Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 7.

CORBA

1. OBJETIVO

El objetivo es mostrar ejemplos sencillos de utilización de la arquitectura CORBA mediante el uso de javaIDL.

2. DESCRIPCIÓN

Para describir los pasos para usar CORBA desde Java, se va a utilizar un ejemplo sencillo, una calculadora que tiene operaciones sencillas. La referencia a objeto remoto (IOR) se va a obtener en este ejemplo desde un fichero. En el segundo ejemplo, se usa un servidor que dice hello para ilustrar el uso del servicio de nombrado de CORBA para la obtención del IOR.

2.1 Obtención del IOR (Referencia a objeto) desde un fichero.

En esta sección, el servidor escribe en un fichero su referencia remota. El cliente lee la referencia remota desde este fichero para ponerse en contacto con el servidor.

2.1.1 Descripción de la interfaz y generación de soportes

En primer lugar, hay que realizar la especificación de la interfaz del servicio que es la siguiente:

```
module prueba
{
   interface Calculadora
   {
      double add(in double x, in double y);
      double substract(in double x, in double y);
      double multiply(in double x, in double y);
      double divide(in double x, in double y);
      void storeMemory(in double x);
      double readMemory();
   };
};
```

Calculadora.idl

Para generar los soportes del cliente (stub) y del servidor (skeleton) hay que ejecutar el compilador de IDL. Se usa la opción –fall para que genere tanto el soporte del cliente como del servidor:

- idlj –fall Calculadora.idl

Se generan de esta forma los ficheros necesarios en el subdirectorio prueba. Con ls prueba puede comprobar los ficheros que se han generado. Estos son:

CalculadoraHelper.java
CalculadoraHolder.java
Calculadora.java
CalculadoraOperations.java
CalculadoraPOA.java
_CalculadoraStub.java

Tanto la clase Helper como la clase Holder son clases auxiliares. Son clases que básicamente contienen métodos estáticos que pueden ser ejecutados sin necesidad de crear un objeto de esa clase. En la clase Helper existe un método llamado narrow y que se va a usar a menudo. Este método sirve para convertir una referencia a un objeto genérico CORBA a un objeto de un cierto tipo. Las referencias genéricas a objetos Java son convertidas en referencias de un cierto tipo habitualmente mediante la utilización de un molde o cast. Sin embargo, una referencia a un objeto genérico CORBA no se puede convertir a una referencia a objeto de un tipo concreto usando un molde, siendo necesario usar el método narrow.

Los ficheros Calculadora.java y CalculadoraOperations.java son interfaces con las operaciones del servicio.

El fichero CalculadoraPOA.java contiene la clase de la que va a derivar la clase de la implementación. Es el adaptador de objeto para el servidor. El adaptador de objetos es el que recibe la petición y la dirige al método adecuado de la implementación, realizando las operaciones oportunas (sería lo que hemos llamado hasta ahora soporte del servidor).

El fichero _CalculadoraStub.java contiene el soporte del cliente.

2.1.2 Implementación de la interfaz (servant)

Para implementar un objeto CORBA, es decir, para ofrecer sus servicios a los clientes, se tiene que realizar lo siguiente:

1. La implementación de los métodos de la interfaz que se ofrece al exterior, es decir, el sirviente (servant).
2. La implementación de un servidor que se quedará esperando las peticiones de los clientes.

El sirviente es un objeto Java que implementa la funcionalidad de los métodos del objeto CORBA. Este sirviente es llamado por el soporte del servidor cuando un cliente llama a un método del objeto CORBA implementado por ese sirviente. El código del sirviente implementa las funciones de la calculadora y es el siguiente:

```
package prueba;

import prueba.*;
```

```
class CalculadoraImpl extends CalculadoraPOA
{
  private double memoria_;
  public CalculadoraImpl()
  {
    memoria_ = 0;
  }
  public double add(double x, double y)
  {
    return x + y;
  }

  public double substract(double x, double y)
  {
    return x - y;
  }
  public double multiply(double x, double y)
  {
    return x * y;
  }

  public double divide(double x, double y)
  {
    double result = 0;

    try
    {
      result = x / y;
    } catch(Exception e)
    {
    }

    return result;
  }
  public void storeMemory(double x)
  {
    memoria_ = x;
  }

  public double readMemory()
  {
    return memoria_;
  }
}
```

CalculadoraImpl.java

Se puede observar que el sirviente hereda de la clase CalculadoraPOA. El POA (Adaptador de Objetos Portable) es la clase correspondiente al soporte del servidor, que se encarga de llamar al método adecuado cuando llega una petición de un cliente. De forma general los sirvientes tienen que heredar de la clase <interfaz>POA, donde interfaz es el nombre de la interfaz especificada en el fichero idl. En nuestro caso Calculadora.

