;login:
THE MAGAZINE OF USENIX & SAGE
June 2001 • Volume 26 • Number 3

inside:

PROGRAMMING
Using CORBA with Java

by Prithvi Rao

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

using CORBA with java

A Mini Napster, Part II

by Prithvi Rao

Prithvi Rao is the co-founder of KiwiLabs, which specializes in software engineering methodology and Java/CORBA training. He has also worked on the development of the MACH OS and a real-time version of MACH. He is an adjunct faculty at Carnegie Mellon and teaches in the Heinz School of Public Policy and Management.

<prithvi+@ux4.sp.cs.cmu.edu>

Introduction

In Part I of this two-part series, I presented the development of a Java client/server application using CORBA. Specifically, I embarked on the development of a "mini Napster" example.

The development effort began with the writing of a Napster IDL (Interface Definition Language) and ended with the description of the functionality of the files that were generated as a result of compiling Napster.idl using the idltojava compiler.

In this article I wish to present the client and server code to complete this example. I hope to demonstrate that writing a CORBA application is neither overwhelming nor limited to those practitioners with advanced knowledge of software engineering and a general software development background.

Indeed, I successfully teach this topic to many students at Carnegie Mellon University who have not had a significant amount of experience in writing software or distributed client-server applications.

The Napster Server

In this section I present the implementation of the Napster Server (NapsterServer.java).

Import the CORBA packages for naming and locating CORBA objects:

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.util.*;
```

Import the package that contains the server skeletons:

```
import Napster.*;
```

Implement the interface generated by the idltojava compiler:

```
public class NapsterServer extends _NapsterServerIImplBase {
    private Vector records_database;
```

Build the constructor for the NapsterServer class:

```
    public NapsterServer()
    {
    records_database = new Vector(10, 5);
    }
```

Write the findItemInServer method:

```
    public Record findItemInServer(String albumName)
    {
        // Get the record with the highest version number
    Enumeration record_iter         = records_database.elements();
    int highest_version_number      = -1;
        int index                   = -1;

        while(record_iter.hasMoreElements())
        {
        Record record = (Record) record_iter.nextElement();
```

```
            if(record.album_name.equals(albumName))
            {
                if(record.version > highest_version_number)
                {
                    highest_version_number = record.version;
                    index = records_database.indexOf(record);
                }
            }
        }

        // Check if a record with album_name = albumName was found
    if(highest_version_number == -1)
    {
        // The record does not exist in the database so return a dummy record
        // with an empty album name. Also you need to set the other fields to
        // some dummy values even though you don't need them logically,
        // because otherwise CORBA will throw an Exception
            Record dummy_record          = new Record();
            dummy_record.album_name      = " ";
            dummy_record.artist_name     = " ";
            dummy_record.owner_name      = " ";
            dummy_record.version         = -1;
            return dummy_record;
        }

        // If we are here then we must have found the record in the database,
        // so return the found record
        return (Record) records_database.elementAt(index);
    }
```

Write the addRecordInServer method:

```
  public String addRecordInServer(Record desiredRecord)
    {
        // Get the record with the highest version number
    Enumeration record_iter = records_database.elements();
        while(record_iter.hasMoreElements())
        {
        Record record = (Record)record_iter.nextElement();
            if((record.album_name.equals(desiredRecord.album_name)) &&
            (record.artist_name.equals(desiredRecord.artist_name)))
            if(record.version >= desiredRecord.version)
            return "Duplicate Record, Record Not Added to the Server";
        } // while

        // We can now safely add the record to the database
    records_database.addElement(desiredRecord);
    return "Record Successfully Added";
  }
```

Write the deleteItemInServer method:

```
  public boolean deleteItemInServer(Record desiredRecord)
    {
        int num_of_records       = records_database.size();
        boolean record_deleted   = false;
        for(int i = 0; i < num_of_records; i++)
        {
```

