Description and concepts

CORBA

By: Hamid Mosavi-Porasl

1 Writing the IDL interface

1.1 Declaration of CORBA IDL Module

```
module HelloApp{ };
```

1. 2 Declaring the interface

```
module HelloApp{

   interface Hello{ };
};
```

1.3 Declaring the operations

```
module HelloApp{

interface Hello{

string sayHello();</SPAN};
};

};
```

1.4 Mapping IDL to Java

> idltojava hello.java

After execution of "idltojava" we have following 5 files:

1. Hello.java
2. HelloHelper.java
3. HelloHolder.java
4. _HelloImpBase.java
5. _HelloStub.java

IDL Statement	Java Statement
module HelloApp	package HelloApp;
interface Hello	public interface Hello
string sayHello();	String sayHello();

Mapping IDL statement to Java statement

2 Developing a client Application

2.1 Performing Basic Setups

- import required Java library packages

- declare application classes

- define a main method

2.1.1 Importing Required Packages

```
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
```

2.1.2 Declaring the Client Class

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

    //Put the try catch block here.

    }
```

2.1.3 Handling CORBA System Exceptions

Insert a try-catch block inside main:

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

```
try{
//Add the rest of the HelloClient code here
}catch(Exception e){
System.out.println("ERROR: "+ e);
e.printStackTrace(System.out);
        }
    }
```

2.1.4 Create an ORB Object

Declare and initialize an ORB variable inside the HelloClient.java's try-catch block

```
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 static void main(String args[]){
    try{
    ORB orb = ORB.init(args, null);
    }catch(Exception e){
    System.out.println("ERROR: "+ e);
    e.printStackTrace(System.out);
        }
    }
```

2.1.5 Finding the Hello Server

Once the application has an ORB, it can ask the ORB to locate the actual service it needs, in this case the Hello server.

2.1.6 Obtaining the initial Naming Context

Call orb.resolve_initial_references to get an object reference to the name server:

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*

try{

public static void main(String args[]){

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

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

}catch(Exception e){

System.out.println("ERROR: "+ e);

e.printStackTrace(System.out);}

}

2.1.7 Narrowing the Object Reference

As with all CORBA objectreferences, *objrerf* is a generic CORBA object. To use it as a NamingContext object you must narrow it to its proper type. Add the call to *narrow*:

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 static void main(String args[]){

try{

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

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

NamingContext ncref = NamingContextHelper.narrow(objRef);

}catch(Exception e){

System.out.println("ERROR: "+ e);

e.printStackTrace(System.out);

}

}

2.2 Finding a Service in Naming

Names can have different structures depending upon the implementation of the naming service. Consequenty, CORBA name servers handle complex names by way of *NameComponent* objects. Each *NameComponent* objec holds a *single port*, or *element*, of the name.

To find the Hello server, you first need a *NameComponent* to hold an identifying string for the Hello server. Add this code directly below the call to *narrow*:

NameComonent nc = new NameComponent("Hello","");

Because the path to the Hello object has just one element, create a single- element array of *nc*.

NameComponent path[] = {nc};

Finally, pass *path* to the naming service's *resolve* method to get an object reference to the Hello server and narrow it to a Hello object:

Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

Here you see the *HelloHelper* helper class at work. The *resolve* method returns a generic CORBA object as you saw above when locating the name service itself.

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 static void main(String args[]){

```
try{

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

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

 NamingContext ncref = NamingContextHelper.narrow(objRef);

NameComponent nc = new NameComponent("Hello","");

NameComponent path[] = {nc};

Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

  }catch(Exception e){

        System.out.println("ERROR: "+ e);

        e.printStackTrace(System.out);

    }

}
```

2.3 Invoking the sayHello Operation

CORBA invocations look like a method call on a local object. The complications of marshaling parameters to the wire, routing them to the server-side ORB, un-marshaling, and placing the up call to the server method are completely transparent to the client programmer.

1. Continuing with the try-catch block in *HelloClient.java*, enter the following invocation below the call to the name service's *resolve* method.

