Distributed Objects and CORBA

Chikkaiah Manjunath
NetDB research team
E-mail: manjunat@cse.uta.edu

Client

Server
Remote Object

CSE@UTA
Outline

- Introduction to Distributed Objects
- Introduction to CORBA
- CORBA Architecture
- Java IDL
- CORBA IDL interfaces and modules
- CORBA development process
- Java IDL examples:
  - Hello Distributed Object example
  - Banking Distributed Object example
- References
Introduction to Distributed Objects

Mainframe Applications

Terminals

Mainframe

Client/Server Applications

Windows Client

Mac Client

Unix Client

Fat Client

Corporate data

Oracle, DB2, MS SQL, ...

Back end Data
Distributed Applications

Windows Client
Mac Client
Unix Client

Thin Client

Middle Tier Services
NT/Unix/AS 400

Corporate data
Oracle, DB2, MS SQL, ……

Back end Data

Distributed Objects
Distributed Objects

- Distributed Applications
  => Data are Distributed
  => Computation is Distributed
  => Users are Distributed

- Distributed Object Systems
  => All entities are modeled as objects
  => Popular paradigm for object-oriented distributed applications
  => CORBA is a standard for Distributed Object Systems
Distributed Object Infrastructures

- CORBA (Common Object Request Broker Architecture)
- Java RMI (Remote Method Invocation)
- DCOM (Distributed Component Object Model)
- EJB (Enterprise Java Beans)
Introduction to CORBA

- CORBA defines an Architecture for Distributed Objects
- It allows distributed, heterogeneous collections of objects to Interoperate
Introduction to CORBA

OMG is responsible for defining CORBA

Industry consortium with over 800 member companies formed to develop a distributed object standard

For more information visit: http://www.omg.org
CORBA Architecture

Sample Client code

RemoteObject o = .......;

o.method();
Major ORB Vendors

- IONA – Orbix
- Inprise – Visibroker
- HP – ORB plus
- SUN – Java IDL
Java IDL

- Java IDL is a CORBA compliant distributed object technology.
- Anywhere, Anytime and Any Language.
- Java IDL is similar to RMI. But RMI is limited only to Java.
- ORB is a class library that enables low-level communication between Java IDL applications and other CORBA compliant applications.
CORBA IDL

- CORBA IDL is defined by OMG. Also known as OMG IDL or IDL.
- Each language that supports CORBA has its own IDL mapping.
- Orbix supports mapping for C++.
  Java IDL supports mapping for Java.
- IDL is used to describe only the data types and object interfaces.
IDL Data types

**IDL simple data types**
- Basic data types are similar to C, C++ or Java.
  - long, short, float, double, char, boolean, ………

**IDL complex data types**
- string – sequence of characters
  - string <256> message;  //bounded
  - string message;  //unbounded
- sequence – one dimensional array
  - sequence <float, 100> arr;  //bounded
  - sequence <float> arr;  //unbounded

**IDL user defined data types**
- typedef, enum, const, struct, ………
CORBA development process

IDL definition

IDL compiler

stub source

Java or C++ compiler

Client Source

Client Program

Java or C++ compiler

skeleton source

Java or C++ compiler

Object Source

Object Implementation
Java IDL Examples

Hello Distributed Object Example

Step(1) Define the Interface  [Hello.idl]

// Hello.idl
module HelloApp
{
    interface Hello
    {
        string sayHello();
    }
};

Hello Distributed Object Example

Step(2) Implement the Server  [HelloServer.java]

import HelloApp.*; // The package containing our stubs.
import org.omg.CosNaming.*; // HelloServer will use the naming service.
import org.omg.CosNaming.NamingContextPackage.*; // The package containing special
// exceptions thrown by the name service.
import org.omg.CORBA.*; // All CORBA applications need these classes.

public class HelloServer
{
    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
            HelloServant helloRef = new HelloServant();
            orb.connect(helloRef);
// 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("Hello", "");
NameComponent path[] = {nc};
ncRef.rebind(path, helloRef);

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

catch(Exception e)
{
    System.err.println("ERROR: " + e);
    e.printStackTrace(System.out);
}
class HelloServant extends _HelloImplBase
{
    public String sayHello()
    {
        return "\nHello world !!\n";
    }
}

**Step(3) Implement the Client  [HelloClient.java]**

import HelloApp.*;       // The package containing our stubs.
import org.omg.CosNaming.*;  // HelloClient will use the naming service.
import org.omg.CORBA.*;      // All CORBA applications need these classes.

public class HelloClient
{
    public static void main(String args[])
    {
        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);

// Resolve the object reference in naming
NameComponent nc = new NameComponent("Hello", "");
NameComponent path[] = {nc};
Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

// Call the Hello server object and print results
String hello = helloRef.sayHello();
System.out.println(hello);
}
catch(Exception e)
{
    System.out.println("ERROR : " + e);
e.printStackTrace(System.out);
}
}
Step(4) Compile all files
> idlj –fall Hello.idl
> javac *.java

Step(5) start the name server
> tnameserv –ORBInitialPort 4000

Step(6) Run the HelloServer.class
> java HelloServer –ORBInitialPort 4000

Step(7) Run the HelloClient.class
> Java HelloClient –ORBInitialPort 4000

Step(8) Results
Hello world !!
Banking Distributed Object Example

**Step(1) Define the Interface [Bank.idl]**

// Bank.idl
module Banking
{
    typedef long Amount;
    interface Account
    {
        long balance( );
        oneway void createaccount(in Amount x);
        Amount deposit(in Amount x);
        Amount withdraw(in Amount x);
    }
};
Step(2) Implement the Server  [BankServer.java]

class BankServant extends _AccountImplBase
{
    private int balance;

    public void createaccount(int amount)
    {
        this.balance = amount;
    }

    public int balance()
    {
        return this.balance;
    }

    public int deposit(int amount)
    {
        this.balance += amount;
        return this.balance;
    }
}
public int withdraw(int amount) {
    if (this.balance >= amount) {
        this.balance -= amount;
        return this.balance;
    } else {
        return -1; // illegal transaction
    }
}

**Step(3)** Implement the Client [BankClient.java]

**Steps(4) – (8) are similar to previous example.**
References