     startClientButton = makeButton(buttonPanel,"b. Start up client");
     stopButton = makeButton(buttonPanel, "c. Stop");
     creationButton = makeButton(buttonPanel, "d. Remote Object Connections");
     connectionButton = makeButton(buttonPanel, "e. Remote Object Creations");
     callButton = makeButton(buttonPanel, "f. Remote Method Calls");
     objectArrayButton = makeButton(buttonPanel, "g. Object Array Transfers");
     byteArrayButton = makeButton(buttonPanel, "h. . Primitive byte Array Transfers");
     intArrayButton = makeButton(buttonPanel, "i. Primitive int Array Transfers");
     doubleArrayButton = makeButton(buttonPanel, "j. Primitive double Array Transfers");
     allOperationButton = makeButton(buttonPanel, "k. All Operations");
     clearButton = makeButton(buttonPanel, "l. Clear Output");
     saveButton = makeButton(buttonPanel, "m. Save Result");
     printButton = makeButton(buttonPanel, "n. Print Result");
     plotGraphButton = makeButton(buttonPanel, "o. Plot Graph");

     Panel p = new Panel();
     p.setLayout( new GridLayout (4,1));
     p.add(new java.awt.Label("Number of iteration:"));
     p.add(iterationField = new TextField("" + iterationNumber) );
     p.add(new java.awt.Label("Server address:"));
     serverName.add("localhost");
     serverName.add("citr-pc1");
     serverName.add("citr-pc2");
     p.add(serverName);
     westPanel.add("South",p);
     add("West", westPanel);
     Panel northPanel = new Panel();
     northPanel.setLayout( new BorderLayout());
     checkBoxGroupPanel.setLayout( new GridLayout(2,6));
     northPanel.add("Center", checkBoxGroupPanel);

     checkBoxGroupPanel.add( new Checkbox ("a. Java RMI 1.1.6", chboxg, true));
     checkBoxGroupPanel.add( new Checkbox ("b. Horb v1.3.b1", chboxg, false));
     checkBoxGroupPanel.add( new Checkbox ("c. Voyager 2.0.0", chboxg, false));
     checkBoxGroupPanel.add( new Checkbox ("d. Visigenic v3.3", chboxg, false));
```

```
        checkBoxGroupPanel.add( new Checkbox ("e. Iona OrbixWeb v3.0", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("f. MS DCOM", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("g. Java Socket", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("h. C Socket", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("i. Java IDL 1.2", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("j. JacORB", chboxg, false));
        checkBoxGroupPanel.add( new Checkbox ("k. IBM Aglets", chboxg, false));

        add("North", checkBoxGroupPanel);
        outputArea.setEditable(false);
        add("Center", outputArea);
        statusField.setEditable(false);
        add("South", statusField);
        pack();
        show();
    }

    /** Add a Button with a name to a panel and register a listener object with this button **/
    public java.awt.Button makeButton(Panel p, String name) {
        java.awt.Button b = new java.awt.Button(name);
        b.addActionListener(this);
        p.add(b);
        return b;

    }

    /** Add a Checkbox with a label to a panel and register a listener object with this Checkbox **/
    public Checkbox addCheckbox(Panel p, String name) {
        Checkbox c = new Checkbox(name);
        //c.addItemListener(this);
        p.add(c);
        return c;
    }



    /** Handle a click on the button panel **/
    public void actionPerformed(ActionEvent evt) {
        char c = evt.getActionCommand().charAt(0);

        if (c == 'a') { // start up server
            serveraddress = serverName.getSelectedItem().trim();
            HorbURL url = new HorbURL("horb://" + serveraddress);
            rcserver = new RemoteController_Proxy(url);
            println("Server testbed has been created at: " + rcserver.getServerName());
            try {
                Thread.sleep(5000);
            } catch (Exception exxxx) {}
        }

        if (c == 'b') { //start up client
            serveraddress = serverName.getSelectedItem().trim();
            println("Server address: " + serveraddress);
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    println("Starting Java RMI Server...");
                    rcserver.startupJavaRMIServer();
                    println("Starting Java RMI Client...");
                    startupJavaRMIClient(serveraddress);
                    running = true;
                    break;
                case 'b':
                    println("Starting HORB Server...");
                    rcserver.startupHorbServer();
                    println("Starting HORB Client...");
                    startupHorbClient(serveraddress);
                    running = true;
                    break;
                case 'c':
                    println("Starting Voyager Server...");
                    rcserver.startupVoyagerServer();
                println("Starting Voyager Client...");
                startupVoyagerClient(serveraddress);
                running = true;
                break;
            case 'd':
                println("Starting VisiBroker Server...");
                rcserver.startupVisiBrokerServer();
                println("Starting VisiBroker Client...");
                startupVisibrokerClient(serveraddress);
                running = true;
                break;
            case 'e':
                println("Starting Iona OrbixWeb Server...");
                rcserver.startupIonaOrbixWebServer();
                println("Starting Iona OrbixWeb Client...");
                startupIonaOrbixWebClient(serveraddress);
                running = true;
                break;
            case 'f':
                println("Starting Microsoft Dcom Client...");
                startupMSDComClient();
                running = true;
                break;
            case 'g':
                println("Starting Java Socket Client...");
                println("Iteration is set to 1000");
                startupJavaSocketClient();
                running = true;
                break;
            case 'h':
                println("Starting C Socket Client...");
                println("Iteration is set to 1000");
                startupCSocketClient();
                running = true;
                break;
            case 'i':
                println("Starting JavaIDL Client...");
                startupJavaIDLClient(serveraddress);
                running = true;
                break;
            case 'j':
                println("Starting JacORB Client...");
                startupJacORBClient();
                running = true;
                break;
            case 'k':
                println("Starting IBM Aglets Client...");
                startupIBMAgletsClient();
                running = true;
                break;
            default:
                println("No Client is running...");
        }
    }

    else if (c == 'c') { // stop a running client
        if (running) {
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    println("Shutting down Java RMI Client...");
                    rmiServer = null;
                    println("Shutting down Java RMI Server...");
                    rcserver.shutDownServer();
                    System.gc();
                    running = false;
                    break;
                case 'b':
                    println("Shutting down HORB Client...");
                    horbServer = null;
                    println("Shutting down HORB Server...");
                    rcserver.shutDownServer();
                    System.gc();
                    running = false;
                    break;
                case 'c':
                    println("Shutting down Voyager Client..");
                    voyagerServer = null;
                    println("Shutting down Voyager Server..");
                    rcserver.shutDownServer();
                    System.gc();
                    running = false;
                    break;
                case 'd':
                    println("Shutting down VisiBroker Client...");
                    visibrokerServer = null;
                    println("Shutting down Voyager Server..");
                    rcserver.shutDownServer();
                    System.gc();
                    running = false;
                    break;
                case 'e':
                    println("Shutting down Iona OrbixWeb Client...");
                    orbixWebServer = null;
                    println("Shutting down Iona OrbixWeb Server...");
                    rcserver.shutDownServer();
                    System.gc();
                    running = false;
                    break;
                case 'f':
                    println("Shutting down Microsoft Dcom Client...");
                    shutDownClient(child);
                    running = false;
                    break;
                case 'g':
                    println("Shutting down Java Socket Client...");
                    shutDownClient(child);
                    break;
                case 'h':
                    println("Shutting down C Socket Client...");
                    shutDownClient(child);
                    running = false;
                    break;
                case 'i':
                    println("Shutting down JavaIDL Client...");
                    shutDownClient(child);
                    running = false;
                    break;
                case 'j':
                    println("Shutting down JacORB Client...");
                    shutDownClient(child);
                    running = false;
                    break;
                case 'k':
                    println("Shutting down IBM Aglets Client...");
                    shutDownClient(child);
                    running = false;
                    break;
                default:
                    println("No client is running...");
            }
            setStatus("No client is running");
        }
    }

    else if (c == 'd') { // Remote Object Connection operation
        if (running) {
            try {
                iterationNumber = Integer.parseInt(iterationField.getText().trim());
            } catch (Exception e) {
                System.err.println(e);
            }
            serveraddress = serverName.getSelectedItem().trim();
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
```

```
                switch (option) {
                    case 'a':
                        JavaRMIRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'b':
                        HorbRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'c':
                        VoyagerRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'd':
                        VisibrokerRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'e':
                        OrbixWebRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    /*case 'f':
                        println("Starting Microsoft Dcom Client...");
                        startupMSDComClient();
                        running = true;
                        break;
                    case 'g':
                        println("Starting Java Socket Client...");
                        println("Iteration is set to 1000");
                        startupJavaSocketClient();
                        running = true;
                        break;
                    case 'h':
                        println("Starting C Socket Client...");
                        println("Iteration is set to 1000");
                        startupCSocketClient();
                        running = true;
                        break;*/
                    case 'i':
                        JavaIDLRemoteObjectConnection(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'j':
                        println("Starting JacORB Client...");
                        startupJacORBClient();
                        running = true;
                        break;
                    case 'k':
                        println("Starting IBM Aglets Client...");
                        startupIBMAgletsClient();
                        running = true;
                        break;
                    default:
                        println("No Client is running...");
                }
            }
            else
                println("Make sure client and server are up and running!");
        }

        else if (c == 'e') { // Remote object creation
            if (running) {
                try {
                    iterationNumber = Integer.parseInt(iterationField.getText().trim());
                } catch (Exception e) {
                    System.err.println(e);
                }
                serveraddress = serverName.getSelectedItem().trim();
                char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
                switch (option) {
                    case 'a':
                        JavaRMIRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'b':
                        HorbRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'c':
                        VoyagerRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'd':
                        VisibrokerRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'e':
                        OrbixWebRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    /*case 'f':
                        println("Starting Microsoft Dcom Client...");
                        startupMSDComClient();
                        running = true;
                        break;
                    case 'g':
                        println("Starting Java Socket Client...");
                        println("Iteration is set to 1000");
                        startupJavaSocketClient();
                        running = true;
                        break;
                    case 'h':
                        println("Starting C Socket Client...");
                        println("Iteration is set to 1000");
                        startupCSocketClient();
                        running = true;
                        break;*/
                    case 'i':
                        JavaIDLRemoteObjectCreation(iterationNumber, serveraddress);
                        println("Finished");
                        break;
                    case 'j':
                        println("Starting JacORB Client...");
                        startupJacORBClient();
                        running = true;
                        break;
                    case 'k':
                        println("Starting IBM Aglets Client...");
                        startupIBMAgletsClient();
                        running = true;
                        break;
                    default:
                        println("No Client is running...");
                }
            }
            else
                println("Make sure client is up and running!");
        }
        else if (c == 'f') { // Remote method call
            if (running) {
                try {
                    iterationNumber = Integer.parseInt(iterationField.getText().trim());
                } catch (Exception e) {
                    System.err.println(e);
                }
                serveraddress = serverName.getSelectedItem().trim();
                char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
                switch (option) {
                    case 'a':
                        JavaRMIRemoteMethodCall(iterationNumber);
                        println("Finished");
                        break;
                    case 'b':
                        HorbRemoteMethodCall(iterationNumber);
                    println("Finished");
                    break;
                case 'c':
                    VoyagerRemoteMethodCall(iterationNumber);
                    println("Finished");
                    break;
                case 'd':
                    VisibrokerRemoteMethodCall(iterationNumber);
                    println("Finished");
                    break;
                case 'e':
                    OrbixWebRemoteMethodCall(iterationNumber);
                    println("Finished");
                    break;
                /*case 'f':
                    println("Starting Microsoft Dcom Client...");
                    startupMSDComClient();
                    running = true;
                    break;
                case 'g':
                    println("Starting Java Socket Client...");
                    println("Iteration is set to 1000");
                    startupJavaSocketClient();
                    running = true;
                    break;
                case 'h':
                    println("Starting C Socket Client...");
                    println("Iteration is set to 1000");
                    startupCSocketClient();
                    running = true;
                    break;*/
                case 'i':
                    JavaIDLRemoteMethodCall(iterationNumber);
                    println("Finished");
                    break;
                case 'j':
                    println("Starting JacORB Client...");
                    startupJacORBClient();
                    running = true;
                    break;
                case 'k':
                    println("Starting IBM Aglets Client...");
                    startupIBMAgletsClient();
                    running = true;
                    break;
                default:
                    println("No Client is running...");
            }
        }
        else
            println("Make sure client is up and running!");
    }

    else if (c == 'g') { // Object array transfer
        if (running) {
            try {
                iterationNumber = Integer.parseInt(iterationField.getText().trim());
            } catch (Exception e) {
                System.err.println(e);
            }

            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    JavaRMIObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'b':
                    HorbObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'c':
                    VoyagerObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'd':
                    VisibrokerObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'e':
                    OrbixWebObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                /*case 'f':
                    println("Starting Microsoft Dcom Client...");
                    startupMSDComClient();
                    running = true;
                    break;
                case 'g':
                    println("Starting Java Socket Client...");
                    println("Iteration is set to 1000");
                    startupJavaSocketClient();
                    running = true;
                    break;
                case 'h':
                    println("Starting C Socket Client...");
                    println("Iteration is set to 1000");
                    startupCSocketClient();
                    running = true;
                    break;*/
                case 'i':
                    JavaIDLObjectDataTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'j':
                    println("Starting JacORB Client...");
                    startupJacORBClient();
                    running = true;
                    break;
                case 'k':
                    println("Starting IBM Aglets Client...");
                    startupIBMAgletsClient();
                    running = true;
                    break;
                default:
                    println("No Client is running...");
            }
        }
        else
            println("Make sure client is up and running!");
    }

    else if (c == 'h') { // byte array transfer
        if (running) {
            try {
                iterationNumber = Integer.parseInt(iterationField.getText().trim());
            } catch (Exception e) {
                System.err.println(e);
            }
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    JavaRMIbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'b':
                    HorbbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'c':
                    VoyagerbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'd':
                    VisibrokerbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'e':
                    OrbixWebbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                /*case 'f':
                    println("Starting Microsoft Dcom Client...");
                    startupMSDComClient();
                    running = true;
                    break;
                case 'g':
                    println("Starting Java Socket Client...");
                    println("Iteration is set to 1000");
                    startupJavaSocketClient();
                    running = true;
                    break;
                case 'h':
                    println("Starting C Socket Client...");
                    println("Iteration is set to 1000");
                    startupCSocketClient();
                    running = true;
                    break;*/
                case 'i':
                    JavaIDLbyteArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'j':
                    println("Starting JacORB Client...");
                    startupJacORBClient();
                    running = true;
                    break;
                case 'k':
                    println("Starting IBM Aglets Client...");
                    startupIBMAgletsClient();
                    running = true;
                    break;
                default:
                    println("No Client is running...");
            }
        }
        else
            println("Make sure client is up and running!");
    }

    else if (c == 'i') { // int array transfer
        if (running) {
            try {
                iterationNumber = Integer.parseInt(iterationField.getText().trim());
            } catch (Exception e) {
                System.err.println(e);
            }
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    JavaRMIintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'b':
                    HorbintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'c':
                    VoyagerintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'd':
                    VisibrokerintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'e':
                    OrbixWebintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                /*case 'f':
                    println("Starting Microsoft Dcom Client...");
                    startupMSDComClient();
                    running = true;
                    break;
                case 'g':
                    println("Starting Java Socket Client...");
                    println("Iteration is set to 1000");
                    startupJavaSocketClient();
                    running = true;
                    break;
                case 'h':
                    println("Starting C Socket Client...");
                    println("Iteration is set to 1000");
                    startupCSocketClient();
                    running = true;
                    break;*/
                case 'i':
                    JavaIDLintArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'j':
                    println("Starting JacORB Client...");
                    startupJacORBClient();
                    running = true;
                    break;
                case 'k':
                    println("Starting IBM Aglets Client...");
                    startupIBMAgletsClient();
                    running = true;
                    break;
                default:
                    println("No Client is running...");
            }
        }
        else
            println("Make sure client is up and running!");
    }
    else if (c == 'j') { // double array transfer
        if (running) {
            try {
                iterationNumber = Integer.parseInt(iterationField.getText().trim());
            } catch (Exception e) {
                System.err.println(e);
            }
            char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
            switch (option) {
                case 'a':
                    JavaRMIdoubleArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'b':
                    HorbdoubleArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'c':
                    VoyagerdoubleArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'd':
                    VisibrokerdoubleArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                case 'e':
                    OrbixWebdoubleArrayTransfer(iterationNumber);
                    println("Finished");
                    break;
                /*case 'f':
                    println("Starting Microsoft Dcom Client...");
```