```
            Record record = (Record) records_database.elementAt(i);
            if((record.album_name.equals(desiredRecord.album_name)) &&
                    (record.artist_name.equals(desiredRecord.artist_name)))
            {
                records_database.removeElementAt(i);
                record_deleted = true;
            }
        }

        return record_deleted;
    }
```

Write the updateRecordInServer method

```
public boolean updateRecordInServer(Record desiredRecord, String
                                    newOwner)
{
        // Get the record with the highest version number
    Enumeration record_iter         = records_database.elements();
        int highest_version_number = 0;
        int index                  = 0;

        while(record_iter.hasMoreElements())
        {
            Record record = (Record)record_iter.nextElement();
            if((record.album_name.equals(desiredRecord.album_name)) &&
                    (record.artist_name.equals(desiredRecord.artist_name)))
            {
                if(record.version > highest_version_number)
                {
                    highest_version_number = record.version;
                    index = records_database.indexOf(record);
                }
            }
        }

        // If the record was not found then return false
        if(index == 0)
            return false;

        // Otherwise update the record
        Record record          = (Record) records_database.elementAt(index);
        record.owner_name      = newOwner;
        record.version +       = 1;

        records_database.addElement(record);

        return true;
    }
}
```

The Napster Main Class

The following is the Napster main class (Napster.java) in which all the CORBA work happens. First import all the CORBA packages:

```
import Napster.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
```

```
import org.omg.CORBA.*;

public class Napster
{
    public static void main(String args[])
    {
        try
        {
```

Create the ORB and initialize it. This is where the ORB knows how to find the location of the NameServer with which to register the name of the server objects.

```
            ORB orb = ORB.init(args, null);
```

Create a Server Object here:

```
            NapsterServer napster_server = new NapsterServer();
```

Connect the Server Object to the ORB:

```
            orb.connect(napster_server);
```

Get a reference to the nameserver:

```
            org.omg.CORBA.Object objRef =
                                    orb.resolve_initial_references("NameService");
```

Get an object reference that is a "Java" object and not a CORBA object:

```
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

Make sure that the server object can be located using "NapsterServer":

```
            NameComponent nc = new NameComponent("NapsterServer", "");
```

Save this in a component array:

```
        NameComponent path[]    = {nc};
```

Bind this component name with the ORB so that the client can get a reference to the server object:

```
        ncRef.rebind(path, napster_server);
```

Make sure that you can have multiple clients connect to the server:

```
            java.lang.Object sync = new java.lang.Object();
            synchronized(sync)
            {
                sync.wait();
            } // try

            catch(Exception e) {
                System.err.println("ERROR: " + e);
                e.printStackTrace(System.out);
            }
        }

    }
```

The Napster Client

In this section I present the Napster Client (NapsterClient.java). First import the CORBA packages and other Java packages:

```
import Napster.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class NapsterClient
{
private NapsterServerI napster_server;
private BufferedReader stdin;
```

Write the constructor for the NapsterClient:

```
public NapsterClient(NapsterServerI p_napster_server, BufferedReader p_stdin)
{
napster_server = p_napster_server;
stdin   = p_stdin;
}
```

Write the method to get a record:

```
public void getRecord()
{
try {
     System.out.print("Please Enter the Album Name: ");
String album_name      = stdin.readLine();
     System.out.print("Please Enter Your Name: ");
String user_name       = stdin.readLine() ;

Record found_record    = new Record();

found_record           = napster_server.findItemInServer(album_name) ;

// If we don't find the record then give an appropriate message to the user
if(!found_record.album_name.equals(album_name))
     System.out.print("The Album Name That You Entered Does Not Exist");
else
{
// We found the record. So create a new record with the current user
// as owner and an increased version number to make it the latest record
     System.out.println("Information Regarding the Album:");
     System.out.println("Album Name : " + found_record.album_name);
     System.out.println("Artist Name   : " + found_record.artist_name);
     System.out.println("Owner         : " + found_record.owner_name);
     System.out.println("Version       : " + found_record.version);

Record new_record = new Record();
     new_record.album_name    = album_name;
     new_record.artist_name   = found_record.artist_name;
     new_record.owner_name    = user_name;
     new_record.version       = ++found_record.version;

String server_response = napster_server.addRecordInServer(new_record);

     System.out.println("New Record with the Following Info Added to the
                         Server:");
     System.out.println("Album Name  : " + new_record.album_name);
     System.out.println("Artist Name   : " + new_record.artist_name);
     System.out.println("Owne          : " + new_record.owner_name);
```

```
        System.out.println("Version: " + new_record.version);

        System.out.print(server_response);
        } // else
        } // try

            catch(Exception e)
            {
            System.out.println("Error: " + e);
            e.printStackTrace(System.out);
        } // catch
    }
```