```
String Hello = HelloRef.sayHello();
```

2. Finally, add code to print the results of the invocation to standard output:

```
System.out.println(hello);

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 static void main(String args[]){

try{

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

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

NamingContext ncref =NamingContextHelper.narrow(objRef);

NameComponent nc = new NameComponent("Hello","");

NameComponent path[] = {nc};

Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

String hello = helloRef.sayHello();// Call the Hello server object

System.out.println(hello);// Print the result

    }catch(Exception e){

      System.out.println("ERROR: "+ e);

      e.printStackTrace(System.out);

      }

}
```

2.3.1 Setting Up the HTML File (Applet Only)

Tutorial.html is provided for displaying your finished applet, but you need to customize a few attributes and parameters.

1. Inside the APPLET tag, enter the path to your project directory as the value for the CODEBASE attribute.
2. In the first param tag, enter the name of the machine where the CORBA name server runs (must likely local machine name) as the value for ORBInitialHost.
3. Make sure the second PARAM tag is set to the value of ORBInitialPort that you are using to run the name server.

3 Developing the Hello Word Server

The structure of a CORBA server program is the same as most Java applications: You import required library packages, declare the server class, define a main method, and remember to handle any exceptions.

3.1 Importing Required Packages

Start your text editor. Import the packages required for the client class

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

3.2 Declaring the Server Class

```
// the package containing our stubs
import HelloApp.*;
// HelloServer will use the namin service
import org.omg.CosNaming.*;
// the package containing special exeception
import org.omg.CosNaming.NamingContextPackage.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;
public class HelloServer{
```

// Add the main method here in the next step

}

3.3 Defining the main Method

// the package containing our stubs

import HelloApp.;*

// HelloServer will use the namin service

import org.omg.CosNaming.;*

*// the package containing special exeception*s

import org.omg.CosNaming.NamingContextPackage.;*

// All CORBA applications need these classes.

import org.omg.CORBA.;*

public class HelloServer{

public static void main(String args[]){

// Add the main method here in the next step}}

3.4 Handling CORBA System Exceptions

Insert try-catch block:

// the package containing our stubs

import HelloApp.;*

// HelloServer will use the namin service

import org.omg.CosNaming.;*

*// the package containing special exeception*s

import org.omg.CosNaming.NamingContextPackage.;*

// All CORBA applications need these classes.

```
import org.omg.CORBA.*;
public class HelloServer{
public static void main(String args[]){
try{
// Add the rest of the HelloServer code here
}catch (Exception e){
System.err.println("ERROR: "+ e);
e.printStackTrace(System.out);
}
    }
  }
```

3.4.1 Create an ORB object

```
// the package containing our stubs
import HelloApp.*;
// HelloServer will use the namin service
import org.omg.CosNaming.*;
// the package containing special execeptions
import org.omg.CosNaming.NamingContextPackage.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;
public class HelloServer{
public static void main(String args[]){
try{
```

```
ORB orb = ORB.init(args,null);
// Add the rest of the HelloServer code here
}catch (Exception e){
System.err.println("ERROR: "+ e);
e.printStackTrace(System.out);
}
}
}
```

The call to the ORB's init method passes in the server's command line arguments, allowing insertion of certain properties at runtime.

3.4.2 Managing The Servant Object

A server is a process that initiates one or more servant objects

3.4.2.1 Initiating the Servant Object

Inside try-catch block just below the call to init, initiate the servant object:

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloServer{
public static void main(String args[]){
try{
ORB orb = ORB.init(args,null);
HelloServant helloRef = new HelloServant();
```

orb.connect(helloRef);

// Add the rest of the HelloServer code here

}catch (Exception e){

System.err.println("ERROR: "+ e);

e.printStackTrace(System.out);

}

}

}

3.4.3 Defining the Servant class

At the end of HelloServer.java, outside the HelloServer class, define the class for the servant objec:

1. Declare the servant class

class HelloServant extends _HelloImplBase{

// Add sayHello method here

}

1. The servant is a subclass of *_HelloImplBase* so that it inherits the general CORBA functionality generated for it by the compiler.
2. Declare the required *sayHello* method:

public String sayHello(){

// Add the method implementation

}

3. write the *sayHello* implementation

System.err.println("Hello world!");

class HelloServant extends _HelloImplBase{

```
    public String sayHello(){

    System.err.println("Hello world!");

    }

}
```

3.4.4 Narrowing the Object reference

As with all CORBA object refernces, objRef is generic CORBA objec. To use it as a *NamingContext* object, you must *narrow* it to its proper type.