```
                                startupMSDComClient();
                                running = true;
                                break;
                            case 'g':
                                println("Starting Java Socket Client...");
                                println("Iteration is set to 1000");
                                startupJavaSocketClient();
                                running = true;
                                break;
                            case 'h':
                                println("Starting C Socket Client...");
                                println("Iteration is set to 1000");
                                startupCSocketClient();
                                running = true;
                                break;*/
                            case 'i':
                                JavaIDLdoubleArrayTransfer(iterationNumber);
                                println("Finished");
                                break;
                            case 'j':
                                println("Starting JacORB Client...");
                                startupJacORBClient();
                                running = true;
                                break;
                            case 'k':
                                println("Starting IBM Aglets Client...");
                                startupIBMAgletsClient();
                                running = true;
                                break;
                            default:
                                println("No Client is running...");
                        }
                    }
                    else
                        println("Make sure client is up and running!");
                }
                else if (c == 'k') { // All operations
                    if (running) {
                        try {
                            iterationNumber = Integer.parseInt(iterationField.getText().trim());
                        } catch (Exception e) {
                            System.err.println(e);
                        }
                        serveraddress = serverName.getSelectedItem().trim();
                        char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
                        switch (option) {
                            case 'a':
                                JavaRMIRemoteObjectConnection(iterationNumber, serveraddress);
                                JavaRMIRemoteObjectCreation(iterationNumber, serveraddress);
                                JavaRMIRemoteMethodCall(iterationNumber);
                                JavaRMIObjectDataTransfer(iterationNumber);
                                JavaRMIbyteArrayTransfer(iterationNumber);
                                JavaRMIintArrayTransfer(iterationNumber);
                                JavaRMIdoubleArrayTransfer(iterationNumber);
                                println("Finished");
                                break;
                            case 'b':
                                HorbRemoteObjectConnection(iterationNumber, serveraddress);
                                HorbRemoteObjectCreation(iterationNumber, serveraddress);
                                HorbRemoteMethodCall(iterationNumber);
                                HorbObjectDataTransfer(iterationNumber);
                                HorbbyteArrayTransfer(iterationNumber);
                                HorbintArrayTransfer(iterationNumber);
                                HorbdoubleArrayTransfer(iterationNumber);
                                println("Finished");
                                break;
                            case 'c':
                                VoyagerRemoteObjectConnection(iterationNumber, serveraddress);
                                VoyagerRemoteObjectCreation(iterationNumber, serveraddress);
                                VoyagerRemoteMethodCall(iterationNumber);
                                VoyagerObjectDataTransfer(iterationNumber);
                                VoyagerbyteArrayTransfer(iterationNumber);
                        VoyagerintArrayTransfer(iterationNumber);
                        VoyagerdoubleArrayTransfer(iterationNumber);
                        println("Finished");
                        break;
                    case 'd':
                        VisibrokerRemoteObjectConnection(iterationNumber, serveraddress);
                        VisibrokerRemoteObjectCreation(iterationNumber, serveraddress);
                        VisibrokerRemoteMethodCall(iterationNumber);
                        VisibrokerObjectDataTransfer(iterationNumber);
                        VisibrokerbyteArrayTransfer(iterationNumber);
                        VisibrokerintArrayTransfer(iterationNumber);
                        VisibrokerdoubleArrayTransfer(iterationNumber);
                        println("Finished");
                        break;
                    case 'e':
                        OrbixWebRemoteObjectConnection(iterationNumber, serveraddress);
                        OrbixWebRemoteObjectCreation(iterationNumber, serveraddress);
                        OrbixWebRemoteMethodCall(iterationNumber);
                        //OrbixWebObjectDataTransfer(iterationNumber);
                        OrbixWebbyteArrayTransfer(iterationNumber);
                        OrbixWebintArrayTransfer(iterationNumber);
                        OrbixWebdoubleArrayTransfer(iterationNumber);
                        println("Finished");
                        break;
                    case 'f':
                        println("Starting Microsoft Dcom Client...");
                        startupMSDComClient();
                        running = true;
                        break;
                    case 'g':
                        println("Starting Java Socket Client...");
                        println("Iteration is set to 1000");
                        startupJavaSocketClient();
                        running = true;
                        break;
                    case 'h':
                        println("Starting C Socket Client...");
                        println("Iteration is set to 1000");
                        startupCSocketClient();
                        running = true;
                        break;
                    case 'i':
                        JavaIDLRemoteObjectConnection(iterationNumber, serveraddress);
                        JavaIDLRemoteObjectCreation(iterationNumber, serveraddress);
                        JavaIDLRemoteMethodCall(iterationNumber);
                        JavaIDLObjectDataTransfer(iterationNumber);
                        JavaIDLbyteArrayTransfer(iterationNumber);
                        JavaIDLintArrayTransfer(iterationNumber);
                        JavaIDLdoubleArrayTransfer(iterationNumber);
                        println("Finished");
                        break;
                    /*case 'j':
                        println("Starting JacORB Client...");
                        startupJacORBClient();
                        running = true;
                        break;
                    case 'k':
                        println("Starting IBM Aglets Client...");
                        startupIBMAgletsClient();
                        running = true;
                        break;*/
                    default:
                        println("No Client is running...");
                }
            }
            else
                println("Make sure client is up and running!");
        }

    else if (c == 'l') { // clear output
            outputArea.setText("");
    }
```

```
else if (c == 'm') { // Save output
        setStatus("Save output");
        FileDialog fileDialog = new FileDialog(this, "Save File", FileDialog.SAVE);
        fileDialog.setFile("*.txt");

        fileDialog.show();
        String outFileName = fileDialog.getDirectory() + fileDialog.getFile();

        if (outFileName != null) {
            try {
                PrintWriter out = new PrintWriter (
                    new FileWriter(outFileName));
                out.println(outputArea.getText());
                out.close();
            } catch (IOException ex) {
                System.out.println("IO Exception");
            }
            setStatus("Output is saved");
        }

        else {
            setStatus("");
            return;
        }
    }

    else if (c == 'n') { // Print result
        PrintJob pjob = getToolkit().getPrintJob(this, "Cool...", p);
        if (pjob != null) {
            Graphics pg = pjob.getGraphics();
            if (pg != null) {
                String s = outputArea.getText();
                printLongString (pjob, pg, s);
                pg.dispose();
            }
            pjob.end();
        }
    }

    else if (c == 'o') { // plot graph
        println("Opening Microsoft Excel...");
        //openExcel();
    }
}

public void shutDownClient(Process p) {
    if (p != null) {
        p.destroy();
    }
}

public void startupJavaRMIClient(String host) {
    try {
        String[] list = Naming.list("//"+host+":2005/");
        for (int i = 0; i < list.length; i++) {
            println(list[i]);
        }
        rmiServer = (DOTBenchmark.rmi.Server)Naming.lookup("//"+host+":2005/Server");   // ignore this
        println("done");
        setStatus("Java RMI client is running...");
    }
    catch (Exception e) {
        System.err.println(e);
    }
}

void JavaRMIRemoteObjectConnection(int n, String host) {
    try {
        System.gc();
        startTime = System.currentTimeMillis();
        for (int i = 0; i < n; i++)
```

```
                rmiServer = (DOTBenchmark.rmi.Server)Naming.lookup("//"+host+":2005/Server");
                  // ?? how do I release the connection???
              }
              time = System.currentTimeMillis() - startTime;
              elapse = (double)(time)/(double)(n);
              println("Java RMI remote object connection benchmark: "+elapse+" msec");
          } catch (Exception e) {
              System.err.println(e);
          }
      }

      void JavaRMIRemoteObjectCreation(int n, String host) {
          try {
              rmiServer = null;
              DOTBenchmark.rmi.Server server2 = null;
              System.gc();
              startTime = System.currentTimeMillis();
              for (int i = 0; i < n; i++) {
              rmiServer = (DOTBenchmark.rmi.Server)Naming.lookup("//"+host+":2005/Server");
                  server2 = rmiServer.createInstance();
                  // ?? how do I release the connection???
              }
              time = System.currentTimeMillis() - startTime;
              elapse = (double)(time)/(double)(n);
              println("Java RMI remote object creation benchmark: "+elapse+" msec");
              rmiServer = null;
              System.gc();
              rmiServer = (DOTBenchmark.rmi.Server)Naming.lookup("//"+host+":2005/Server");
          }
          catch (Exception e) {
              System.err.println(e);
          }
      }


      void JavaRMIRemoteMethodCall(int iter) {
          int i, x;

          try {
              // Ignore this just WARM UP
              System.gc();
              for (i = iter; i > 0; --i)
                      x = rmiServer.methodA1(100);

              // rmc with one argument
              System.gc();
              startTime = System.currentTimeMillis();
              for (i = iter; i > 0; --i)
                      x = rmiServer.methodA1(100);
              time = System.currentTimeMillis() - startTime;
              elapse = (double)(time)/(double)(iter);
              println("int server.methodA1(int a1): "+elapse+" msec");

              // rmc with two arguments
              System.gc();
              startTime = System.currentTimeMillis();
              for (i = iter; i > 0; --i)
                      x = rmiServer.methodA2(100, 100);
              time = System.currentTimeMillis() - startTime;
              elapse = (double)(time)/(double)(iter);
              println("int server.methodA2(int, int): "+elapse+" msec");

              // rmc with three arguments
              System.gc();
              startTime = System.currentTimeMillis();
              for (i = iter; i > 0; --i)
                      x = rmiServer.methodA3(100, 100, 100);
              time = System.currentTimeMillis() - startTime;
              elapse = (double)(time)/(double)(iter);
              println("int server.methodA3(int, int, int): "+elapse+" msec");

              // rmc with four arguments
```

```
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                    x = rmiServer.methodA4(100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA4(int, int, int, int): "+elapse+" msec");

            // rmc with five arguments
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                    x = rmiServer.methodA5(100, 100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

            // rmc with six arguments
            System.gc();
            tartTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                    x = rmiServer.methodA6(100, 100, 100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void JavaRMIObjectDataTransfer(int iter) {
        // Object send in array without cast benchmark

        int[] sizes = {1, 10, 20, 30, 40, 50};
        DOTBenchmark.rmi.Data[] data;

        try {
            for (int s = 0; s < 6; s++) {
                data = new DOTBenchmark.rmi.Data[sizes[s]];
                for (int j = 0; j < sizes[s]; j++)
                    data[j] = new DOTBenchmark.rmi.Data();
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                        rmiServer.methodDA(data);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodDA(DOTBenchmark.rmi.Data["+sizes[s]+"]): "+elapse+" msec");
            }
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void JavaRMIbyteArrayTransfer(int iter) {
        byte[] bufb;

        try {
            // to send byte array
            for (int j = 0; j < array_sizes.length; j++) {
                bufb = new byte[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    rmiServer.methodB(bufb);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j])/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void JavaRMIintArrayTransfer(int iter) {
        int iteration;
        int[] bufi;

        try {
            // to send int array
            for (int j = 0; j < 6; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    rmiServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }

            for (int j = 6; j < array_sizes.length; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                iteration = iter/10; //change the iteration
                startTime = System.currentTimeMillis();
                for (int i = iteration; i > 0; --i) {
                    rmiServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec")
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void JavaRMIdoubleArrayTransfer(int iter) {
        int i, j, iteration;
        double[] bufd;
        try {
            // to send double array
            for ( j = 0; j < 4; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    rmiServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (i = iteration; i > 0; --i) {
                rmiServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
         }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    public void startupHorbClient(String host) {
        try {
            HORBClient.setPort(8898); // make a different port
            url = new HorbURL(host,"HServer");
            DOTBenchmark.horb.Server_Proxy junk = new DOTBenchmark.horb.Server_Proxy(url);  // ignore this
            horbServer = new DOTBenchmark.horb.Server_Proxy(url);
            setStatus("Horb client is running...");
        }
        catch(Exception e) {
            System.err.println(e);
        }
    }

    void HorbRemoteObjectConnection(int iter, String host) {
        int i;

        try {
            DOTBenchmark.horb.Server_Proxy horbServers[] = new DOTBenchmark.horb.Server_Proxy[iter];
            url = new HorbURL(host,"Server");
            //try {
                //Thread.sleep(5000);
            //} catch (Exception exxxx) {}
            DOTBenchmark.horb.Server_Proxy junk = new DOTBenchmark.horb.Server_Proxy(url);  // ignore this
            System.gc();
            startTime = System.currentTimeMillis();
            for ( i = 0; i < iter; i++) {
                horbServers[i] = new DOTBenchmark.horb.Server_Proxy(url);
            }
            time = System.currentTimeMillis() - startTime;

            for( i = 0; i < iter; i++) {
                horbServers[i]._release();
            }
            elapse = (double)(time)/(double)(iter);
            println("Horb remote object connection benchmark: "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void HorbRemoteObjectCreation(int iter, String host) {
        int i;
        try {
            url = new HorbURL(host,null);
            DOTBenchmark.horb.Server_Proxy horbServers[] = new DOTBenchmark.horb.Server_Proxy[iter];
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = 0; i < iter; i++) {
                horbServers[i] = new DOTBenchmark.horb.Server_Proxy(url);
                horbServers[i].Server();
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Horb remote object connection benchmark : "+elapse+" msec");
            for( i = 0; i < iter; i++) {
                horbServers[i]._release();
            }
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void HorbRemoteMethodCall(int iter) {
        int i,x;
        HORB.useSerialization();// Using new feature in HORB
        try {
            // load class files explicitely
            x = horbServer.methodA1(0);
            // ignore this just WARM UP
            System.gc();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA1(100);
            // remote method invocation benchmark

            // one argument
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA1(100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA1(int a1): "+elapse+" msec");

            // two arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA2(100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA2(int, int): "+elapse+" msec");

            // three arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA3(100, 100, 100);
            time = System.currentTimeMillis() – startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA3(int, int, int): "+elapse+" msec");

            // four arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA4(100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA4(int, int, int, int): "+elapse+" msec");

            // five arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA5(100, 100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

            // six arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = horbServer.methodA6(100, 100, 100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void HorbObjectDataTransfer(int iter) {
        // Object send in array without cast benchmark
        int i,j;
        int[] sizes = {1, 10, 20, 30, 40, 50};
        HORB.useSerialization();// Using new feature in HORB
        try {
            for (int s = 0; s < 6; s++) {
                DOTBenchmark.horb.Data data[] = new DOTBenchmark.horb.Data[sizes[s]];
                for (j = 0; j < sizes[s]; j++)
                    data[j] = new DOTBenchmark.horb.Data();
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    horbServer.methodDA(data);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodDA(Data["+sizes[s]+"]): "+elapse+" msec");
            }
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void HorbbyteArrayTransfer(int iter) {
        byte[] bufb;
        HORB.useSerialization();// Using new feature in HORB
        try {
            // to send byte array
            for (int j = 0; j < array_sizes.length; j++) {
                bufb = new byte[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    horbServer.methodB(bufb);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j])/elapse) + " KB/sec")
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void HorbintArrayTransfer(int iter) {
        int iteration;
        int[] bufi;

        HORB.useSerialization();// Using new feature in HORB
        try {
            // to send int array
            for (int j = 0; j < 6; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    horbServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
            for (int j = 6; j < array_sizes.length; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (int i = iteration; i > 0; --i)
                    horbServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void HorbdoubleArrayTransfer(int iter) {
        int i, j, iteration;
        double[] bufd;

        HORB.useSerialization();// Using new feature in HORB
        try {
            // to send double array
            for ( j = 0; j < 4; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    horbServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (i = iteration; i > 0; --i)
                    horbServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    public void startupVoyagerClient(String host) {
            try {
                Voyager.startup(); // startup as client
                println("looking up...");
                voyagerServer = (DOTBenchmark.voy20.IServer)Namespace.lookup("//"+host+":8000/VServer");
                println("done");
                setStatus("Voyager client is running...");
            }
            catch (Exception e) {
                System.err.println(e);
            }
    }

    void VoyagerRemoteObjectConnection(int iter, String host) {
        int i;

        try {
            DOTBenchmark.voy20.IServer servers[] = new DOTBenchmark.voy20.IServer[iter];
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = 0; i < iter; i++)
                servers[i] = (DOTBenchmark.voy20.IServer)Namespace.lookup("//"+host+":8000/VServer");
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Voyager remote object connection benchmark: "+elapse+" msec");
            // release connection...
            for( i = 0; i < iter; i++) {
                servers[i] = null;
            }
            System.gc();
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VoyagerRemoteObjectCreation(int iter, String host) {
        int i;

        try {
            DOTBenchmark.voy20.IServer servers[] = new DOTBenchmark.voy20.IServer[iter];
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = 0; i < iter; i++)
                // create a remote Server object
                servers[i] = (DOTBenchmark.voy20.IServer) Factory.create("DOTBenchmark.voy20.Server",
"//"+host+":8000");

            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Voyager remote object creation benchmark: "+elapse+" msec");
            // release connection...
            for( i = 0; i < iter; i++) {
                servers[i] = null;
            }
            System.gc();
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VoyagerRemoteMethodCall(int iter) {
        int i,x;

        try {
           // load class files explicitely
           x = voyagerServer.methodA1(0);
           // ignore this just WARM UP
           System.gc();
           for (i = iter; i > 0; --i)
                x = voyagerServer.methodA1(100);

           // remote method invocation benchmark

            // one argument
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA1(100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA1(int a1): "+elapse+" msec");

            // two arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA2(100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA2(int, int): "+elapse+" msec");

            // three arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA3(100, 100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA3(int, int, int): "+elapse+" msec");

            // four arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA4(100, 100, 100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA4(int, int, int, int): "+elapse+" msec");

            // five arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA5(100, 100, 100, 100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

            // six arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = voyagerServer.methodA6(100, 100, 100, 100, 100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VoyagerObjectDataTransfer(int iter) {
        // Object send in array without cast benchmark
        int i,j;
        int[] sizes = {1, 10, 20, 30, 40, 50};
        try {
            for (int s = 0; s < 6; s++) {
                DOTBenchmark.voy20.Data data[] = new DOTBenchmark.voy20.Data[sizes[s]];
                for (j = 0; j < sizes[s]; j++)
                data[j] = new DOTBenchmark.voy20.Data();
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    voyagerServer.methodDA(data);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodDA(Data["+sizes[s]+"]): "+elapse+" msec");
            }
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VoyagerbyteArrayTransfer(int iter) {
        byte[] bufb;

        try {
            // to send byte array
            for (int j = 0; j < array_sizes.length; j++) {
                bufb = new byte[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
```

```
                    voyagerServer.methodB(bufb);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j])/elapse) + " KB/sec");
            }

        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void VoyagerintArrayTransfer(int iter) {
        int iteration;
        int[] bufi;

        try {
            // to send int array
            for (int j = 0; j < 6; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    voyagerServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
            for (int j = 6; j < array_sizes.length; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (int i = iteration; i > 0; --i)
                    voyagerServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
         }
         catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void VoyagerdoubleArrayTransfer(int iter) {
        int i, j, iteration;
        double[] bufd;

        try {
            // to send double array
            for ( j = 0; j < 4; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    voyagerServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (i = iteration; i > 0; --i)
                    voyagerServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    public void startupVisibrokerClient(String host) {
        try {
          // Initialize the ORB
            orb = org.omg.CORBA.ORB.init();
            // Bind to the server
            visibrokerServer = vbroker.ServerHelper.bind(orb, "Server");
            setStatus("Visibroker client is running...");
        }
        catch(Exception e) {
            System.err.println(e);
        }
    }

    void VisibrokerRemoteObjectConnection(int iter, String host) {
        int i;
        try {
            // remote object connection benchmark
            System.gc();
            vbroker.Server junk = vbroker.ServerHelper.bind(orb, "Server");  // ignore the first one
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
                //server._release();
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Visibroker remote object connection benchmark: "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VisibrokerRemoteObjectCreation(int iter, String host) {
        int I;
        try {
            // remote object creation benchmark
            System.gc();
            vbroker.Server server2 = null;
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
                server2 = server.createInstance();
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Visibroker remote object creation benchmark : "+elapse+" msec");