2.1.3 Implementación del servidor

El servidor es el proceso que se queda esperando peticiones de los clientes sobre los objetos CORBA implementados en el sirviente. El servidor es un programa Java que deja activado un sirviente para el objeto CORBA. El código del servidor es el siguiente:

```
package prueba;

import org.omg.CORBA.*;
import org.omg.PortableServer.*;

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

      // Iniciar el ORB
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      // Objeto auxiliar
      org.omg.CORBA.Object o;

      // Encontrar el POA raiz
      POA rootPOA;
      o = orb.resolve_initial_references("RootPOA");
      rootPOA = POAHelper.narrow(o);
      // Activar el POA
      rootPOA.the_POAManager().activate();

      // Crear el objeto implementacion
      prueba.CalculadoraImpl calcImpl = new prueba.CalculadoraImpl();

      // Registrarlo en el POA
      o = rootPOA.servant_to_reference(calcImpl);

      prueba.Calculadora calc = prueba.CalculadoraHelper.narrow(o);

      // Producir la direccion del objeto
      String ior = orb.object_to_string(calc);

      // String ior = orb.object_to_string(o);
      System.out.println(ior);

      // Esperar llamadas
      orb.run();

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

Servidor.java

El servidor contiene casi toda la carga de la dificultad de la programación con CORBA. Las tareas que realiza son las siguientes:

- Iniciar el ORB. Normalmente el ORB se implementa como una función de librería. Esta llamada a init realiza todas las funciones de inicialización para el ORB.
- Encontrar y activar el POA raíz. Una vez iniciado el ORB, se usa éste para obtener la referencia del POA raíz mediante el método resolve_initial_references. En CORBA, los adaptadores de objetos se pueden configurar como una jerarquía, comenzando en el raíz. Una vez obtenido el POA raíz, se obtiene el POA manager que se encarga de controlar a un conjunto de adaptadores de objetos y se activa mediante el método activate. El método narrow se usa para convertir el objeto devuelto por el orb al tipo del POA raíz. Esto se puede ver como un cast de C.
- Crear el objeto implementación: Se crea el objeto sirviente del tipo CalculadoraImpl.
- Registrarlo en el POA: Mediante el método servant_to_reference se obtiene una referencia de objeto remoto a partir del sirviente y se registra en el POA raíz.
- Mediante el método narrow del helper se convierte la referencia o del tipo Org.omg.CORBA.Object a una referencia a prueba.Calculadora. Esto se puede ver de forma similar a un cast de C, pero algo más complejo. Este paso no es realmente necesario pero se muestra por completitud.
- Producir la dirección del objeto: Mediante el método object_to_string del orb se obtiene la referencia del objeto remoto calculadora en un string.
- A continuación se imprime el IOR como un string en la salida estándar.
- Esperar llamadas: Por último mediante el método run del orb el servidor se pone a esperar peticiones de los clientes.

2.1.4 Cliente

Por último queda implementar el cliente que use la referencia del objeto remoto para pedir sus servicios. En este caso, el cliente obtiene la referencia del objeto remoto usando el fichero que ha escrito el servidor con su referencia. El código del cliente es el siguiente:

```
package prueba;

import prueba.*;
import org.omg.CORBA.*;
import java.io.*;

public class Cliente
{
  public static void main(String args[])
  {
    try
    {
      // Iniciar el ORB
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

      // Leer el IOR del fichero
      File IORFile = new File("IOR");
      FileReader reader = new FileReader(IORFile);
      BufferedReader buf = new BufferedReader(reader);
      String IOR = buf.readLine();

      // Convertir el IOR en un objeto
      org.omg.CORBA.Object o = orb.string_to_object(IOR);
      Calculadora calc = CalculadoraHelper.narrow(o);