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

Here you see the use of an *idltojava* -generated helper class, similar in function to *HelloHelper*. The ncRef object is now an *org.omg.CosNaming.NamingContext* and you can use it to access the name service and register the server.

```
import HelloApp.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

public class HelloServer{

public static void main(String args[]){

try{

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

HelloServant helloRef = new HelloServant();

orb.connect(helloRef);

NamingContext ncRef = NamingContextHelper.narrow(objRef);

// Add the rest of the HelloServer code here

}catch (Exception e){
```

System.err.println("ERROR: "+ e);

e.printStackTrace(System.out);

}

}

}

3.4.5 Registering the Servant with the Name Server

1. Create a new *NameComonent* member:

NameComponent nc = new NameComonent("Hello", "");

2. This statement sets the *id* field of *nc*, the new *NameComonent*, to "Hello" and the *kind* component to the empty string. Because the path to the *hello* has a single element, create the single element array that *NamingContex.resolve* requires for its work:

NameComponent path[] ={nc};

3. Finally, pass *path* and the servant object to the naming service, binding servant object to the "Hello" id:

ncRef.rebind(path, helloRef);

Now, when the client calls resolve("Hello") on the initial naming context, the naming service returns an object reference to the Hello servant.

import HelloApp.;*

import org.omg.CosNaming.;*

import org.omg.CosNaming.NamingContextPackage.;*

import org.omg.CORBA.;*

public class HelloServer{

public static void main(String args[]){

try{

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

HelloServant helloRef = new HelloServant();

orb.connect(helloRef);

NamingContext ncRef = NamingContextHelper.narrow(objRef);

NameComponent nc = new NameComonent("Hello", "");

NameComponent path[] ={nc};

ncRef.rebind(path, helloRef);

// Add the rest of the HelloServer code here

}catch (Exception e){

System.err.println("ERROR: "+ e);

e.printStackTrace(System.out);

                }

        }

  }
```

4 Waiting for Invocation

The server is ready; it simply needs to wait around for a client to request its service. To achieve that, enter the following code**:**

```
java.lang.Object sync = new java.lang.Object();

synchronized(sync){

sync.wait();

}

import HelloApp.*;

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

```
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloServer{
public static void main(String args[]){
try{
ORB orb = ORB.init(args,null);
HelloServant helloRef = new HelloServant();
orb.connect(helloRef);
NamingContext ncRef = NamingContextHelper.narrow(objRef);
NameComonent nc = new NameComonent("Hello", "");
NameComonent path[] ={nc};
ncRef.rebind(path, helloRef);
java.lang.Object sync = new java.lang.Object();
                synchronized(sync){
                sync.wait();
                }
        }catch (Exception e){
        System.err.println("ERROR: "+ e);
        e.printStackTrace(System.out);
        }
        }}
```

5 Compiling and Running the Hello World Application

5.1 Compile the Client & Server Applications

> *javac HelloClietn.Java HelloApp**

> *javac HelloServer.Java HelloApp**

5.2 Running the Client-Server Application

1. From command shell (MS-DOS in Windows), start the Java name server:

> *tnameserv -ORBInitialPort nameserverport*

****Note:* that *Nameserverport* is port on which you want the name server to run.

2. From a second prompt or shell, start the Hello server:

> java HelloServer - ORBInitialHost nameserverhost -ORBInitialPort nameserverport

****Note:* that *nameserverhost* is the host on which the IDL name server is running.

You can omit -*ORBInitialHost nameserverhost* if the name server is running on the same host as the Hello Serever. You can leave out -*ORBInitialPort nameseverport* if the name server is running on the default port.

3. From third prompt or shell, run the Hello application client:

> java HelloClient - ORBInitialHost nameserverhost -ORBInitialPort nameserverport

****Note:* that *nameserverhost* is the host on which the IDL name server is running.

You can omit -*ORBInitialHost nameserverhost* if the name server is running on the same host as the Hello Serever. You can leave out -*ORBInitialPort nameseverport* if the name server is running on the default port.

4. The client prints the string from the server to the command line:

Hello World !!