        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VisibrokerRemoteMethodCall(int iter) {
        int i,x;

        try {
           // load class files explicitely
           x = visibrokerServer.methodA1(0);
           // ignore this just WARM UP
```

```
            System.gc();
            for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA1(100);
             // remote method invocation benchmark

             // one argument
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA1(100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA1(int a1): "+elapse+" msec");

             // two arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA2(100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA2(int, int): "+elapse+" msec");

             // three arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA3(100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA3(int, int, int): "+elapse+" msec");

             // four arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA4(100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA4(int, int, int, int): "+elapse+" msec");

             // five arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA5(100, 100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

             // six arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = visibrokerServer.methodA6(100, 100, 100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
             System.err.println(e);
        }
   }

   void VisibrokerObjectDataTransfer(int iter) {
        // Object send in array without cast benchmark
        int[] sizes = {1, 10, 20, 30, 40, 50};

        try {
             // Object send benchmark
             DOTBenchmark.visibroker.ServerOT server = DOTBenchmark.visibroker.ServerOTHelper.bind(orb,
"ServerOT");
```

```
            for (int s = 0; s < 6; s++) {
                DOTBenchmark.visibroker.DataOT[] data = new DOTBenchmark.visibroker.DataOT[sizes[s]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    server.methodDA(data);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodDA(Data["+sizes[s]+"]): "+ elapse+" msec");
            }
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void VisibrokerbyteArrayTransfer(int iter) {
        byte[] bufb;
        try {
            // to send byte array
            for (int j = 0; j < array_sizes.length; j++) {
                bufb = new byte[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    visibrokerServer.methodB(bufb);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j])/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
                System.err.println(e);
        }

    }

    void VisibrokerintArrayTransfer(int iter) {
        int iteration;
        int[] bufi;
        try {
            // to send int array
            for (int j = 0; j < 6; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (int i = iter; i > 0; --i)
                    visibrokerServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
            for (int j = 6; j < array_sizes.length; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (int i = iteration; i > 0; --i)
                    visibrokerServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
         }
         catch ( Exception e ) {
            System.err.println(e);
        }
    }
```

```
void VisibrokerdoubleArrayTransfer(int iter) {
    int i, j, iteration;
    double[] bufd;
    try {
        // to send double array
        for ( j = 0; j < 4; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                visibrokerServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }
        for ( j = 4; j < array_sizes.length; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            iteration = iter/10;//change the iteration
            startTime = System.currentTimeMillis();
            for (i = iteration; i > 0; --i)
                visibrokerServer.methodD(bufd);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iteration);
            println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }
     }
     catch ( Exception e ) {
        System.err.println(e);
    }
}

public void  startupIonaOrbixWebClient(String host) {
    try {
        ORB.init();
        try {
            Thread.sleep(3*1000);
        } catch (Exception e) {}

        orbixWebServer = DOTBenchmark.orbix.ServerHelper.bind(":Server", host);
        setStatus("Iona OrbixWeb client is running...");
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

void OrbixWebRemoteObjectConnection(int iter, String host) {
    int i;
    DOTBenchmark.orbix.Server server = null;
    server = DOTBenchmark.orbix.ServerHelper.bind(":Server", host);

    try {
        System.gc();
        startTime = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server = DOTBenchmark.orbix.ServerHelper.bind(":Server", host);
            // ?? how do I release the connection??
        }
        time = System.currentTimeMillis() - startTime;
        elapse = (double)(time)/(double)(iter);
        println("Iona OrbixWeb remote object connection benchmark: "+elapse+" msec");
    }
    catch (Exception e) {
        System.err.println(e);
    }
}

void OrbixWebRemoteObjectCreation(int iter, String host) {
    int i;
```

```
        DOTBenchmark.orbix.Server server = null;
        server = DOTBenchmark.orbix.ServerHelper.bind(":Server", host);
        try {
            // remote object creation benchmark
            System.gc();
            DOTBenchmark.orbix.Server server2 = null;
            startTime = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server = DOTBenchmark.orbix.ServerHelper.bind(":Server", host);
                server2 = (DOTBenchmark.orbix.Server)server.createInstance();
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("Iona OrbixWeb remote object creation benchmark : "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void OrbixWebRemoteMethodCall(int iter) {
        int i,x;
        try {
            // load class files explicitely
            x = orbixWebServer.methodA1(0);
            // ignore this just WARM UP
            System.gc();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA1(100);

            // remote method invocation benchmark
            // one argument
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA1(100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA1(int a1): "+elapse+" msec");

            // two arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA2(100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA2(int, int): "+elapse+" msec");

            // three arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA3(100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA3(int, int, int): "+elapse+" msec");

            // four arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA4(100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA4(int, int, int, int): "+elapse+" msec");

            // five arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                x = orbixWebServer.methodA5(100, 100, 100, 100, 100);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

            // six arguments
            System.gc();
            startTime= System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                x = orbixWebServer.methodA6(100, 100, 100, 100, 100, 100);
            }
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
            System.err.println(e);
        }
    }

    void OrbixWebObjectDataTransfer(int iter) {
        println("OrbixWeb v3.0 for Java does not support transfer object by value");
    }

    void OrbixWebbyteArrayTransfer(int iter) {
        byte[] bufb;
        try {
             // to send byte array
            for (int j = 0; j < array_sizes.length; j++) {
                bufb = new byte[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
            for (int i = iter; i > 0; --i)
                orbixWebServer.methodB(bufb);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j])/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void OrbixWebintArrayTransfer(int iter) {
        int iteration;
        int[] bufi;
        try {
            // to send int array
            for (int j = 0; j < 6; j++) {
                bufi = new int[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
            for (int i = iter; i > 0; --i)
                orbixWebServer.methodI(bufi);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
            for (int j = 6; j < array_sizes.length; j++) {
                bufi = new int[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (int i = iteration; i > 0; --i)
                    orbixWebServer.methodI(bufi);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }

    void OrbixWebdoubleArrayTransfer(int iter) {
        int i, j, iteration;
        double[] bufd;

        try {
            // to send double array
            for ( j = 0; j < 4; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                startTime = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    orbixWebServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10;//change the iteration
                startTime = System.currentTimeMillis();
                for (i = iteration; i > 0; --i)
                    orbixWebServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }
    }
    public void  startupMSDComClient() {
        String cmd = "start /m jview client1 1000";
        try {
            setStatus("MS Dcom client is running...");
            child = Runtime.getRuntime().exec(cmd);
            getProcessOutput(child);
        }
        catch(IOException  e) {
            System.err.println(e);
        }
    }

    public void  startupJavaSocketClient() {
        String cmd = "java DOTBenchmark.javasocket.SockClient 1000";
        try {
            setStatus("Java client socket is running...");
            child = Runtime.getRuntime().exec(cmd);
            getProcessOutput(child);
        }
        catch(IOException  e) {
                System.err.println(e);
        }
    }

    public void  startupCSocketClient() {
        String cmd = "C:\\Project\\DOTBenchmark\\csocket\\client 1000";
        try {
            setStatus("C client socket is running...");
            child = Runtime.getRuntime().exec(cmd);
            getProcessOutput(child);
        }
        catch(IOException  e) {
              System.err.println(e);
        }
   }

   public void  startupJavaIDLClient(String host) {
    /* try {
        // Create and initialize the ORB
             ORB orb = ORB.init(args, null);

             // Get the root naming context
             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
             NamingContext ncRef = NamingContextHelper.narrow(objRef);


             // remote object connection benchmark
             // Resolve the object reference in naming
             NameComponent nc = new NameComponent("Server", " ");
             NameComponent path[] = {nc};
             Server junk = DOTBenchmark.javaidl.ServerHelper.narrow(ncRef.resolve(path));
          setStatus("Java IDL client is running...");
        }
        catch(Exception e) {
             e.printStackTrace();
        }*/

   }

   void JavaIDLRemoteObjectConnection(int iter, String host) {
        /*int i;
        DOTBenchmark.javaidl.Server server = null;
        server = DOTBenchmark.javaidl.ServerHelper.narrow(ncRef.resolve(path));
        try {
             System.gc();
             startTime = System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  server = DOTBenchmark.javaidl.ServerHelper.narrow(ncRef.resolve(path));
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("Iona OrbixWeb remote object connection benchmark: "+elapse+" msec");
        }
        catch (Exception e) {
             System.err.println(e);
        }*/
   }

   void JavaIDLRemoteObjectCreation(int iter, String host) {
        /*int i;

        DOTBenchmark.javaidl.Server server = null;
        server = DOTBenchmark.javaidl.ServerHelper.narrow(ncRef.resolve(path));
        try {
             //
             // remote object creation benchmark
             System.gc();
             DOTBenchmark.javaidl.Server server2 = null;
             startTime = System.currentTimeMillis();
             for (i = iter; i > 0; --i) {
                  server = DOTBenchmark.javaidl.ServerHelper.narrow(ncRef.resolve(path));
                  server2 = (DOTBenchmark.javaidl.Server)server.createInstance();
             }
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("Iona OrbixWeb remote object creation benchmark : "+elapse+" msec");

        }
        catch (Exception e) {
             System.err.println(e);
        }*/
   }

   void JavaIDLRemoteMethodCall(int iter) {
        /*int i,x;

        try {
            // load class files explicitely
            x = javaIDLServer.methodA1(0);

            // ignore this just WARM UP
            System.gc();
            for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA1(100);
             // remote method invocation benchmark
             // one argument
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA1(100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA1(int a1): "+elapse+" msec");

             // two arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA2(100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA2(int, int): "+elapse+" msec");

             // three arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA3(100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA3(int, int, int): "+elapse+" msec");

             // four arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA4(100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
             println("int server.methodA4(int, int, int, int): "+elapse+" msec");

             // five arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA5(100, 100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
              println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

             // six arguments
             System.gc();
             startTime= System.currentTimeMillis();
             for (i = iter; i > 0; --i)
                  x = javaIDLServer.methodA6(100, 100, 100, 100, 100, 100);
             time = System.currentTimeMillis() - startTime;
             elapse = (double)(time)/(double)(iter);
                  println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        }
        catch (Exception e) {
             System.err.println(e);
        }*/

    }
```

```
void JavaIDLObjectDataTransfer(int iter) {
    println("Java IDL 1.2 does not support transfer object by value");
}

void JavaIDLbyteArrayTransfer(int iter) {
    /*byte[] bufb;

    try {
         // to send byte array
        for (int j = 0; j < array_sizes.length; j++) {
            bufb = new byte[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
            for (int i = iter; i > 0; --i)
                javaIDLServer.methodB(bufb);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j])/elapse) + " KB/sec");
        }

   }
    catch ( Exception e ) {
        System.err.println(e);
    }*/
}

void JavaIDLintArrayTransfer(int iter) {
    /*int iteration;
    int[] bufi;

    try {
        // to send int array
        for (int j = 0; j < 6; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
            for (int i = iter; i > 0; --i)
                javaIDLServer.methodI(bufi);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iter);
            println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
        }
        for (int j = 6; j < array_sizes.length; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            iteration = iter/10;//change the iteration
            startTime = System.currentTimeMillis();
            for (int i = iteration; i > 0; --i)
                javaIDLServer.methodI(bufi);
            time = System.currentTimeMillis() - startTime;
            elapse = (double)(time)/(double)(iteration);
            println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
        }
    }
     catch ( Exception e ) {
        System.err.println(e);
    }*/
}

void JavaIDLdoubleArrayTransfer(int iter) {
    /*int i, j, iteration;
    double[] bufd;

    try {
        // to send double array
         for ( j = 0; j < 4; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            startTime = System.currentTimeMillis();
```

```
                for (i = iter; i > 0; --i)
                    javaIDLServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iter);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }

            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10;
                startTime = System.currentTimeMillis();
                for (i = iteration; i > 0; --i)
                    javaIDLServer.methodD(bufd);
                time = System.currentTimeMillis() - startTime;
                elapse = (double)(time)/(double)(iteration);
                println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
        }
        catch ( Exception e ) {
            System.err.println(e);
        }*/
    }

    public void  startupJacORBClient() {
    }

    public void  startupIBMAgletsClient() {
    }

    public Insets getInsets() {
        return new Insets(40, 20, 10, 10);
    }

    public void disableButtonsAndCheckbox() {
        Component[] components = buttonPanel.getComponents();
        for ( int i = 2 ; i < components.length ; i++)
            components[i].setEnabled(false);
        checkBoxGroupPanel.setEnabled(false);
        serverName.setEnabled(false);
    }

    public void enableButtonsAndCheckbox() {
        Component[] components = buttonPanel.getComponents();
        for ( int i = 2 ; i < components.length ; i++)
            components[i].setEnabled(true);
        checkBoxGroupPanel.setEnabled(true);
        serverName.setEnabled(false);
    }

    protected void print (String s) {
        outputArea.append(s);
    }

    protected void print (char c) {
        outputArea.append("" + c);
    }

    protected void println (String s) {
        outputArea.append(s + "\n");
    }

    protected void setStatus (String s) {
        statusField.setText(""+ s);
    }


    protected void getProcessOutput(Process p) {
        StringBuffer temp = new StringBuffer();
        try {
            Reader in = new BufferedReader(
                new InputStreamReader(p.getInputStream()));
            //InputStream in = p.getInputStream();
            int ch;
            // echo output of the running process p
            while ((ch = in.read()) != -1) {
                //System.out.print((char)c);
                //print((char)ch);
                temp.append( (char)ch );
            }
            in.close();
            // Wait for subprocess to exit
            try {
                p.waitFor();
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
            // Display exit status of subprocess
            setStatus("child process exited with " + p.exitValue());
        } catch (IOException ex) {
            System.err.println(ex);
        }

        // print the result from the string buffer
        setStatus("Displaying result...");
        for (int i = 0; i < temp.length(); i++) {
            print(temp.charAt(i));
        }
        setStatus("Finished displaying");
    }

    // Print string to graphics via printjob
    // Does not deal with word wrap or tabs
    void printLongString (PrintJob pjob, Graphics pg, String s) {
        int pageNum = 1;
        int linesForThisPage = 0;
        int linesForThisJob = 0;
        // Note: String is immutable so won't change while printing.
        if (!(pg instanceof PrintGraphics)) {
            throw new IllegalArgumentException ("Graphics context not PrintGraphics");
        }

        StringReader sr = new StringReader (s);
        LineNumberReader lnr = new LineNumberReader (sr);
        String nextLine;
        int pageHeight = pjob.getPageDimension().height;
        //Font helv = new Font("Helvetica", Font.PLAIN, 12);
        java.awt.Font cour = new java.awt.Font("Courier", java.awt.Font.PLAIN, 10);
        //have to set the font to get any output
        //pg.setFont (helv);
        pg.setFont (cour);
        FontMetrics fm = pg.getFontMetrics(cour);
        int fontHeight = fm.getHeight();
        int fontDescent = fm.getDescent();
        int curHeight = 0;
        try {
            do {
                nextLine = lnr.readLine();
                if (nextLine != null) {
                    if ((curHeight + fontHeight) > pageHeight) {
                        // New Page
                        System.out.println ("" + linesForThisPage + " lines printed for page " + pageNum);
                        pageNum++;
                        linesForThisPage = 0;
                        pg.dispose();
                        pg = pjob.getGraphics();
                        if (pg != null) {
                            pg.setFont (cour);
                        }
                        curHeight = 0;
                    }
                    curHeight += fontHeight;
                    if (pg != null) {
                        pg.drawString (nextLine, 0, curHeight - fontDescent);
                        linesForThisPage++;
                        linesForThisJob++;
                    } else {
                        System.out.println ("pg null");
                    }
                }
            } while (nextLine != null);

        } catch (EOFException eof) {
            // Fine, ignore
        } catch (Throwable t) { // Anything else
            t.printStackTrace();
        }
        setStatus ("" + linesForThisPage + " lines printed for page " + pageNum);
        setStatus ("pages printed: " + pageNum);
       setStatus ("total lines printed: " + linesForThisJob);
    }

    // this method is used to invoke MS Excel application
    // it opens an existing workbook call DOTBenchmark.xls
    /*public void openExcel() {
        _Global globXL = null;
        _Application appXL = null;
        Workbooks books = null;
        _Workbook book = null;

        try{
            globXL = (_Global)new Global();
            appXL = (_Application)globXL.getApplication();
            appXL.setVisible(0,true);
            books = (Workbooks)appXL.getWorkbooks();

            Variant vTemp = new Variant();
            vTemp.putString("C:\\Project\\DOTBenchmark\\DOTBenchmark.xls");
            Variant vOptional = new Variant();
            vOptional.noParam();
            book =
            (_Workbook)books.Open("C:\\Project\\DOTBenchmark\\DOTBenchmark.xls",vOptional,vOptional,vOption
            al,vOptional,vOptional,vOptional,vOptional,vOptional,vOptional,vOptional,vOptional,vOptional,0);
            openxls = true;
        }
        catch(ComFailException e) {
            System.out.println(e.getMessage());
        }
    }

    public void closeExcel() {
        //Sheets wsXL = null;
        //Cells cXL = null;

        try {
            println("Closing down Microsoft Excel...");
            openxls = false;
            //wsXL = book.getSheets();
            Variant vOptional = new Variant();
            vOptional.noParam();
            book.Close(vOptional,vOptional,vOptional,0);
            book = null;
            books.Close(0);
            books = null;
            globXL=null;
            appXL.Quit();
            appXL=null;
        } catch (ComFailException e) {
            System.out.println(e.getMessage());
        }
    }*/

    public  String getServerName() {
            String address = "";
            try {
                address = InetAddress.getLocalHost().getHostName();
                println("" + address);
            } catch (java.net.UnknownHostException e) {
                System.err.println("Unknown host");
            }
            return address;
        }
}
```

20.2 The Remote Interface

The RemoteController Java Interface

```
//RemoteController.java
package DOTBenchmark.gui;

public interface RemoteController {
      public String  getServerName();
      public void startupJavaRMIServer();
      public void shutDownRMIServer();
      public void startupHorbServer();
      public void shutDownHORBServer();
      public void startupVoyagerServer();
      public void shutDownVoyagerServer();
      public void startupVisiBrokerServer();
      public void startupIonaOrbixWebServer();
      public void shutDownServer();
}
```

20.3 The Server GUI

The Server GUI Testbed