Write the method to add a record:

```
    public void addRecord()
    {
        try
            {
            // Get the fields of the new record to create

            System.out.print("Please Enter Your Name: ");
            String user_name       =  stdin.readLine();

            System.out.print("Please Enter the Album Name: ");
            String album_name     =  stdin.readLine();

            System.out.print("Please Enter the Artist Name: ");
            String artist_name      =  stdin.readLine() ;

            // Create a new record and initialize its fields

            Record new_record         =     new Record();
            new_record.album_name =     album_name;
            new_record.owner_name =     user_name;
            new_record.artist_name  =     artist_name;
            new_record.version        =     1;

        String server_response = napster_server.addRecordInServer(new_record);
            system.out.print(server_response);

            } //try
        catch(Exception e)
        {
        System.out.println("Error: " + e);
        e.printStackTrace(System.out);
        } // catch
    }
```

Write a method to delete a method:

```
    public void deleteRecord()
    {
        try
            {
            System.out.print("Please Enter the Album Name: ");
            String album_name     = stdin.readLine()j;

            System.out.print("Please Enter the Artist Name: ");
            String artist_name      = stdin.readLine();
```

```
        // Create a record that you would like to be deleted from the server
        // database

        Record new_record          =  new Record();
        new_record.album_name      =  album_name;
        new_record.owner_name      =  new String("Aravind");
        new_record.artist_name     =  artist_name;
        new_record.version         =  1;

    System.out.println("Before Calling Server Delete");
    boolean record_deleted = napster_server.deleteItemInServer(new_record);

    System.out.println("After Calling Server Delete");

    if(record_deleted)
        System.out.print("The Record Was Successfully Deleted on the
                    Server");
    else
        System.out.print("The Record Could Not Be Deleted on the Server");
        } // try
    catch(Exception e)
    {
    System.out.println("Error: " + e);
    e.printStackTrace(System.out);
    } // catch
}
```

Write a method to update a record:

```
public void updateRecord()
{
    try
        {
        System.out.print("Please Enter the Album Name: ");
        String album_name = stdin.readLine();

        System.out.print("Please Enter the Artist Name: ");
        String artist_name = stdin.readLine() ;

        // Create a record that you would like to be updated from the server
        // database
        Record update_record          =  new Record();
        update_record.album_name      =  album_name;
        update_record.artist_name     =  artist_name;

        // Got to give dummy values for the other values in the record otherwise
        // CORBA will complain
        update_record.owner_name    = new String(" ");
        update_record.version       = 1;

        System.out.print("Please Enter the New Owner Name: ");
        String new_owner = stdin.readLine() ;

    // Call the update function on the server
    boolean record_updated =
          napster_server.updateRecordInServer(update_record, new_owner);

        if(record_updated)
        {
            System.out.print("The Record Was Successfully Updated on the
                    Server");
```

```
            System.out.println("Album Name  : " + album_name);
            System.out.println("Artist Name   : " + artist_name);
            System.out.println("Owne          : " + new_owner);
            }
            else
            System.out.print("The Record Could Not Be Updated on the Server");
          }
          // try
          catch(Exception e)
          {
          System.out.println("Error: " + e);
          e.printStackTrace(System.out);
        } // catch
}
```

Write a method to display menu options:

```
public void displayMenu()
      {
      // Display the Menu
      System.out.println("\n ");
      System.out.println("Enter One of the Following Options");
      System.out.println("1. To Get an Album on the Server");
      System.out.println("2. To Add an Album to the Server");
      System.out.println("3. To Delete an Album on the Server");
      System.out.println("4. To Update an Album on the Server");
      System.out.println("5. To Exit");
      }
```