      // Usar la calculadora
```

```
      System.out.println(calc.add(2.0, 3.0));
    } catch(Exception e)
    {
        e.printStackTrace();
    }
  }
}
```

Cliente.java

El cliente realiza los siguientes pasos:

- Iniciar el ORB: Se inicia el ORB igual que en el servidor.
- Leer el IOR del fichero: Se lee el IOR del fichero llamado IOR.
- Convertir el IOR en un objeto: A partir del string que se ha leído del fichero, se obtiene la referencia al objeto usando el método string_to_object del orb.
- Usar la calculadora: Se llama al método add de la calculadora y el resultado se imprime por pantalla.

En el código se puede ver que incluye el paquete prueba. Este paquete contiene el soporte del cliente para el tipo de objeto Calculadora.

2.1.5 Compilación y ejecución del programa

Para la compilación de la interfaz IDL y la generación de los soportes del cliente y del servidor:

idlj –fall Calculadora.idl

Para la compilación de los ficheros java:

javac prueba/*.java

Para la ejecución del servidor:

java prueba.Servidor | tee IOR

se utiliza tee IOR para que lo que vaya a la salida estándar también vaya al fichero llamado IOR. Una vez que el servidor ha generado la referencia se puede ver por la salida estándar y también en este fichero usando la orden cat.

Para la ejecución del cliente:

java prueba.Cliente

⇒ 1. Descargue el fichero de código correspondiente a esta práctica. Compile y ejecute este ejemplo para ver el resultado.

2.2 Obtención del IOR mediante el servicio de nombrado de CORBA.

En el ejemplo anterior, el cliente obtiene la referencia del objeto CORBA de un fichero, previamente escrito por el servidor. En este segundo ejemplo, la referencia del objeto servidor se obtendrá del servicio de nombrado. El ejemplo que se utiliza es un objeto servidor que dice hola.

2.2.1 Interfaz del servidor

La interfaz del servidor es la siguiente:

```
module HelloApp
{
   interface Hello
   {
      string sayHello();
   };
};
```

Hello.idl

2.2.2 Servidor e implementación de la interfaz

El fichero HelloServer.java contiene tanto la implementación como el servidor:

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class HelloImpl extends HelloPOA {

  // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }

}

public class HelloServer {

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

      // get reference to rootpoa & activate the POAManager
      POA rootpoa =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // create servant and register it with the ORB
```

```
      HelloImpl helloImpl = new HelloImpl();

      // get object reference from the servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // get the root naming context
      // NameService invokes the name service
      org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the Interoperable
      // Naming Service (INS) specification.
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // bind the Object Reference in Naming
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);

      System.out.println("HelloServer ready and waiting ...");

      // wait for invocations from clients
      orb.run();
    }

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

  }
}
```

HelloServer.java

En este ejemplo se puede ver que una vez obtenida la referencia al objeto se enlaza en el servicio de nombrado (clase NamingContextExt) mediante el método rebind. La referencia al servicio de nombrado se obtiene mediante el método resolve_initial_reference del ORB.

2.2.3 Cliente

Y a continuación se muestra el cliente:

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  static Hello helloImpl;

  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");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " +
helloImpl);
        System.out.println(helloImpl.sayHello());

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

}
```

HelloClient.java

El cliente obtiene la referencia al objeto servidor usando el método resolve_str del servicio de nombrado.

2.2.4 Compilación y ejecución del programa

Para la compilación de la interfaz IDL y la generación de los soportes del cliente y del servidor:

idlj –fall Hello.idl

Para la compilación de los ficheros java:

javac *.java

El compilador se encarga de compilar también los ficheros necesarios del directorio HelloApp, ya que se usa este paquete.

En este caso es necesaria la ejecución del servicio de nombrado que se realiza mediante la ejecución del orb:

orbd –ORBInitialPort 1080

Para la ejecución del servidor:

java HelloServer –ORBInitialPort 1080

se indica el puerto del orb.

Para la ejecución del cliente:

java HelloClient –ORBInitialPort 1080 –ORBInitialHost localhost

se indica el puerto y la máquina donde se ejecuta el orb.

⇒ 2. Compile y ejecute este ejemplo para ver el resultado

⇒ 3. Implemente un servicio de Fecha. La interfaz debe ser la siguiente:

```
module TiempoApp
{
   interface Tiempo
   {
      struct tiempo {
         long hora;
         long minutos;
         long segundos;
      };

      long getHora();
      long getMinutos();
      long getSegundos();

      tiempo getTiempo();

   };
};
```

tiempo.idl

Para obtener la hora, los minutos y los segundos desde java puede usar la clase Calendar de la siguiente forma:

```
import java.util.Calendar;

Calendar ahora = Calendar.getInstance();

ahora.get(Calendar.HOUR_OF_DAY);
ahora.get(Calendar.MINUTE);
ahora.get(Calendar.SECOND);
```

⇒ 4. En la tarea habilitada para la práctica, entregue un fichero comprimido (.zip) con nombre su login que sea el directorio de nombre su login comprimido (que contiene el directorio p7codigo con todo su trabajo).