```
// The RemoteController_Impl.java
/*
* Written for DOT Project 490.450DT
* @version v1.0
* @author Michael Ta
//
// Last modified: September 7th, 1998
//
*/

package DOTBenchmark.gui;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Properties;

import java.rmi.*;
import java.rmi.server.*;

class RemoteController_Impl extends Frame implements ActionListener, RemoteController {
      private Panel checkBoxGroupPanel = new Panel();
      private Panel buttonPanel = new Panel();
      private TextArea outputArea = new TextArea();
      private TextField statusField = new TextField();
      private TextField serverName;
      private Button runButton, stopButton;
      private String serveraddress;
      private CheckboxGroup chboxg = new CheckboxGroup();
      private Checkbox rmichbox, horbchbox, voychbox, visibrokerchbox, orbixchbox,dcomchbox,
            sockjchbox, sockcchbox, javaidlchbox, jacorbchbox, agletchbox;
    private boolean running = false;

    Process child, rmireg, visiosagent, orbixdj, tnamingserv;

    public static void main(String args[]) {
       RemoteController_Impl project = new RemoteController_Impl();
       project.setStatus("" + "No server is running");
    }

    public RemoteController_Impl() {
         setTitle("DOT Benchmark Testbed (Server) v1.0");
         setSize(800,600);
         /** Register to handle window events */
         addWindowListener(new WindowAdapter() {
                 public void windowClosing( WindowEvent evt ) {
                      dispose();
                      System.exit( 0 );
                 }
         } );

         setLayout(new BorderLayout());
         Panel westPanel = new Panel();
         westPanel.setLayout (new BorderLayout());
         buttonPanel.setLayout(new GridLayout(2,1));
         westPanel.add("North", buttonPanel);

         runButton = addButton(buttonPanel, "Run");
         stopButton = addButton(buttonPanel, "Stop");

         Panel p = new Panel();
         p.setLayout( new GridLayout (2,1));
         p.add(new Label("Server address:"));
         p.add(serverName = new TextField("localhost",20));
         serverName.setEditable(false);
         westPanel.add("South",p);
         add("West", westPanel);

         Panel northPanel = new Panel();
         northPanel.setLayout( new BorderLayout());
         checkBoxGroupPanel.setLayout( new GridLayout(2,6));
         northPanel.add("Center", checkBoxGroupPanel);


         rmichbox = new Checkbox ("a. Java RMI 1.1.6", chboxg, false);
         checkBoxGroupPanel.add(rmichbox);
         horbchbox = new Checkbox ("b. Horb v1.3.b1", chboxg, false);
         checkBoxGroupPanel.add(horbchbox);
         voychbox = new Checkbox ("c. Voyager 2.0.0", chboxg, false);
         checkBoxGroupPanel.add( voychbox );
         visibrokerchbox =  new Checkbox ("d. Visigenic v3.2", chboxg, false);
         checkBoxGroupPanel.add(visibrokerchbox);
         orbixchbox =  new Checkbox ("e. Iona OrbixWeb v3.0", chboxg, false);
         checkBoxGroupPanel.add(orbixchbox);
         dcomchbox =  new Checkbox ("f. MS DCOM", chboxg, false);
         checkBoxGroupPanel.add(dcomchbox);
         sockjchbox = new Checkbox ("g. Java Socket", chboxg, false);
         checkBoxGroupPanel.add( sockjchbox );
         sockcchbox= new Checkbox ("h. C Socket", chboxg, false);
         checkBoxGroupPanel.add( sockcchbox );
         javaidlchbox =  new Checkbox ("i. Java IDL 1.2", chboxg, false);
         checkBoxGroupPanel.add(javaidlchbox);
         jacorbchbox =  new Checkbox ("j. JacORB", chboxg, false);
         checkBoxGroupPanel.add(jacorbchbox);
         agletchbox =  new Checkbox ("k. IBM Aglets", chboxg, false);
         checkBoxGroupPanel.add(agletchbox);
         add("North", checkBoxGroupPanel);

         outputArea.setEditable(false);
         add("Center", outputArea);
         statusField.setEditable(false);
         add("South", statusField);
```

```
        pack();
        show();
    }



    /** Add a Button with a name to a panel and register a listener object with this button **/
    public Button addButton(Panel p, String name) {
        Button b = new Button(name);
        b.addActionListener(this);
        p.add(b);
        return b;
    }

    /** Add a Checkbox with a label to a panel and register a listener object with this Checkbox **/
    public void addCheckbox(Panel p, String name, CheckboxGroup g, boolean b) {
        Checkbox c = new Checkbox(name, g, b);
        //c.addItemListener(this);
        p.add(c);
    }

    public void add(Component c, GridBagConstraints  gbc, int x, int y, int w, int h) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        add(c,gbc);
    }


    /** Handle a click on the button panel **/
    public void actionPerformed(ActionEvent evt) {
        String what = evt.getActionCommand();

        if (what.equals("Run")) {
            disableButtonsAndCheckbox();
            if (!running) {
                char option = chboxg.getSelectedCheckbox().getLabel().charAt(0);
                switch (option) {
                    case 'a':
                        println("Starting Java RMI Server...");
                        startupJavaRMIServer();
                        running = true;
                        break;
                    case 'b':
                        println("Starting HORB Server...");
                        startupHorbServer();
                        running = true;
                        break;
                    case 'c':
                        println("Starting Voyager Server...");
                        startupVoyagerServer();
                        running = true;
                        break;
                    case 'd':
                        println("Starting VisiBroker Server...");
                        startupVisiBrokerServer();
                        running = true;
                        break;
                    case 'e':
                        println("Starting Iona OrbixWeb Server...");
                        startupIonaOrbixWebServer();
                        running = true;
                        break;
                    case 'f':
                        println("Starting Microsoft Dcom Server...");
                        startupMSDComServer();
                        running = true;
                        break;
                    case 'g':
                        println("Starting Java Socket Server...");
                        startupJavaSocketServer();
                        running = true;
                        break;
                    case 'h':
                        println("Starting C Socket Server...");
                        startupCSocketServer();
                        running = true;
                        break;
                    case 'i':
                        println("Starting Java IDL Server...");
                        startupJavaIDLServer();
                        running = true;
                        break;
                    case 'j':
                        println("Starting JacORB Server...");
                        startupJacORBServer();
                        running = true;
                        break;
                    case 'k':
                        println("Starting IBM Aglets Server...");
                        startupIBMAgletsServer();
                        running = true;
                        break;
                    default:
                        println("No server is running");
                        setStatus("No server is running");
                }
            }
        }

        else if (what.equals("Stop")) {
            enableButtonsAndCheckbox();
            if (running)
                shutDownServer();
            else
                setStatus(""+ "No server is running");
        }
    }

    public void shutDownServer() { // for local and remote method call
        if (child != null) {
            child.destroy();
            println("No server is running");
            setStatus("No server is running");
            running = false;
        }
        else
            setStatus("No server is running");
    }

    public void startupJavaRMIServer() {
        String[] progarray = new String[2];
        progarray[0] = "java";
        progarray[1] = "DOTBenchmark.rmi.ServerImpl";

        chboxg.setSelectedCheckbox(rmichbox);
        try {
            //rmireg = Runtime.getRuntime().exec("rmiregistry 2005");
            child = Runtime.getRuntime().exec(progarray);
            getProcessOutput(child);
            setStatus("Java RMI server is running...");
        }
        catch( IOException e) {
            System.err.println(e);
        }
    }

    public void  startupHorbServer() {
        String cmd = "horb -port 8898 -ioci horb.orb.SerializingIOCI  -start DOTBenchmark.horb.Server HServer";
        chboxg.setSelectedCheckbox(horbchbox);
        try {
            child = Runtime.getRuntime().exec(cmd);
            setStatus("Horb server is running...");
            //getProcessOutput(child);
        }
        catch(IOException  e) {
            System.err.println(e);
        }
    }

    public void  startupVoyagerServer() {
        String cmd = "java DOTBenchmark.voy20.Server";
        chboxg.setSelectedCheckbox(voychbox);
        try {
            child = Runtime.getRuntime().exec(cmd);
            getProcessOutput(child);
            setStatus("Voyager server is running...");
        }
        catch(IOException  e) {
            System.err.println(e);
        }
    }

    public void startupVisiBrokerServer() {
        String[] progarray = new String[2];
        progarray[0] = "java";
        progarray[1] = "DOTBenchmark.visibroker.ServerImpl";

        chboxg.setSelectedCheckbox(visibrokerchbox);
        try {
            // remember to start up visibroker osagent first
            child = Runtime.getRuntime().exec(progarray);
            getProcessOutput(child);
            setStatus("Visibroker server is running...");
        }
        catch( IOException e) {
            System.err.println(e);
        }
    }

    public void  startupIonaOrbixWebServer() {
        String cmd = "java DOTBenchmark.orbix.ServerImpl";
        chboxg.setSelectedCheckbox(orbixchbox);
        try {
            // Remember to start ups orbixdj on the command line
            child = Runtime.getRuntime().exec(cmd);
                getProcessOutput(child);
                setStatus("Iona OrbixWeb server is running...");
        }
        catch( IOException e) {
            System.err.println(e);
        }
    }

    public void  startupMSDComServer() {
        chboxg.setSelectedCheckbox(dcomchbox);
    }

    public void  startupJavaSocketServer() {
        String[] progarray = new String[2];
        progarray[0] = "java";
        progarray[1] = "DOTBenchmark.javasocket.SockServer 1000";

        chboxg.setSelectedCheckbox(sockjchbox);
        try {
            child = Runtime.getRuntime().exec(progarray);
            setStatus("Java  server socket is running...");
            //getProcessOutput(child);
        }
        catch( IOException e) {
            System.err.println(e);
        }
    }
```

```
public void  startupCSocketServer() {
    String cmd = "C:\\Project\\DOTBenchmark\\csocket\\server 1000";
    chboxg.setSelectedCheckbox(sockcchbox);

    try {
        child = Runtime.getRuntime().exec(cmd);
        setStatus("C server socket is running...");
        //getProcessOutput(child);
    }
    catch( IOException e) {
        System.err.println(e);
    }
}

public void  startupJavaIDLServer() {
    String[] progarray = new String[2];
    progarray[0] = "java";
    progarray[1] = "DOTBenchmark.javaidl.Server";

    chboxg.setSelectedCheckbox(javaidlchbox);
    try {
        //tnamingserv = Runtime.getRuntime().exec("C:\\Program Files\\jdk1.2beta4\\bin\\tnameserv.exe");
        // remember to start up tnameserv
        child = Runtime.getRuntime().exec(progarray);
        getProcessOutput(child);
        setStatus("Java IDL server is running...");
    }
    catch( IOException e) {
        System.err.println(e);
    }
}

public void  startupJacORBServer() {
    chboxg.setSelectedCheckbox(jacorbchbox);
}

public void  startupIBMAgletsServer() {
    chboxg.setSelectedCheckbox(agletchbox);
}

protected void print (String s) {
    outputArea.append(s);
}

protected void print (char c) {
    outputArea.append("" + c);
}

protected void println (String s) {
    outputArea.append(s + "\n");
}

protected void setStatus (String s) {
    statusField.setText("" + s);
}

protected void getProcessOutput(Process p) {
    try {
        Reader in = new BufferedReader(
            new InputStreamReader(p.getInputStream()));
        //InputStream in = p.getInputStream();
        int c, newline = 0;
        // echo output of the running process p
        while ((c = in.read()) != -1 && newline < 1) {
            char ch = (char)c;
            print(ch);
            if (ch == '\n')
            ++newline;
        }
        in.close();
        // Wait for subprocess to exit
        /*try {
```

```
                    p.waitFor();
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }*/
                // Display exit status of subprocess
                //System.out.println("child exited with " + p.exitValue());
                //println("child exited with " + p.exitValue());
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }

        public Insets getInsets() {
            return new Insets(40, 20, 10, 10);
        }

        public void disableButtonsAndCheckbox() {
            Component[] components = buttonPanel.getComponents();
            for ( int i = 0 ; i < components.length - 1 ; i++)
                components[i].setEnabled(false);
            checkBoxGroupPanel.setEnabled(false);

        }

        public void enableButtonsAndCheckbox() {
            Component[] components = buttonPanel.getComponents();
            for ( int i = 0 ; i < components.length - 1 ; i++)
                components[i].setEnabled(true);
            checkBoxGroupPanel.setEnabled(true);
        }

        public String getServerName() {
            String address = "";
            try {
                address = InetAddress.getLocalHost().getHostName();
                println("" + address);
                serverName.setText("" + address);
                setStatus("No server is running");
            } catch (java.net.UnknownHostException e) {
                System.out.println("Unknown host");
            }
            return address;
        }
}
```

21. Appendix II-ETL Micro-Benchmark Suite

21.1 Java RMI

Java RMI

```
// Server.java
package DOTBenchmark.rmi;
import java.rmi.*;
public interface Server extends java.rmi.Remote {
    Server createInstance() throws RemoteException;
    int methodA1(int a1) throws RemoteException;
    int methodA2(int a1, int a2) throws RemoteException;
    int methodA3(int a1, int a2, int a3) throws RemoteException;
    int methodA4(int a1, int a2, int a3, int a4) throws RemoteException;
    int methodA5(int a1, int a2, int a3, int a4, int a5) throws RemoteException;
    int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) throws RemoteException;
    void methodDA(Data[] dat) throws RemoteException;
    void methodB(byte[] a) throws RemoteException;
    void methodI(int[] ia) throws RemoteException;
    void methodD(double[] da)throws RemoteException;
}
```

//ServerImpl.java

```
package DOTBenchmark.rmi;
import java.rmi.*;
import java.rmi.server.*;

public class ServerImpl extends UnicastRemoteObject implements Server {
    static String serverName;
    static {
        try {
            serverName = java.net.InetAddress.getLocalHost().getHostName();
        } catch (Exception e) {
            serverName = "localhost";
        }
        System.out.println("Server name: "+serverName);
    }

    public ServerImpl() throws Exception {
        super();
    }

    public ServerImpl(String name) throws RemoteException {
        super();
        try {
            Naming.rebind("//"+name+":2005/Server",this);
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public Server createInstance() throws RemoteException {
        ServerImpl obj = null;
        try {
            obj = new ServerImpl();
            Naming.rebind("//"+serverName+":2005/Server", obj);
            return obj;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return obj;
    }

    public int methodA1(int a1) throws RemoteException {
        return(a1);
    }

    public int methodA2(int a1, int a2) throws RemoteException {
        return(a1);
    }

    public int methodA3(int a1, int a2, int a3) throws RemoteException {
        return(a1);
    }

    public int methodA4(int a1, int a2, int a3, int a4) throws RemoteException {
        return(a1);
    }

    public int methodA5(int a1, int a2, int a3, int a4, int a5) throws RemoteException {
        return(a1);
    }

    public int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) throws RemoteException {
        return(a1);
    }

    public void methodDA(Data[] dat) throws RemoteException {}
    public void methodB(byte[] a) throws RemoteException{}
    public void methodI(int[] a) throws RemoteException{}
    public void methodD(double[] a) throws RemoteException{}

    public static void main(String argv[]) throws Exception {
```

```
    System.setSecurityManager(new RMISecurityManager());
    ServerImpl obj = new ServerImpl();    // to create one i.e run rmiregistry
    Naming.rebind("//"+serverName+":2005/Server", obj);
        System.out.println("bind done");
        System.out.println("Server is listening at " + serverName + ":2005/Server");
    }
}
```

// Client.java

```
package DOTBenchmark.rmi;
import java.rmi.*;

class Client {
    public static void main(String argv[]) throws Exception {
    int         iter = 100;
    int         i, j, x;
    long          start, time;
    double        elapse;
    byte[]        bufb;
    int[]         bufi;
    double[]      bufd;
    Data[]        data;
    String        host = "localhost";


    if (argv.length == 2) {
            iter = Integer.parseInt(argv[0]);
            host = argv[1];
    } else if (argv.length == 1) {
            iter = Integer.parseInt(argv[0]);
    } else {
            System.err.println("java Client [iteration [host]]");
            System.exit(1);
    }
     //
     // remote object connection benchmark
     //
       Server server = null;
       String[] list = Naming.list("//"+host+":2005/");
       for (i = 0; i < list.length; i++) {
            System.out.println(list[i]);
       }

        System.out.println("looking up...");
        server = (Server)Naming.lookup("//"+host+":2005/Server");   // ignore this
        System.out.println("done");
        System.out.println("done");

       System.gc();
       start = System.currentTimeMillis();
       for (i = 0; i < iter; i++) {
            server = (Server)Naming.lookup("//"+host+":2005/Server");
            // ?? how do I release the connection???
       }
       time = System.currentTimeMillis() - start;
       elapse = (double)(time)/(double)(iter);
       System.out.println("Remote object connect: Naming.lookup(): "+elapse+" msec");

        /* try {
        Thread.sleep(30*1000);
        } catch (Exception e) {} */

        //
        // remote object creation benchmark
        //
        server = null;
        Server server2 = null;
        System.gc();
        start = System.currentTimeMillis();
```

```
for (i = 0; i < iter; i++) {
      server = (Server)Naming.lookup("//"+host+":2005/Server");
      server2 = server.createInstance();
      // ?? how do I release the connection???
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("Remote object creation: "+elapse+" msec");

server = null;
server = (Server)Naming.lookup("//"+host+":2005/Server");

// remote method call with variable parameters benchmark
// Ignore this just WARM UP
System.gc();
for (i = iter; i > 0; --i)
      x = server.methodA1(100);

// rmc with one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
      x = server.methodA1(100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// rmc with two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
      x = server.methodA2(100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

// rmc with three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
      x = server.methodA3(100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

// rmc with four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
      x = server.methodA4(100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// rmc with five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
      x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

// rmc with six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
      x = server.methodA6(100, 100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
```

```
        System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");
        //
        // Object send in array without cast benchmark
        //
        int[] sizes = {1, 10, 20, 30, 40, 50};
        for (int s = 0; s < 6; s++) {
            data = new Data[sizes[s]];
            for (j = 0; j < sizes[s]; j++) {
                data[j] = new Data();
            }
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodDA(data);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodDA(DOTBenchmark.rmi.Data["+sizes[s]+"]): "+elapse+" msec");
        }