Write the client main method:

```
public static void main(String args[])
{
try
{
```

Initialize the ORB. The args array tells the ORB on the client machine where the name-server is running (machine name and port to use to connect to the nameserver):

```
ORB orb = ORB.init(args, null);
```

Get a reference to the NameServer:

```
org.omg.CORBA.Object objRef = orb.resolve_initial_references
                                        ("NameService");
```

Create a Java object from a CORBA object:

```
NamingContext ncRef   = NamingContextHelper.narrow(objRef);
```

Find the path on the NameServer where the server object is located:

```
NameComponent nc     = new NameComponent("NapsterServer", " ");
```

Set the path:

```
NameComponent path[]= {nc};
```

Get a server object and narrow the reference from a CORBA object to a java object (all in one call):

```
NapsterServerI napster_server =
NapsterServerIHelper.narrow(ncRef.resolve(path));
BufferedReader stdin =
new BufferedReader(new InputStreamReader(System.in));
```

Start the client:

```
NapsterClient napster_client = new NapsterClient(napster_server, stdin);

String choice_str;
int choice_int = 1;

// Keep looping till the user tells us to quit while(choice_int != 5)
{
      napster_client.displayMenu() ;
      choice_str = stdin.readLine() ;
      choice_int = Integer.parseInt(choice_str) ;

      switch(choice_int)
      {
         case 1:
         napster_client.getRecord();
            break;
         case 2:
         napster_client.addRecord() ;
            break;
         case 3:
         napster_client.deleteRecord();
            break;
         case 4:
         napster_client.updateRecord() ;
            break;
         case 5:
         choice_int = 5;
            break;
         default:
         System.out.println("Invalid Option. Please Try Again");
            break;
         } // switch
            } // while
         }
         catch(Exception e)
      {
         System.out.println("Error: " + e);
         e.printStackTrace(System.out);
         } // catch
      } // main
} // class
```

Running Napster

1. Install the idltojava compiler.

2. Install JDK1.2.1.

3. Compile the Napster.idl file: idltojava Napster.idl.

4. Compile the remaining files: Napster.java, NapsterServer.java, NapsterClient.java.

```
javac *.java Napster/*.java
```

5. Run the nameserver.

```
tnameserv -ORBInitialHost <IP address of machine running nameserver>
        -ORBInitialPort <port used: default is 900>
```

6. Run the NapsterServer.

```
java Napster -ORBInitialHost <IP address of machine running nameserver>
        -ORBInitialPort <port used: default is 900>
```

7. Run the NapsterClient.

```
java NapsterClient -ORBInitialHost <IP address of machine running
                        nameserver>
        -ORBInitialPort <port used: deafault is 900>
```

Discussion

It is evident that the server and clients both perform similar steps for initialization. Specifically, the server has to register its server object implementations in the nameserver so that the client knows where to find them in the nameserver. It is also evident that conversions from CORBA to Java objects must be performed because the Java interpreter cannot manipulate CORBA objects.

Conclusion

In this two-part series I have presented the development of a mini Napster example. This example is not particularly sophisticated, but it demonstrates how to write a Java client-server application using CORBA.

We can make the observation that once the CORBA initialization is complete then the rest of the development effort is a matter of pure Java. Indeed, this is mostly true in larger Java/CORBA applications. The main considerations when using advanced features of CORBA relate to the various CORBA services such as security, event, transaction, timing, and many others. Even so, the programming task is to gain access to a server object implementation providing the service.

The Napster example hopefully accomplishes at least the following:

- Demonstrates that CORBA is no more difficult to understand than Java
- Demonstrates that the CORBA and Java object model are very similar (modules match to packages and interfaces in CORBA and to interfaces in Java)

The true power of CORBA can best be understood by doing, and hopefully this article has provided readers with the incentive to perform further investigation into this powerful technology.