        // numerical data transfer
        int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                            4096*8, 4096*16, 4096*32, 4096*64 };
        int iteration;
        // to send byte array
        for ( j = 0; j < array_sizes.length; j++) {
        bufb = new byte[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodB(bufb);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j])/elapse) + " KB/sec");
        }

        // to send int array
        for ( j = 0; j < 6; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodI(bufi);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
       }
       for ( j = 6; j < array_sizes.length; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            iteration = iter/10;
            start = System.currentTimeMillis();
            for (i = iteration; i > 0; --i) {
                server.methodI(bufi);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iteration);
            System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
       }

       // to send double array
       for ( j = 0; j < 4; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodD(bufd);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }

        for ( j = 4; j < array_sizes.length; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            iteration = iter/10;
            start = System.currentTimeMillis();
            for (i = iteration; i > 0; --i) {
                server.methodD(bufd);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iteration);
            System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }
    }
}
```

//Data.java

```
package DOTBenchmark.rmi;

import java.io.*;
import java.rmi.*;

public class Data implements Serializable {
    int a;
}
```

21.2 HORB

HORB 1.3b1

// Server.java

```
package DOTBenchmark.horb;

// public class Server {
public class Server implements java.io.Serializable {

    public int methodA1(int a1) { return a1; }
    public int methodA2(int a1, int a2) { return a1; }
    public int methodA3(int a1, int a2, int a3) { return a1; }
    public int methodA4(int a1, int a2, int a3, int a4) { return a1; }
    public int methodA5(int a1, int a2, int a3, int a4, int a5) { return a1; }
    public int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) { return a1; }
    public void methodDA(Data[] dat) { }
    public void methodB(byte[] a) {}
    public void methodI(int[] ia) {}
    public void methodD(double[] da){}
}
```

//Data.java

```
package DOTBenchmark.horb;
public class Data {
    int a;
}
```

//Client.java

```
package DOTBenchmark.horb;
import horb.orb.*;

class Client {
    public static void main(String argv[]) {
        int iter = 100;
        int i, j, x;
        long start, time;
        double elapse;
        byte bufb[];
        int  bufi[];
        double bufd[];
        String host = "localhost";

        if (argv.length == 2) {
              iter = Integer.parseInt(argv[0]);
              host = argv[1];
        } else if (argv.length == 1) {
              iter = Integer.parseInt(argv[0]);
        } else {
              System.err.println("java horb.examples.bench.horb.Client [iteration [host]]");
              System.exit(1);
        }

        Server_Proxy  servers[] = new Server_Proxy[iter];
        // remote object connection benchmark
        try {
              Thread.sleep(5000);
         } catch (Exception exxxx) {}

        HorbURL url = new HorbURL(host,"Server");
        Server_Proxy junk = new Server_Proxy(url);  // ignore the first one
        System.gc();
        start = System.currentTimeMillis();
        for (i = 0; i < iter; i++) {
              servers[i] = new Server_Proxy(url);
        }
        time = System.currentTimeMillis() - start;
        for( i = 0; i < iter; i++)
              servers[i]._release();
        elapse = (double)(time)/(double)(iter);
        System.out.println("bind to server(): "+elapse+" msec");

        /*
           try {
           Thread.sleep(120*1000);
           } catch (Exception exxxx) {}
        */

      // remote object creation benchmark
      url = new HorbURL(host,null);
      //url = new HorbURL("horb://"+host);
      System.gc();
      start = System.currentTimeMillis();
      for (i = 0; i < iter; i++) {
              servers[i] = new Server_Proxy(url);
              servers[i].Server();
        }
        time = System.currentTimeMillis() - start;
        for( i = 0; i < iter; i++)
              servers[i]._release();
        elapse = (double)(time)/(double)(iter);
        System.out.println("create server(): "+elapse+" msec");

        // prepare server for next benchmark tests
        HORB.useSerialization();

        Server_Proxy server = null;

        try {
           Thread.sleep(5000);
        } catch (Exception exxxx) {}

        server = new Server_Proxy("horb://"+host);
        // remote method call with variable parameters benchmark
        // load class files explicitely
        x = server.methodA1(0);
```

```
try {
    // load class files explicitely
    x = server.methodA1(0);
    // ignore this just WARM UP
    System.gc();
    for (i = iter; i > 0; --i)
        x = server.methodA1(100);
    // remote method invocation benchmark

    // one argument
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA1(100);
    time = System.currentTimeMillis() - startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA1(int a1): "+elapse+" msec");

    // two arguments
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA2(100, 100);
    time = System.currentTimeMillis() - startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA2(int, int): "+elapse+" msec");

    // three arguments
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA3(100, 100, 100);
    time = System.currentTimeMillis() – startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA3(int, int, int): "+elapse+" msec");

    // four arguments
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA4(100, 100, 100, 100);
    time = System.currentTimeMillis() - startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA4(int, int, int, int): "+elapse+" msec");

    // five arguments
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA5(100, 100, 100, 100, 100);
    time = System.currentTimeMillis() - startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

    // six arguments
    System.gc();
    startTime= System.currentTimeMillis();
    for (i = iter; i > 0; --i)
        x = server.methodA6(100, 100, 100, 100, 100, 100);
    time = System.currentTimeMillis() - startTime;
    elapse = (double)(time)/(double)(iter);
    println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");



    //
    // Object send in array without cast benchmark
    //

    int[] sizes = {1, 10, 20, 30, 40, 50};
```

```
        for (int s = 0; s < 6; s++) {
            Data[] data = new Data[sizes[s]];
            for (j = 0; j < sizes[s]; j++) {
                data[j] = new Data();
            }
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                server.methodDA(data);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodDA(Data["+sizes[s]+"]): "+elapse+" msec");
        }

        // Numerical array transfer benchmark
        // to send numerical data
        //
        int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4, 4096*8, 4096*16,
            4096*32, 4096*64};
        int iteration;

        // to send byte array
        for ( j = 0; j < array_sizes.length; j++) {
        bufb = new byte[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodB(bufb);
        }
        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec; "
        +((double)(array_sizes[j])/elapse) + " KB/sec");
}

  // to send int array
  for ( j = 0; j < 6; j++) {
        bufi = new int[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodI(bufi);
        }
   time = System.currentTimeMillis() - start;
   elapse = (double)(time)/(double)(iter);
   System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec; "
        +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
  }
  for ( j = 6; j < array_sizes.length; j++) {
        bufi = new int[array_sizes[j]];
        System.gc();
        iteration = iter/2;
        start = System.currentTimeMillis();
        for (i = iteration; i > 0; --i) {
            server.methodI(bufi);
        }
        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iteration);
        System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec; "
            +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
}

// to send double array
for ( j = 0; j < 4; j++) {
        bufd = new double[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodD(bufd);
        }
        time = System.currentTimeMillis() - start;
```

```
                elapse = (double)(time)/(double)(iter);
                System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec; "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
            for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/2;
                start = System.currentTimeMillis();
                for (i = iteration; i > 0; --i) {
                    server.methodD(bufd);
                }
                time = System.currentTimeMillis() - start;
                elapse = (double)(time)/(double)(iteration);
                System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec; "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
            }
        }
}
```

21.3 Voyager

Voyager 2.0.0 (modified by Michael Ta)

```
// Data.java
package DOTBenchmark.voy20;

public class Data implements java.io.Serializable {
    int a;
}

//IServer.java
package DOTBenchmark.voy20;
public interface IServer {
    int methodA1(int a1);
    int methodA2(int a1, int a2);
    int methodA3(int a1, int a2, int a3);
    int methodA4(int a1, int a2, int a3, int a4);
    int methodA5(int a1, int a2, int a3, int a4, int a5);
    int methodA6(int a1, int a2, int a3, int a4, int a5, int a6);
    void methodDA(Data[] dat);

    void methodB(byte[] a);
    void methodI(int[] ia);
    void methodD(double[] da);
}

//Server.java
package DOTBenchmark.voy20;

import com.objectspace.voyager.*;
import java.io.*;

public class Server implements IServer {
    static String serverName;
    static {
        try {
            serverName = java.net.InetAddress.getLocalHost().getHostName();
        } catch (Exception e) {
            serverName = "localhost";
        }
        System.out.println("Server name: "+serverName);
    }

    public Server () {
        super();
    }

    public int methodA1(int a1) { return a1; }
    public int methodA2(int a1, int a2) { return a1; }
    public int methodA3(int a1, int a2, int a3) { return a1; }
    public int methodA4(int a1, int a2, int a3, int a4) { return a1; }
    public int methodA5(int a1, int a2, int a3, int a4, int a5) { return a1; }
    public int methodA6(int a1, int a2, int a3, int a4, int a5, int a6) { return a1; }

    public void methodDA(Data[] dat) { }

    public void methodB(byte[] a) {}
    public void methodI(int[] ia) {}
    public void methodD(double[] da){}

    public static void main(String argv[]) throws Exception {
        Voyager.startup("//" + serverName + ":8000"); // startup Voyager server
        String serverclass = Server.class.getName();
        IServer server = (IServer) Factory.create(serverclass,"//" + serverName + ":8000");
        Namespace.rebind("VServer",server);
        System.out.println("bind done");
        System.out.println("Server is listening at " + serverName + ":8000/VServer");
    }
}
```

//Client.java

```
// for Voyager 2.0.0 production release
//
package DOTBenchmark.voy20;

import com.objectspace.voyager.*;

class Client {
    public static void main(String argv[]) {
        int iter = 100;
        int i, j, x;
        long start, time;
        double elapse;
        byte bufb[];
        int  bufi[];
        double bufd[];
        String host = "localhost";

         try {
         if (argv.length == 2) {
                 iter = Integer.parseInt(argv[0]);
                 host = argv[1];
           } else if (argv.length == 1) {
                 iter = Integer.parseInt(argv[0]);
           } else {
                 System.err.println("java Client [iteration [host:port]]");
                 System.exit(1);
           }
         } catch (Exception e) {
             System.err.println(e);
         }

         try {
             Voyager.startup(); // startup as client
             IServer server = null;
             System.out.println("looking up...");
             server = (IServer)Namespace.lookup("//"+host+":8000/VServer");   // ignore this
             System.out.println("done");
             System.gc();
             start = System.currentTimeMillis();
             for (i = 0; i < iter; i++) {
                 server = (IServer)Namespace.lookup("//"+host+":8000/VServer");
                 // ?? how do I release the connection???
             }
             time = System.currentTimeMillis() - start;
             elapse = (double)(time)/(double)(iter);
             System.out.println("Remote object connect: Namespace.lookup(): "+elapse+" msec");


             IServer[] servers = new IServer[iter];
```

```
// Remote object creation bench
System.gc();
start = System.currentTimeMillis();
for (i = 0; i < iter; i++) {
    // create a remote Server object
    servers[i] = (IServer) Factory.create("DOTBenchmark.voy20.Server", "//"+host+":8000" );
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("create server(): "+elapse+" msec");

// remote method call with variable parameters benchmark
server = null;
// newly created server for next bench
server = (IServer) Namespace.lookup("//"+host+":8000/VServer");
// load class files explicitely
// ignore this just WARM UP
System.gc();
for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}

// remote method benchmark
// one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA1(100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

// two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA2(100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

//  three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA3(100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

// four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA4(100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

// five arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
    x = server.methodA5(100, 100, 100, 100, 100);
}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
```

```
    System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

    // six arguments
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
        x = server.methodA6(100, 100, 100, 100, 100, 100);
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");

    // Object send in array without cast benchmark
    // one of two 1's is for warming up
     int[] sizes = {1, 1, 10, 20, 30, 40, 50};
    for (int s = 0; s < 7; s++) {
        Data[] data = new Data[sizes[s]];
        for (j = 0; j < sizes[s]; j++) {
            data[j] = new Data();
        }
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i)
        server.methodDA(data);
        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodDA(Data["+sizes[s]+"]): "+elapse+" msec");
    }

   // to send numerical data
   //
   int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4, 4096*8, 4096*16,
                              4096*32, 4096*64};
   int iteration;
   // to send byte array
   for ( j = 0; j < array_sizes.length; j++) {
        bufb = new byte[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodB(bufb);
        }
        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j])/elapse) + " KB/sec");
   }

   // to send int array
   for ( j = 0; j < 6; j++) {
        bufi = new int[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i) {
            server.methodI(bufi);
        }
        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
        }
        for ( j = 6; j < array_sizes.length; j++) {
            bufi = new int[array_sizes[j]];
            System.gc();
            iteration = iter/10;
            start = System.currentTimeMillis();
            for (i = iteration; i > 0; --i) {
                server.methodI(bufi);
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iteration);
            System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
                        +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
                }

                // to send double array
                for ( j = 0; j < 4; j++) {
                    bufd = new double[array_sizes[j]];
                    System.gc();
                    start = System.currentTimeMillis();
                    for (i = iter; i > 0; --i) {
                        server.methodD(bufd);
                    }
                    time = System.currentTimeMillis() - start;
                    elapse = (double)(time)/(double)(iter);
                    System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                        +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
                }
                for ( j = 4; j < array_sizes.length; j++) {
                    bufd = new double[array_sizes[j]];
                    System.gc();
                    iteration = iter/10;
                    start = System.currentTimeMillis();
                    for (i = iteration; i > 0; --i) {
                        server.methodD(bufd);
                    }
                    time = System.currentTimeMillis() - start;
                    elapse = (double)(time)/(double)(iteration);
                    System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                        +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
                }
                Voyager.shutdown();//  by Michael Ta
        } catch (Exception ex) {
                System.err.println(ex);
        }
    }
}
```

21.4 VisiBroker

VisiBroker 3.3 for Java

```
  interface Server {
  Server createInstance();
  long methodA1(in long a1);
  long methodA2(in long a1, in long a2);
  long methodA3(in long a1, in long a2, in long a3);
  long methodA4(in long a1, in long a2, in long a3, in long a4);
  long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
  long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

  typedef sequence<octet> ByteArray;
  typedef sequence<long> IntArray;
  typedef sequence<double> DoubleArray;
  void methodB(in ByteArray ba);
  void methodI(in IntArray ia);
  void methodD(in DoubleArray da);

  };
};
```

//ServerImpl.java

```
//package horb.examples.bench.vbroker;
//import netscape.security.*;
//import netscape.WAI.*;
//import org.omg.CORBA.*;
//import com.visigenic.vbroker.URLNaming.*;
/**
<p>
<ul>
<li> <b>Java Class</b> horb.examples.bench.Caffeine._example_Server
<li> <b>Source File</b> horb\examples\bench\Caffeine\_example_Server.java
<li> <b>IDL Absolute Name</b> ::Server
<li> <b>Repository Identifier</b> IDL:Server:1.0
</ul>
</p>
*/
package DOTBenchmark.visibroker;


public class ServerImpl extends vbroker._ServerImplBase {
    /** Construct a persistently named object. */
    public ServerImpl(java.lang.String name) {
        super(name);
    }

    /** Construct a transient object. */
    public ServerImpl() {
    super();
    }

    public static void main(String argv[]) {
    try {
            // initialize the ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

            // initialize the BOA (Basic Object Adaptor)
            org.omg.CORBA.BOA boa = orb.BOA_init();

            String name = "Server";
            String name2 = "ServerOT";
            DOTBenchmark.visibroker.ServerImpl server = new DOTBenchmark.visibroker.ServerImpl(name);
            DOTBenchmark.visibroker._example_ServerOT server2 = new
DOTBenchmark.visibroker._example_ServerOT(name2);
            //System.out.println("server was made: "+server);
            //System.out.println("server was made: "+server2);

            try {
                boa.obj_is_ready(server);
                boa.obj_is_ready(server2);
                System.out.println("ServerImpl is ready at: "+name);
                System.out.println("ServerOTImpl is ready at: "+name2);
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(1);
            }
            // wait for incoming requests
            boa.impl_is_ready();

        }
    catch (Exception e) {
            e.printStackTrace();
    }
  }

    public vbroker.Server createInstance() {
    ServerImpl obj = null;
    try {
            obj = new ServerImpl();
            return obj;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
    }

  public int methodA1(
    int a1
  ) {
    return 0;
  }

  public int methodA2(
    int a1,
    int a2
  ) {
    return 0;
  }

  public int methodA3(
    int a1,
    int a2,
    int a3
  ) {
    return 0;
  }

  public int methodA4(
    int a1,
    int a2,
    int a3,
    int a4
  ) {
    // implement operation...
    return 0;
  }

  public int methodA5(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5
  ) {
    return 0;
  }
  public int methodA6(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5,
    int a6
  ) {
    return 0;
  }
  public void methodB(
    byte[] ba
  ) {
  }

  public void methodI(
    int[] ia
  ) {
  }

  public void methodD(
    double[] da
  ) {
    // implement operation...
  }

}
```

//Client.java

```
//package horb.examples.bench.vbroker;
package DOTBenchmark.visibroker;


class Client {
      public static void main(String argv[]) {
      int iter = 100;
          int i, j, x;
          long start, time;
          double elapse;
          byte bufb[];
          int  bufi[];
          double bufd[];
          String host = "127.0.0.1";
          String host2;

          if (argv.length == 2) {
                 iter = Integer.parseInt(argv[0]);
                 host = argv[1];
          } else if (argv.length == 1) {
                 iter = Integer.parseInt(argv[0]);
          } else {
                 System.err.println("java DOTBenchmark.visibroker.Client [iteration] [host]");
                 System.exit(1);
          }
      // Initialize the ORB.
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv,null);

      //
      // remote object connection benchmark
      //
      vbroker.Server junk = vbroker.ServerHelper.bind(orb, "Server");  // ignore the first one

      start = System.currentTimeMillis();
      for (i = iter; i > 0; --i) {
                 vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
           //      server._release();
      }
      time = System.currentTimeMillis() - start;
      elapse = (double)(time)/(double)(iter);
      System.out.println("bind to server(): "+elapse+" msec");

    //     try {
```

```
//      Thread.sleep(30*1000);
//      } catch ( Exception e ) { }

    //
    // remote object creation benchmark
    //
    vbroker.Server server2 = null;
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
            vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");
            server2 = server.createInstance();
            //server2.methodA1(100);
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("object creation: "+elapse+" msec");


    //
    // prepare server for next benchmark tests
    //
    vbroker.Server server = vbroker.ServerHelper.bind(orb, "Server");

    // start evaluation

    //
    // remote method call with variable parameters benchmark
    //

    // one argument
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i)
            x = server.methodA1(100);

    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("int server.methodA1(int a1): "+elapse+" msec");

        // two arguments
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i)
            x = server.methodA2(100, 100);

    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("int server.methodA2(int, int): "+elapse+" msec");

        // three arguments
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i)
            x = server.methodA3(100, 100, 100);

    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

        // four arguments
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i)
            x = server.methodA4(100, 100, 100, 100);

    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

        // five arguments
    System.gc();
    start = System.currentTimeMillis();
```

```
for (i = iter; i > 0; --i)
          x = server.methodA5(100, 100, 100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

     // six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
          x = server.methodA6(100, 100, 100, 100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");


//
// numerical data transfer
//
int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                4096*8, 4096*16, 4096*32, 4096*64 };
int iteration;

// to send byte array
for ( j = 0; j < array_sizes.length; j++) {
     bufb = new byte[array_sizes[j]];
     System.gc();
     start = System.currentTimeMillis();
     for (i = iter; i > 0; --i)
     server.methodB(bufb);

     time = System.currentTimeMillis() - start;
     elapse = (double)(time)/(double)(iter);
     System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
          +((double)(array_sizes[j])/elapse) + " KB/sec");
}


// to send int array
for ( j = 0; j < 6; j++) {
          bufi = new int[array_sizes[j]];
          System.gc();
          start = System.currentTimeMillis();
          for (i = iter; i > 0; --i)
               server.methodI(bufi);

          time = System.currentTimeMillis() - start;
          elapse = (double)(time)/(double)(iter);
          System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
               +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
}

for ( j = 6; j < array_sizes.length; j++) {
          bufi = new int[array_sizes[j]];
          System.gc();
          iteration = iter/10;  // reduce the iteration 10 times to save time!
          start = System.currentTimeMillis();
          for (i = iteration; i > 0; --i)
               server.methodI(bufi);

          time = System.currentTimeMillis() - start;
          elapse = (double)(time)/(double)(iteration);
          System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
               +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
}


// to send double array
for ( j = 0; j < 4; j++) {
          bufd = new double[array_sizes[j]];
```

```
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
                server.methodD(bufd);

            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
    }

    for ( j = 4; j < array_sizes.length; j++) {
            bufd = new double[array_sizes[j]];
            System.gc();
            iteration = iter/10; // reduce the iteration 10 times
            start = System.currentTimeMillis();
            for (i = iteration; i > 0; --i)
                server.methodD(bufd);

            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iteration);
            System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
    }
    }
}
```

//DataOT.java

```
package DOTBenchmark.visibroker;

public class DataOT implements java.io.Serializable {
    private int data;

    public void set(int n) {
    data = n;
    }

    public int get() {
    return data;
    }
}
```

//ServerOTImpl.java

```
package DOTBenchmark.visibroker;

public class ServerOTImpl {

    /** Construct a persistently named object. */
    public ServerOTImpl(java.lang.String name) {
  //    super(name);
    }

    /** Construct a transient object. */
    public ServerOTImpl() {
    super();
    }

    public static void main(String argv[]) {
    try {
        // initialize the ORB
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

        // initialize the BOA (Basic Object Adaptor)
        org.omg.CORBA.BOA boa = orb.BOA_init();
        String host = null, name = null;

        name = "Server";
        _example_ServerOT server = new _example_ServerOT(name);
        System.out.println("server was made: "+server);
        try {
            boa.obj_is_ready(server);
```

```
                System.out.println("ServerImpl is ready at: "+name);
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(1);
            }
            // wait for incoming requests
            boa.impl_is_ready();
        }
        catch (Exception e) {
                e.printStackTrace();
        }
        }

}
```

//ClientOT.java

```
package DOTBenchmark.visibroker;

import java.io.*;
import java.util.*;

class ClientOT {
        public static void main(String argv[]) {
        int iter = 100;
            int i, x;
            long start, time;
            double elapse;
            //    sequence_of_DataHolder data = new sequence_of_DataHolder();
            String host = "127.0.0.1";
            String host2;

            if (argv.length == 2) {
                iter = Integer.parseInt(argv[0]);
                host = argv[1];
            } else if (argv.length == 1) {
                iter = Integer.parseInt(argv[0]);
            } else {
                System.err.println("java horb.examples.bench.Caffeine.Client [iteration]");
                System.exit(1);
            }

        //
        // Object send in array without cast benchmark
        //
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(argv,null);

        // Object send benchmark

        DOTBenchmark.visibroker.ServerOT server = DOTBenchmark.visibroker.ServerOTHelper.bind(orb, "Server");


        int[] sizes = {1, 10, 20, 30, 40, 50};
        for (int s = 0; s < 6; s++) {

            DOTBenchmark.visibroker.DataOT[] data = new DOTBenchmark.visibroker.DataOT[sizes[s]];
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i)
            server.methodDA(data);

            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iter);
            System.out.println("void server.methodDA(Data["+sizes[s]+"]): "+
                 elapse+" msec");
        }
        }
}
```

21.5 OrbixWeb

OrbixWeb 3.0 Modified by Michael Ta

```
//Server.idl

 interface Server {
  Server createInstance();
  long methodA1(in long a1);
  long methodA2(in long a1, in long a2);
  long methodA3(in long a1, in long a2, in long a3);
  long methodA4(in long a1, in long a2, in long a3, in long a4);
  long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
  long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

  typedef sequence<octet> ByteArray;
  typedef sequence<long> IntArray;
  typedef sequence<double> DoubleArray;
  void methodB(in ByteArray ba);
  void methodI(in IntArray ia);
  void methodD(in DoubleArray da);

  };
```

```
// ServerImpl.java
package DOTBenchmark.orbix;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb.Features.Config;
import org.omg.CORBA.ORB;
import ServerPackage.*;

//package horb.examples.bench.orbix;
//import netscape.security.*;

import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.SystemException;

public class ServerImpl implements _ServerOperations {
  public static void main(String argv[]) throws Exception { ORB.init();
      String host = null, name = null;
      try {
         host = java.net.InetAddress.getLocalHost().getHostName();
      } catch (Exception e) {
         e.printStackTrace();
         System.exit(1);
      }
      ServerImpl server;
      Server serverTie;

      server = new ServerImpl();
      serverTie = new _tie_Server(server);

      System.out.println("server was made");
      _CORBA.Orbix.impl_is_ready("Server");
    }

  public Server createInstance() {
    ServerImpl obj = null;
    try {
      obj = new ServerImpl();
      Server serverTie = new _tie_Server(obj);
      return serverTie;
    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }
```

```
/**
<p>
Operation: <b>::Server::methodA1</b>.
<pre>
  long methodA1(
    in long a1
  );
</pre>
</p>
*/
public int methodA1(
  int a1
) {
  // implement operation...
  return a1;
}
/**
<p>
Operation: <b>::Server::methodA2</b>.
<pre>
  long methodA2(
    in long a1,
    in long a2
  );
</pre>
</p>
*/
public int methodA2(
  int a1,
  int a2
) {
  // implement operation...
  return a1;
}
/**
<p>
Operation: <b>::Server::methodA3</b>.
<pre>
  long methodA3(
    in long a1,
    in long a2,
    in long a3
  );
</pre>
</p>
*/
public int methodA3(
  int a1,
  int a2,
  int a3
) {
  // implement operation...
  return a1;
}
/**
<p>
Operation: <b>::Server::methodA4</b>.
<pre>
  long methodA4(
    in long a1,
    in long a2,
    in long a3,
    in long a4
  );
</pre>
</p>
*/
public int methodA4(
  int a1,
  int a2,
  int a3,
    int a4
  ) {
    // implement operation...
    return a1;
  }
  /**
  <p>
  Operation: <b>::Server::methodA5</b>.
  <pre>
    long methodA5(
      in long a1,
      in long a2,
      in long a3,
      in long a4,
      in long a5
    );
  </pre>
  </p>
  */
  public int methodA5(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5
  ) {
    // implement operation...
    return a1;
  }
  /**
  <p>
  Operation: <b>::Server::methodA6</b>.
  <pre>
    long methodA6(
      in long a1,
      in long a2,
      in long a3,
      in long a4,
      in long a5,
      in long a6
    );
  </pre>
  </p>
  */
  public int methodA6(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5,
    int a6
  ) {
    // implement operation...
    return a1;
  }

//  public void methodDA(horb.examples.bench.orbix._Server.DataArray a1) {}

  public void methodB(byte[] ba) {}
  public void methodI(int[] ba) {}
  public void methodD(double[] ba) {}
}
```

//Client.java

```
//package Project.Benchmark.Orbix.v3;

package DOTBenchmark.orbix;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb.Features.Config;
import org.omg.CORBA.ORB;
```

```
//package horb.examples.bench.orbix;
import IE.Iona.OrbixWeb._CORBA;


class Client {
  public static void main(String argv[]) throws Exception {
                ORB.init();
    int iter = 100;
    int i, j, x;
    long start, time;
    double elapse;
    byte[] bufb;
    int[]  bufi;
    double[] bufd;
    String host = "127.0.0.1";
    Server server = null;

    if (argv.length == 2) {
      iter = Integer.parseInt(argv[0]);
      host = argv[1];
    } else if (argv.length == 1) {
      iter = Integer.parseInt(argv[0]);
    } else {
      System.err.println("java DOTBenchmark.orbix.Client [iteration [host]]");
      System.exit(1);
    }

    //
    // remote object connection benchmark
    //
    server = ServerHelper.bind(":Server", host);   // ignore this one

    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
      server = ServerHelper.bind(":Server", host);
//      server.release();
//_OrbixWeb.ORB(ORB.init()).closeConnection(server);
      // ?? how do I release the connection??
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("bind to server(): "+elapse+" msec");

    try {
      Thread.sleep(30*1000);
    } catch (Exception e) {}

    //
    // remote object creation benchmark
    //
    Server server2 = null;
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i) {
      server = ServerHelper.bind(":Server", host);
      server2 = (Server)server.createInstance();
//      server2.release();
//_OrbixWeb.ORB(ORB.init()).closeConnection(server2);
//      server.release();
//_OrbixWeb.ORB(ORB.init()).closeConnection(server);
      // ?? how do I release the connection??
    }
    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("create remote object: "+elapse+" msec");


    server = ServerHelper.bind(":Server", host);
//    System.out.println("bind to server success");
    // load class files explicitely
    x = server.methodA1(0);
    // start evaluation
// integer send benchmark

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA1(100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA2(100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA3(100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA4(100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA5(100, 100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
  x = server.methodA6(100, 100, 100, 100, 100, 100);
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");

//
// to send numerical data
//
int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4, 4096*8, 4096*16,
     4096*32, 4096*64};
int iteration;

// to send byte array
for ( j = 0; j < array_sizes.length; j++) {
  bufb = new byte[array_sizes[j]];
  System.gc();
  start = System.currentTimeMillis();
  for (i = iter; i > 0; --i)
    server.methodB(bufb);
  time = System.currentTimeMillis() - start;
  elapse = (double)(time)/(double)(iter);
  System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
     +((double)(array_sizes[j])/elapse) + " KB/sec");
}
```

```
    // to send int array
    for ( j = 0; j < 6; j++) {
      bufi = new int[array_sizes[j]];
      System.gc();
      start = System.currentTimeMillis();
      for (i = iter; i > 0; --i)
        server.methodI(bufi);
      time = System.currentTimeMillis() - start;
      elapse = (double)(time)/(double)(iter);
      System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
        +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
    }
    for ( j = 6; j < array_sizes.length; j++) {
      bufi = new int[array_sizes[j]];
      System.gc();
      iteration = iter/10;
      start = System.currentTimeMillis();
      for (i = iteration; i > 0; --i)
        server.methodI(bufi);
      time = System.currentTimeMillis() - start;
      elapse = (double)(time)/(double)(iteration);
      System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
        +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
    }

    // to send double array
    for ( j = 0; j < 4; j++) {
      bufd = new double[array_sizes[j]];
      System.gc();
      start = System.currentTimeMillis();
      for (i = iter; i > 0; --i)
        server.methodD(bufd);
      time = System.currentTimeMillis() - start;
      elapse = (double)(time)/(double)(iter);
      System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
        +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
    }
    for ( j = 4; j < array_sizes.length; j++) {
      bufd = new double[array_sizes[j]];
      System.gc();
      iteration = iter/10;
      start = System.currentTimeMillis();
      for (i = iteration; i > 0; --i)
        server.methodD(bufd);
      time = System.currentTimeMillis() - start;
      elapse = (double)(time)/(double)(iteration);
      System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
        +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
    }

  }
}
```

21.6 DCOM

DCOM

// dcoms.odl

```
[ uuid (29EE83A0-37AE-11d2-9539-F7F021215302) ]
library Ldcoms
{
	importlib("stdole32.tlb");

	[	uuid(29EE83A1-37AE-11d2-9539-F7F021215302),
		helpstring("IDConnect Interface")]
	dispinterface IDConnect
	{
	properties:
	methods:
		[id(1)]int dserv_1( [in] int p1 );
		[id(2)]int dserv_2( [in] int p1, [in] int p2 );
		[id(3)]int dserv_3( [in] int p1, [in] int p2, [in] int p3 );
		[id(4)]int dserv_4( [in] int p1, [in] int p2, [in] int p3, [in] int p4 );
		[id(5)]int dserv_5( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5 );
		[id(6)]int dserv_6( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6 );
		[id(7)]int dserv_7( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6, [in] int p7 );
		[id(8)]int dserv_8( [in] int p1, [in] int p2, [in] int p3, [in] int p4, [in] int p5, [in] int p6, [in] int p7, [in] int p8
);
		[id(9)]void dserv_iarray( [in] int * p1 );
		[id(10)]void dserv_barray( [in] unsigned char * p1 );
		[id(11)]void dserv_darray( [in] double * p1 );
		[id(12)]BSTR hello();
	}

	[	uuid(29EE83A2-37AE-11d2-9539-F7F021215302),
		helpstring("CDConnect Class")]
	coclass CDConnect
	{
		dispinterface IDConnect;
	};
};
```

//CDConnect.java

```
//
//
// CDConnect
//
//

import dcoms.*;
import com.ms.com.*;
import java.net.*;

public class CDConnect implements IDConnect
{
	public int dserv_1( int p1 ) throws ComException{
		int x1 = p1;
		return 100; }
	public int dserv_2( int p1, int p2 ) throws ComException{
		int x1 = p1;
		int x2 = p2;
		return 100; }
	public int dserv_3( int p1, int p2, int p3 ) throws ComException{
		int x1 = p1;
		int x2 = p2;
		int x3 = p3;
		return 100; }
	public int dserv_4( int p1, int p2, int p3, int p4 ) throws ComException{
		int x1 = p1;
		int x2 = p2;
		int x3 = p3;
		int x4 = p4;
```

```
        return 100; }
    public int dserv_5( int p1, int p2, int p3, int p4, int p5 ) throws ComException{
        int x1 = p1;
        int x2 = p2;
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        return 100; }
    public int dserv_6( int p1, int p2, int p3, int p4, int p5, int p6 ) throws ComException{
        int x1 = p1;
        int x2 = p2;
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        int x6 = p6;
        return 100; }
    public int dserv_7( int p1, int p2, int p3, int p4, int p5, int p6, int p7 ) throws ComException{
        int x1 = p1;
        int x2 = p2;
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        int x6 = p6;
        int x7 = p7;
        return 100; }
    public int dserv_8( int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8 ) throws ComException{
        int x1 = p1;
        int x2 = p2;
        int x3 = p3;
        int x4 = p4;
        int x5 = p5;
        int x6 = p6;
        int x7 = p7;
        int x8 = p8;
        return 100; }
    public void dserv_iarray( int [] p1 ){
        int x,i;
        for( i = 0; i < p1.length; i++ )
            x = p1[i];

        return;
    }
    public void dserv_barray( byte [] p1 ){
        int i;
        byte x;
        for( i = 0; i < p1.length; i++ )
            x = p1[i];

        return;
    }
    public void dserv_darray( double [] p1 ){
        int i;
        double x;
        for( i = 0; i < p1.length; i++ )
            x = p1[i];

        return;
    }
    public String hello() throws ComException {
    String host = null;
   try {
     host = InetAddress.getLocalHost().getHostName();
     System.out.println(host);
   } catch (UnknownHostException e) {
     System.out.println(e);
     return "**";
   }
    return host;
  }
}
```

//client1.java

```
//
//
// client
//
// This file is modified by Michael Ta on 19/08/98
// Just cleanup at the moment

//package dcoms;

import dcoms.*;

public class client1
{
    public static void main( String argv[] ){
        int iter = 100;
        String host = "localhost";
        long start, time;
        double elapse;
        IDConnect cdc;
        int i;
        int x;
        byte [] bufb;
        int [] bufi;
        double [] bufd;

        if (argv.length == 2) {
            iter = Integer.parseInt(argv[0]);
            host = argv[1];
        } else if (argv.length == 1) {
            iter = Integer.parseInt(argv[0]);
        } else {
            System.err.println("Client [iteration [host]]");
            System.exit(1);
        }

        cdc = (IDConnect)new dcoms.CDConnect();
        System.out.println("server: "+cdc.hello());
        //System.out.println("All time measured in ms");

        //Object Creation Benchmark
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            cdc = ( IDConnect )new dcoms.CDConnect();

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("Object creation bench:"+elapse+"msec");
        //System.out.println("\t\t\t"+time+" msec(total)");

        cdc = (IDConnect)new dcoms.CDConnect();

        //   method call bench int( int *1)
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            x = cdc.dserv_1( 100 );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( int *1 ):"+elapse+"msec");
        //System.out.println("\t\t\t"+time+"msec(total)");

        //   method call bench int( int *2)
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            x = cdc.dserv_2( 100, 100 );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( int *2 ):"+elapse+"msec");
```

```
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *3)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_3( 100, 100, 100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *3 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *4)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_4( 100, 100, 100,100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *4 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *5)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_5( 100, 100, 100, 100,100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *5 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *6)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_6( 100, 100, 100, 100, 100,100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *6 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *7)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_7( 100, 100, 100, 100, 100, 100,100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *7 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

//   method call bench int( int *8)
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     x = cdc.dserv_8( 100, 100, 100, 100, 100, 100, 100,100 );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int *8 ):"+elapse+"msec");
//System.out.println("\t\t\t"+time+"msec(total)");

// transfering byte array 1
System.gc();
bufb = new byte[1];
start = System.currentTimeMillis();
```

```
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[1] ):"+elapse+"msec; "+(double)(1/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");

// transfering byte array 256
bufb = new byte[256];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[256] ):"+elapse+"msec; "+(double)(256/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 512
bufb = new byte[512];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[512] ):"+elapse+"msec; "+(double)(512/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 1024
bufb = new byte[1024];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[1024] ):"+elapse+"msec; "+(double)(1024/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 2048
bufb = new byte[2048];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[2048] ):"+elapse+"msec; "+(double)(2048/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 4096
bufb = new byte[4096];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[4096] ):"+elapse+"msec; "+(double)(4096/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");
```

```
// transfering byte array 4096 *2
bufb = new byte[4096*2];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[4096*2] ):"+elapse+"msec; "+(double)(4096*2/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 4096*4
bufb = new byte[4096*4];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[4096*4] ):"+elapse+"msec; "+(double)(4096*4/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering byte array 4096 *8
bufb = new byte[4096*8];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_barray( bufb );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( byte[4096*8] ):"+elapse+"msec; "+(double)(4096*8/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");

// transfering int array 1
System.gc();
bufi = new int[1];
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[1] ):"+elapse+"msec; "+(double)(1/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 256
bufi = new int[256];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[256] ):"+elapse+"msec; "+(double)(256/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 512
bufi = new int[512];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );
```

```
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[512] ):"+elapse+"msec; "+(double)(512/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 1024
bufi = new int[1024];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[1024] ):"+elapse+"msec; "+(double)(1024/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 2048
bufi = new int[2048];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[2048] ):"+elapse+"msec; "+(double)(2048/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 4096
bufi = new int[4096];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[4096] ):"+elapse+"msec; "+(double)(4096/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 4096 *2
bufi = new int[4096*2];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[4096*2] ):"+elapse+"msec; "+(double)(4096*2/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 4096*4
bufi = new int[4096*4];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[4096*4] ):"+elapse+"msec; "+(double)(4096*4/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering int array 4096 *8
```

```
bufi = new int[4096*8];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_iarray( bufi );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( int[4096*8] ):"+elapse+"msec; "+(double)(4096*8/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");

// transfering double array 1
System.gc();
bufd = new double[1];
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_darray( bufd );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( double[1] ):"+elapse+"msec; "+(double)(1/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering double array 256
bufd = new double[256];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_darray( bufd );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( double[256] ):"+elapse+"msec; "+(double)(256/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering double array 512
bufd = new double[512];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_darray( bufd );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( double[512] ):"+elapse+"msec; "+(double)(512/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering double array 1024
bufd = new double[1024];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_darray( bufd );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int method( double[1024] ):"+elapse+"msec; "+(double)(1024/elapse) + " KB/sec");
//System.out.println("\t\t\t"+time+"msec(total)");


// transfering double array 2048
bufd = new double[2048];
System.gc();
start = System.currentTimeMillis();
for( i = iter; i > 0; --i )
     cdc.dserv_darray( bufd );

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
```

```
        System.out.println("int method( double[2048] ):"+elapse+"msec; "+(double)(2048/elapse) + " KB/sec");
        //System.out.println("\t\t\t"+time+"msec(total)");


        // transfering double array 4096
        bufd = new double[4096];
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            cdc.dserv_darray( bufd );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( double[4096] ):"+elapse+"msec; "+(double)(4096/elapse) + " KB/sec");
        //System.out.println("\t\t\t"+time+"msec(total)");


        // transfering double array 4096 *2
        bufd = new double[4096*2];
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            cdc.dserv_darray( bufd );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( double[4096*2] ):"+elapse+"msec; "+(double)(4096*2/elapse) + "
KB/sec");
        //System.out.println("\t\t\t"+time+"msec(total)");


        // transfering double array 4096*4
        bufd = new double[4096*4];
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            cdc.dserv_darray( bufd );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( double[4096*4] ):"+elapse+"msec; "+(double)(4096*4/elapse) + "
KB/sec");
        //System.out.println("\t\t\t"+time+"msec(total)");


        // transfering double array 4096 *8
        bufd = new double[4096*8];
        System.gc();
        start = System.currentTimeMillis();
        for( i = iter; i > 0; --i )
            cdc.dserv_darray( bufd );

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int method( double[4096*8] ):"+elapse+"msec; "+(double)(4096*8/elapse) + "
KB/sec");
        //System.out.println("\t\t\t"+time+"msec(total)");
    }
}
```

21.7 Java IDL

Java IDL by Michael Ta

//Server.idl
```
module JavaIDLServer {
    interface Server {
    Server createInstance();
    long methodA1(in long a1);
    long methodA2(in long a1, in long a2);
    long methodA3(in long a1, in long a2, in long a3);
    long methodA4(in long a1, in long a2, in long a3, in long a4);
    long methodA5(in long a1, in long a2, in long a3, in long a4, in long a5);
    long methodA6(in long a1, in long a2, in long a3, in long a4, in long a5, in long a6);

    typedef sequence<octet> ByteArray;
    typedef sequence<long> IntArray;
    typedef sequence<double> DoubleArray;

    void methodB(in ByteArray ba);
    void methodI(in IntArray ia);
    void methodD(in DoubleArray da);

    };
};
```

// ServerImpl.java

```
package DOTBenchmark.javaidl;
import JavaIDLServer.*;  // The package containing our stubs.

import org.omg.CosNaming.*;
// The package containing special exceptions thrown by the name service.
import org.omg.CosNaming.NamingContextPackage.*;

// All CORBA applications need these classes.
import org.omg.CORBA.*;


public class ServerImpl extends _ServerImplBase {
    /** Construct a persistently named object. */
    //public ServerImpl(java.lang.String name) {
        //super(name);
    //}

    /** Construct a transient object. */
    public ServerImpl() {
    super();
    }

    public static void main(String args[]) {
    try {
             // Create and initialize the ORB
            ORB orb = ORB.init(args, null);

            // Create the servant and register it with the ORB
            ServerImpl serverRef = new ServerImpl();
            orb.connect(serverRef);

            // Get the root naming context
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);


            // Bind the object reference in naming
            NameComponent nc = new NameComponent("Server", " ");
            NameComponent path[] = {nc};
            ncRef.rebind(path, serverRef);
            System.out.println("server was made: "+serverRef);

            // Wait for invocations from clients
            java.lang.Object sync = new java.lang.Object();
            synchronized(sync){
            sync.wait();
            }

        }
    catch (Exception e) {
            e.printStackTrace();
    }
  }

    public Server createInstance() {
    ServerImpl obj = null;
    try {
            obj = new ServerImpl();
            return obj;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
    }

  public int methodA1(
    int a1
  ) {
    return 0;
  }

  public int methodA2(
    int a1,
    int a2
  ) {
    return 0;
  }

  public int methodA3(
    int a1,
    int a2,
    int a3
  ) {
    return 0;
  }

  public int methodA4(
    int a1,
    int a2,
    int a3,
    int a4
  ) {
    // implement operation...
    return 0;
  }

  public int methodA5(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5
  ) {
    return 0;
  }

  public int methodA6(
    int a1,
    int a2,
    int a3,
    int a4,
    int a5,
    int a6
  ) {
    return 0;
  }
  public void methodB(
    byte[] ba
  ) {
  }

  public void methodI(
    int[] ia
  ) {
  }

  public void methodD(
    double[] da
  ) {
    // implement operation...
  }

}
```

//Client.java

```
package DOTBenchmark.javaidl;
import JavaIDLServer.*;  // The package containing our stubs.

import org.omg.CosNaming.*;
import org.omg.CORBA.*;

class Client {
      public static void main(String args[]) {
      int iter = 100;
          int i, j, x;
          long start, time;
          double elapse;
          byte bufb[];
          int  bufi[];
          double bufd[];
          String host = "127.0.0.1";
          String host2;

          if (args.length == 2) {
                 iter = Integer.parseInt(args[0]);
                 host = args[1];
          } else if (args.length == 1) {
                 iter = Integer.parseInt(args[0]);
          } else {
                 System.err.println("java DOTBenchmark.visibroker.Client [iteration] [host]");
                 System.exit(1);
          }

          try{
            // Create and initialize the ORB
                 ORB orb = ORB.init(args, null);

                 // Get the root naming context
                 org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
                 NamingContext ncRef = NamingContextHelper.narrow(objRef);


                 /* remote object connection benchmark */
                 // Resolve the object reference in naming
                 NameComponent nc = new NameComponent("Server", " ");
                 NameComponent path[] = {nc};
                 Server junk = ServerHelper.narrow(ncRef.resolve(path));

                 start = System.currentTimeMillis();

                 for (i = iter; i > 0; --i) {
                      Server server = ServerHelper.narrow(ncRef.resolve(path));
                 }
```

```
     time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("bind to server(): "+elapse+" msec");


/* remote object creation benchmark */
Server server2 = null;
start = System.currentTimeMillis();
for (i = iter; i > 0; --i) {
          Server server = ServerHelper.narrow(ncRef.resolve(path));
          server2 = server.createInstance();

}
time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("object creation: "+elapse+" msec");



/* prepare server for next benchmark tests */

Server server = ServerHelper.narrow(ncRef.resolve(path));

// start evaluation

//
// remote method call with variable parameters benchmark
//

// one argument
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
          x = server.methodA1(100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA1(int a1): "+elapse+" msec");

     // two arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
          x = server.methodA2(100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA2(int, int): "+elapse+" msec");

     // three arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
          x = server.methodA3(100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA3(int, int, int): "+elapse+" msec");

     // four arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
          x = server.methodA4(100, 100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA4(int, int, int, int): "+elapse+" msec");

     // five arguments
System.gc();
```

```
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA5(100, 100, 100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA5(int, int, int, int, int): "+elapse+" msec");

    // six arguments
System.gc();
start = System.currentTimeMillis();
for (i = iter; i > 0; --i)
        x = server.methodA6(100, 100, 100, 100, 100, 100);

time = System.currentTimeMillis() - start;
elapse = (double)(time)/(double)(iter);
System.out.println("int server.methodA6(int, int, int, int, int, int): "+elapse+" msec");


//
// numerical data transfer
//
int[] array_sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                4096*8, 4096*16, 4096*32, 4096*64 };
int iteration;

// to send byte array
for ( j = 0; j < array_sizes.length; j++) {
    bufb = new byte[array_sizes[j]];
    System.gc();
    start = System.currentTimeMillis();
    for (i = iter; i > 0; --i)
    server.methodB(bufb);

    time = System.currentTimeMillis() - start;
    elapse = (double)(time)/(double)(iter);
    System.out.println("void server.methodB(byte["+array_sizes[j]+"]): "+elapse+" msec, "
        +((double)(array_sizes[j])/elapse) + " KB/sec");
}


// to send int array
for ( j = 0; j < 6; j++) {
        bufi = new int[array_sizes[j]];
        System.gc();
        start = System.currentTimeMillis();
        for (i = iter; i > 0; --i)
            server.methodI(bufi);

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
}

for ( j = 6; j < array_sizes.length; j++) {
        bufi = new int[array_sizes[j]];
        System.gc();
        iteration = iter/10;  // reduce the iteration 10 times to save time!
        start = System.currentTimeMillis();
        for (i = iteration; i > 0; --i)
            server.methodI(bufi);

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iteration);
        System.out.println("void server.methodI(int["+array_sizes[j]+"]): "+elapse+" msec, "
            +((double)(array_sizes[j]*4)/elapse) + " KB/sec");
}


// to send double array
for ( j = 0; j < 4; j++) {
```

```
                bufd = new double[array_sizes[j]];
                System.gc();
                start = System.currentTimeMillis();
                for (i = iter; i > 0; --i)
                    server.methodD(bufd);

                time = System.currentTimeMillis() - start;
                elapse = (double)(time)/(double)(iter);
                System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }

        for ( j = 4; j < array_sizes.length; j++) {
                bufd = new double[array_sizes[j]];
                System.gc();
                iteration = iter/10; // reduce the iteration 10 times to save time!
                start = System.currentTimeMillis();
                for (i = iteration; i > 0; --i)
                    server.methodD(bufd);

                time = System.currentTimeMillis() - start;
                elapse = (double)(time)/(double)(iteration);
                System.out.println("void server.methodD(double["+array_sizes[j]+"]): "+elapse+" msec, "
                    +((double)(array_sizes[j]*8)/elapse) + " KB/sec");
        }


    } catch(Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
    }

}
```

21.8 SocketJava

Socket Java

```
//SockServer.java
//package horb.examples.bench.socket;
package DOTBenchmark.javasocket;

import java.io.*;
import java.net.*;

class SockServer {
     public static void main(String argv[]) throws IOException {
     int iter = 100;
     int i, j, k, x;
     byte b;
     double d;
     int realcount, previouscount;
     DataInputStream in;
     DataOutputStream out;
     int[] sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                              4096*8, 4096*16, 4096*32, 4096*64 };
     byte[] bufb = new byte[4096*64];
          int iteration;



     if (argv.length == 1) {
               iter = Integer.parseInt(argv[0]);
     } else {
               System.err.println("java horb.examples.bench.socket.SockServerClient iteration");
               System.err.println("java DOTBenchmark.javasocket.SockServer iteration");
               System.exit(1);
     }

     // load class files explicitely
     ServerSocket serverSocket = new ServerSocket(8887, 10);
         Socket socket = serverSocket.accept();
         in = new DataInputStream(new BufferedInputStream(socket.getInputStream(), 1024*32));
         out = new DataOutputStream(new BufferedOutputStream(socket.getOutputStream(), 1024*32));

     // start evaluation
     // integer send benchmark

         x = in.readInt();
         out.writeInt(100);
         out.flush();

     // warm up
     for (j = 1; j <= 1; j++) {
               for (i = iter; i > 0; --i) {
                    for (k = 0; k < j; k++) {
                         x = in.readInt();
                    }
                    out.writeInt(100);
                    out.flush();
               }
     }

     for (j = 1; j <= 6; j++) {
               for (i = iter; i > 0; --i) {
                    for (k = 0; k < j; k++) {
                         x = in.readInt();
                    }
                    out.writeInt(100);
                    out.flush();
               }
     }

     // to send byte array
```

```
    //for ( j = 0; j < sizes.length; j++) {
    for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
        for (i = iter; i > 0; --i) {
                in.readFully(bufb, 0, sizes[j]);
                out.writeInt(100);
                out.flush();
            }
    }

    for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
        iteration = iter/100;
        for (i = iteration; i > 0; --i) {
                in.readFully(bufb, 0, sizes[j]);
                out.writeInt(100);
                out.flush();
            }
    }

    // to send int array
    //for ( j = 0; j < sizes.length; j++) {
    for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
        for (i = iter; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++) {
                    x = in.readInt();
                }
                out.writeInt(100);
                out.flush();
            }
    }

        for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
            iteration = iter/100;
        for (i = iteration; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++) {
                    x = in.readInt();
                }
                out.writeInt(100);
                out.flush();
            }
    }

    // to send double array
    //for ( j = 0; j < sizes.length; j++) {
    for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
        for (i = iter; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++) {
                    d = in.readDouble();
                }
                out.writeInt(100);
                out.flush();
            }
    }

    for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
        iteration = iter/100;
        for (i = iteration; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++) {
                    d = in.readDouble();
                }
                out.writeInt(100);
                out.flush();
            }
    }

    }
}
```

// SockClient.java

```
//package horb.examples.bench.socket;\
package DOTBenchmark.javasocket;

import java.io.*;
import java.net.*;

class SockClient {
    public static void main(String argv[]) throws IOException {
    int iter = 100;
        int i, j, k, x;
        DataInputStream in;
        DataOutputStream out;
        long start, time;
        double elapse;
        byte bufb[];
        String host = "localhost";
        int[] sizes = {1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                    4096*8, 4096*16, 4096*32, 4096*64 };
          int iteration;


    if (argv.length == 2) {
                iter = Integer.parseInt(argv[0]);
                host = argv[1];
    } else if (argv.length == 1) {
                iter = Integer.parseInt(argv[0]);
    } else {
                System.err.println("java horb.examples.bench.socket.SockClient [iteration [host]]");
                System.err.println("java DOTBenchmark.javasocket.SockClient [iteration [host]]");
                System.exit(1);
    }

        Socket socket = new Socket(host, 8887);

        in = new DataInputStream(new BufferedInputStream(socket.getInputStream(), 1024*32));
        out = new DataOutputStream(new BufferedOutputStream(socket.getOutputStream(), 1024*32));

        // start evaluation

        // integer send benchmark
        out.writeInt(100);
        out.flush();
        x = in.readInt();

        // warm up
        for (j = 1 ; j <= 1; j++) {
                System.gc();
                for (i = iter; i > 0; --i) {
                      for (k = 0; k < j; k++) {
                            out.writeInt(100);
                      }
                      out.flush();
                      x = in.readInt();
                }
        }

    for (j = 1 ; j <= 6; j++) {
                System.gc();
                start = System.currentTimeMillis();
                for (i = iter; i > 0; --i) {
                      for (k = 0; k < j; k++) {
                            out.writeInt(100);
                      }
                      out.flush();
                      x = in.readInt();
                }
                time = System.currentTimeMillis() - start;
                elapse = (double)(time)/(double)(iter);
                System.out.println("int socket(int a1*"+j+"): "+elapse+" msec");
    }

    //
    // numerical data transfer
    //

    // to send byte array
```

```
//for ( j = 0; j < sizes.length; j++) {
for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
        bufb = new byte[sizes[j]];
        System.gc();
        start = System.currentTimeMillis();

        for (i = iter; i > 0; --i) {
            out.write(bufb, 0, sizes[j]);
            out.flush();
            x = in.readInt();
        }

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("byte["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
            +((double)(sizes[j])/elapse) + " KB/sec");
}

    for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
        bufb = new byte[sizes[j]];
        System.gc();
        iteration = iter/100;
    start = System.currentTimeMillis();

        for (i = iteration; i > 0; --i) {
            out.write(bufb, 0, sizes[j]);
            out.flush();
            x = in.readInt();
        }

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iteration);
        System.out.println("byte["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
            +((double)(sizes[j])/elapse) + " KB/sec");
}


// to send integer array
//for ( j = 0; j < sizes.length; j++) {
for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
        System.gc();
        start = System.currentTimeMillis();

        for (i = iter; i > 0; --i) {
            for (k = 0; k < sizes[j]; k++) {
                out.writeInt(100);
            }
            out.flush();
            x = in.readInt();
        }

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("int["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
            +((double)(sizes[j]*4)/elapse) + " KB/sec");
}

    for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
        System.gc();
        iteration = iter/100;
        start = System.currentTimeMillis();

        for (i = iteration; i > 0; --i) {
            for (k = 0; k < sizes[j]; k++) {
                out.writeInt(100);
            }
            out.flush();
            x = in.readInt();
        }

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iteration);
```

```
            System.out.println("int["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
                +((double)(sizes[j]*4)/elapse) + " KB/sec");
    }

    // to send double array
    // for ( j = 0; j < sizes.length; j++) {
    for ( j = 0; j < 5; j++) { //for 1000 iter by MTA
            System.gc();
            start = System.currentTimeMillis();
            for (i = iter; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++) {
                    out.writeDouble(100.0);
                }
                out.flush();
                x = in.readInt();
        }

        time = System.currentTimeMillis() - start;
        elapse = (double)(time)/(double)(iter);
        System.out.println("double["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
            +((double)(sizes[j]*8)/elapse) + " KB/sec");
    }

        for ( j = 5; j < sizes.length; j++) { //for 1000 iter by MTA
            System.gc();
            iteration = iter/100;
            start = System.currentTimeMillis();
            for (i = iteration; i > 0; --i) {
                for (k = 0; k < sizes[j]; k++)
                    out.writeDouble(100.0);
                out.flush();
                x = in.readInt();
            }
            time = System.currentTimeMillis() - start;
            elapse = (double)(time)/(double)(iteration);
            System.out.println("double["+sizes[j]+"] transfer by socket : "+elapse+" msec, "
                +((double)(sizes[j]*8)/elapse) + " KB/sec");
    }
    }
}
```

21.9 Socket C

Socket C

//server.c

```
/*
 * server.c
 * Copyright 1998 HORB Open
 * Copyright 1998 Electrotechnical Laboratory, AIST, Japan
 * Author: Kazuo Yamamoto (HORB Open, MSS)
 *
 */
#include <stdio.h>
#include <winsock.h>

#define PORT_NO 8887
#define MAXSIZE 4096*64
#define TMPBUFFSIZE 32*1024

typedef unsigned char Byte;

// Prototypes
int  inReadInt(void);
void inReadFully(char *, int);
void inReadIntBuff(int *, int );
void inReadDoubleBuff(double *, int );
void outWriteInt(int);
void outFlushInt(void);


// Global Variable
union u_tag {
    Byte     b[MAXSIZE];
    int      i[MAXSIZE];
    double d[MAXSIZE];
} oBuff,iBuff;

char  tmpbuff[TMPBUFFSIZE];

int           outIndex = 0;
int           inIndex = 0;

SOCKET s, sc;

int main(int argc, char *argv[])
{
    int  ret, len;
    int  iter = 100;
    int i, j, k, x;
    char host[256];
    SOCKADDR_IN addr, addrc;
    WSADATA wsaData;
    HOSTENT *shost;
    Byte b;
    int sizes[]={1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                4096*8, 4096*16, 4096*32, 4096*64};
    Byte bufb[4096*64];

    // check arguments
    if (argc == 2) {
        iter = atoi(argv[1]);
    } else {
        fprintf( stderr,"usage: %s iteration\n", argv[0]);
        exit(-1);
    }

    /* initialize winsock */
    WSAStartup( (WORD)MAKEWORD(2,0), &wsaData);
    // creat socket
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if (s == INVALID_SOCKET) {
        printf("... Invalid socket [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // get local host
    ret = gethostname(host, sizeof(host));
    if (ret == SOCKET_ERROR) {
        printf("... gethostname error [%d]\n", WSAGetLastError());
        exit(-1);
    }
   shost = gethostbyname(host);
    if (shost == NULL) {
        printf("... error gethostbyname [%d]\n", WSAGetLastError());
        exit(-1);
    }
    // dump hostent
    // printf("shost->name=%s\n", shost->h_name);
    // printf("shost->h_addrtype=%d\n", shost->h_addrtype);
    // printf("shost->h_length=%d\n", shost->h_length);

    // bind
    memset(&addr, 0x00, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT_NO);
    memcpy(&addr.sin_addr.s_addr, shost->h_addr, shost->h_length);
    ret = bind(s, (SOCKADDR*)&addr, sizeof(addr));
    if (ret == SOCKET_ERROR) {
        printf("... bind error [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // listen
    ret = listen(s, 4);
    if (ret == SOCKET_ERROR) {
        printf("... listen error [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // accept
    memset(&addrc, 0x00, sizeof(addrc));
    len = sizeof(addrc);
    sc = accept(s, (SOCKADDR*)&addrc, &len);
    if (sc == INVALID_SOCKET) {
        printf("... accept error [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // start evaluation
    // integer send benchmark
    x = inReadInt();
    outWriteInt(100);
    outFlushInt();

    // Warm up ?
    for (j=1; j<=1; j++) {
        for (i = iter; i>0; --i){
            for (k=0; k<j; k++)
                x = inReadInt();
            outWriteInt(100 * j);
            outFlushInt();
        }
    }

    for (j=1; j<=6; j++) {
        for (i = iter; i>0; --i){
            for (k=0; k<j; k++)
                x = inReadInt();
            outWriteInt(100 * j);
            outFlushInt();
            }
        }

        //
        // numerical data transfer
        //

        // to send byte array
        for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
            for (i=iter; i>0; --i) {
                inReadFully(iBuff.b, sizes[j]);
                outWriteInt(100);
                outFlushInt();
            }
        }


        // for time saving
        if (iter>=100) {
            iter /=10;
            printf("*** iteration change to %d\n", iter);
        }

        // to send int array
        for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
            for (i=iter; i>0; --i) {
                //for (k=0; k<sizes[j]; k++)
                //  x = inReadInt();
                inReadIntBuff(iBuff.i, sizes[j]);
                outWriteInt(100);
                outFlushInt();
            }
        }

        // to send double array
        for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
            for (i=iter; i>0; --i) {
                //for (k=0; k<sizes[j]; k++)
                //  x = inReadDouble();
                inReadDoubleBuff(iBuff.d, sizes[j]);
                outWriteInt(100);
                outFlushInt();
            }
        }

        closesocket(s);
        closesocket(sc);
        WSACleanup();
        return 0;
}

/*
*       inReadInt();
*/
int inReadInt( void )
{
        int  len, x;

        len = recv(sc, (char*)&x, sizeof(int), 0);
        if (len == SOCKET_ERROR) {
            printf("... receive error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        if (len != sizeof(int)) {
            printf("... inReadInt Error receive length=%d\n", len);
        }else {
            return ntohl((u_long)x);
        }
}


/*
```

```
*         inReadDouble();
*/
int inReadDouble( void )
{
     int  len, size;
     int i=0;
     double x;
     char * p;

     size = sizeof(double);
     p = (char *)&x;

     do {
          len = recv(sc, p, size, 0);
          if (len == SOCKET_ERROR) {
               printf("... receive error [%d]\n", WSAGetLastError());
               exit(-1);
          }
          // printf("... inReadDouble(): %d byte receive(%d) \n", len, i++);
          size -= len;
          p += len;
     } while (size > 0);

     return x;
}


/*
*         inReadFully()
*/
void inReadFully(char *pbuf, int size)
{
     int  len, rcvsize;
     int  tatalsize=0, loopcout=0;
     char *p;

     do {
          if (size < TMPBUFFSIZE)
               rcvsize = size;
          else
               rcvsize = TMPBUFFSIZE;
          len = recv(sc, tmpbuff, rcvsize, 0);
          if (len == SOCKET_ERROR) {
               printf("... receive error [%d]\n", WSAGetLastError());
               exit(-1);
          }
          size -= len;
          memcpy(pbuf, tmpbuff, len);
          pbuf += len;
          // tatalsize += len;
          // loopcout++;
          // if (size == 0)
          //    printf("tatal size = %d (%d)\n", tatalsize, loopcout);
     } while (size > 0);
}


/*
* intReadIntBuff();
*/
void inReadIntBuff(int *pbuf, int size)
{
     int i, j=0;
     int  loop=0;
     int  len, rcvsize;
     char *p;

     size *= sizeof(int);

     do {
          if (size < TMPBUFFSIZE)
               rcvsize = size;
```

```
        else
            rcvsize = TMPBUFFSIZE;
        len = recv(sc, tmpbuff, rcvsize, 0);
        if (len == SOCKET_ERROR) {
            printf("... receive error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        size -= len;

        for(i=0; i<len/sizeof(int); i+=sizeof(int),j++) {
            pbuf[j] = ntohl((u_long)tmpbuff[i]);
        }
    } while (size > 0);
}


/*
* intReadDoubleBuff();
*/
void inReadDoubleBuff(double *pbuf, int size)
{
    int i, j=0;
    int  loop=0, tatalsize=0;
    int  len, rcvsize;
    u_int *p;
    u_int val = 0;

    size *= sizeof(double);
    p = (u_int *)pbuf;

    do {
        if (size < TMPBUFFSIZE)
            rcvsize = size;
        else
            rcvsize = TMPBUFFSIZE;
        len = recv(sc, tmpbuff, rcvsize, 0);
        if (len == SOCKET_ERROR) {
            printf("... receive error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        size -= len;

        for(i=0; i<len/sizeof(double); i+=sizeof(double),p+=sizeof(double)) {
          *(p+0) = ntohl((u_long)tmpbuff[i]);
          *(p+1) = ntohl((u_long)tmpbuff[i+4]);
        }
    } while (size > 0);
}


/*
*   outWriteInt()
*/
void outWriteInt(int a)
{
    oBuff.i[outIndex++]=htons((u_short)a);
}


/*
*   outFlushInt()
*/
void outFlushInt(void)
{
    int i=0;
    int  size, sent;
    int  index=0;

    size = outIndex * sizeof(int);
    outIndex = 0;

    do {
        //printf("size=%d (%d)\n", size, i++);
        sent = send(sc, (char *)&oBuff.i[index], size, 0);
        if (sent == SOCKET_ERROR) {
            printf("... send error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        size -= sent;
        index += sent/sizeof(int);
    } while (size > 0);
}
```

//client.c

```
/* Client.c */
/* This file is modified by Michael Ta on Sun July 12th 1998 */
/* Change _timeb to timeb
    Change _ftime to ftime
*/

#include <stdio.h>
#include <time.h>
#include <sys/timeb.h>
#include <winsock.h>

#define PORT_NO 8887
#define MAXSIZE 4096*64

typedef unsigned char Byte;

// Prototypes
void outWriteInt(int);
void outFlushInt(void);
void outWriteDouble(double);
void outFlushDouble(void);
int  inReadInt(void);
void outWriteByteArray( char *, int );
void outFlushByteArray(void);

// Global Variable
union u_tag {
    Byte    b[MAXSIZE];
    int     i[MAXSIZE];
    double d[MAXSIZE];
} oBuff,iBuff;

int         outIndex = 0;
int         inIndex = 0;

SOCKET      s;

int main(int argc, char *argv[])
{
    int  ret, len;
    int  iter = 100;
    int i, j, k, x;
    SOCKADDR_IN addr;
    WSADATA wsaData;
    HOSTENT *shost;
    struct timeb start_time, end_time;  // edited by Michael Ta
    long diff;
    double elapse;
    Byte *bufb;
    char host[256]="localhost";
    int sizes[]={1, 256, 512, 1024, 2048, 4096, 4096*2, 4096*4,
                 4096*8, 4096*16, 4096*32, 4096*64};

    /* check arguments */
    if (argc == 3) {
        iter = atoi(argv[1]);
        strcpy(host, argv[2]);
    } else if (argc == 2) {
        iter = atoi(argv[1]);
    } else {
        fprintf( stderr,"usage: %s [iteration [host]]\n", argv[0]);
        exit(-1);
    }

    /* initialize winsock */
    WSAStartup( (WORD)MAKEWORD(2,0), &wsaData);

    // create socket
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if (s == INVALID_SOCKET) {
        printf("... Invalid socket [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // get host entry
    shost = gethostbyname(host);
    if (shost == NULL) {
        printf("... error gethostbyname [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // connect
    memset(&addr, 0x00, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT_NO);
    memcpy(&addr.sin_addr.s_addr, shost->h_addr, shost->h_length);

    //addr.sin_addr.s_addr = htons(INADDR_ANY);
    ret = connect(s, (SOCKADDR*)&addr, sizeof(addr));
    if (ret != 0) {
        printf("... connet error [%d]\n", WSAGetLastError());
        exit(-1);
    }

    // start evaluation
    //
    // integer send benchmark
  //
    outWriteInt(100);
    outFlushInt();
    x = inReadInt();

    // warm up?
    for (j=1; j<=1; j++) {
        for (i=iter; i>0; --i) {
            for (k=0; k<j; k++) {
                outWriteInt(100);
            }
            outFlushInt();
            x = inReadInt();
        }
    }

    for (j=1; j<=6; j++) {
        ftime(&start_time);                 // start of time
        for (i=iter; i>0; --i) {
            for (k=0; k<j; k++) {
                outWriteInt(100);
            }
            outFlushInt();
            x = inReadInt();
        }
        ftime(&end_time);                   // end of time
        diff = (end_time.time - start_time.time)*1000
            + ((long)end_time.millitm - (long)start_time.millitm);
        elapse = (double)diff/(double)iter;
        printf("int socket(int a1*%d): %7.5f msec\n", j, elapse);
    }

    //
    // numerical data transfer
    //
    // to send byte array
    for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
        bufb = malloc( sizes[j]);
        if (bufb == NULL) {
            printf(" ... malloc error\n");
            exit(-1);
        }
        ftime(&start_time);            // start of time
        for (i=iter; i>0; --i) {
            outWriteByteArray(bufb, sizes[j]);
            outFlushByteArray();
            x = inReadInt();
            //printf("x=%d\n", x);
        }
        ftime(&end_time);              // end of time
        diff = (end_time.time - start_time.time)*1000
            + ((long)end_time.millitm - (long)start_time.millitm);
        elapse = (double)diff/(double)iter;
        printf("byte[%d] transfer by socket : %7.5f msec %7.5f KB/sec\n",
                sizes[j], elapse, (double)(sizes[j]/elapse));
        free(bufb);
    }


    // for time saving
    if (iter>=100) {
        iter /=10;
        printf("*** iteration change to %d\n", iter);
    }


    // to send int array
    for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
        ftime(&start_time);            // start of time
        for (i=iter; i>0; --i) {
            for (k=0; k<sizes[j]; k++)
                outWriteInt(100);
            outFlushInt();
            x = inReadInt();
        }
        ftime(&end_time);              // end of time
        diff = (end_time.time - start_time.time)*1000
            + ((long)end_time.millitm - (long)start_time.millitm);
        elapse = (double)diff/(double)iter;
        printf("int[%d] transfer by socket : %7.5f msec %7.5f KB/sec\n",
                sizes[j], elapse, (double)(sizes[j]*4/elapse));
    }


    // to send double array
    for (j=0; j<sizeof(sizes)/sizeof(sizes[0]); j++) {
        ftime(&start_time);            // start of time
        for (i=iter; i>0; --i) {
            for (k=0; k<sizes[j]; k++)
                outWriteDouble(100.0);
            outFlushDouble();
            x = inReadInt();
        }
        ftime(&end_time);              // end of time
        diff = (end_time.time - start_time.time)*1000
            + ((long)end_time.millitm - (long)start_time.millitm);
        elapse = (double)diff/(double)iter;
        printf("double[%d] transfer by socket : %7.5f msec %7.5f KB/sec\n",
                sizes[j], elapse, (double)(sizes[j]*8/elapse));
    }

    closesocket(s);
    WSACleanup();

    return 0;
}
```

```
void outWriteInt(int a)
{
    oBuff.i[outIndex++]=htonl((u_long)a);
}


void outFlushInt(void)
{
    int i=0;
    int  size, sent;
    int  index=0;

    size = outIndex * sizeof(int);
    outIndex = 0;

    do {
        //printf("size=%d (%d)\n", size, i++);
        sent = send(s, (char *)&oBuff.i[index], size, 0);
        if (sent == SOCKET_ERROR) {
            printf("... send error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        size -= sent;
        index += sent/sizeof(int);
    } while (size > 0);
}

/*
*   outWriteDouble()
*/
void outWriteDouble(double a)
{
    u_long *p, *q;

    p = (u_long*)&oBuff.d[outIndex++];
    q = (u_long*)&a;
    *(p+0) = htonl(*(q+0));
    *(p+1) = htonl(*(q+1));

    //oBuff.d[outIndex++]=htonl((unsigned long)a);
    //oBuff.d[outIndex++]=a;
}


/*
*   outFlushDouble()
*/
void outFlushDouble(void)
{
    int i=0;
    int  size, sent;
    int  index=0;

    size = outIndex * sizeof(double);
    outIndex = 0;

    do {
        sent = send(s, (char *)&oBuff.d[index], size, 0);
        if (sent == SOCKET_ERROR) {
            printf("... send error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        //printf("... outFlushDouble: sent %d bytes\n", sent);
        size -= sent;
        index += sent/sizeof(double);
    } while (size > 0);
}


void outWriteByteArray( char * pbuf, int size)
```

```
{
    memcpy(oBuff.b, pbuf, size);
    outIndex += size;
}


void outFlushByteArray()
{
    int i=0;
    int  size, sent;
    int  index=0;

    size = outIndex * sizeof(char);
    outIndex = 0;

    do {
        sent = send(s, (char *)&oBuff.b[index], size, 0);
        if (sent == SOCKET_ERROR) {
            printf("... send error [%d]\n", WSAGetLastError());
            exit(-1);
        }
        size -= sent;
        index += sent/sizeof(char);
    } while (size > 0);
}


int inReadInt( void )
{
    int  len, x;

    len = recv(s, (char*)&x, sizeof(int), 0);
    if (len == SOCKET_ERROR) {
        printf("... receive error [%d]\n", WSAGetLastError());
        exit(-1);
    }
    if (len != sizeof(int)) {
        printf("... inReadInt Error receive length=%d\n");
    }else {
        return ntohl((u_long)x);
    }
}
```