Interacción entre Aplicaciones:
objetos distribuidos e invocación remota (CORBA)

El modelo de programación distribuida con CORBA

Esta práctica muestra una aplicación que utiliza CORBA para invocar métodos entre objetos distribuidos. Se utiliza el ORB que se incluye a partir del JDK1.2 Se trata de observar las diferencias que le aparecen al programador de la aplicación respeto a Java-RMI. En la realidad, el código con CORBA puede extenderse bastante más a medida que se utilizan los servicios CORBA.

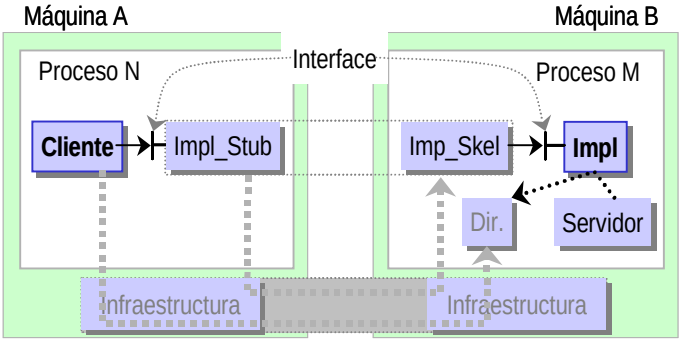

Fig. 1: Modelo general de invocación de métodos remota.

Cuando un objeto quiere invocar un método de un objeto remoto:

1) Ha de conocerlo (tener referencia) o localizarlo previamente (mediante un servicio adecuado). Algún proceso en la máquina remota habrá instanciado o al menos registrado un objeto que ofrece cierto servicio. Para ello se consigue un identificador del objeto servidor en forma de string (IOR: una estructura serializada y expresada textualmente: laaaarga), o se podía usar un servicio de directorio: el servicio de nombres o el servicio de "trading". En el caso siguiente, para simplificar (pero irreal), se obtiene de un fichero que contiene la referencia escrita por el proceso servidor.

```
      ORB orb = ORB.init(args, null);     // crear e inicializar ORB
... obtener la representación textual de un interface ...
      org.omg.CORBA.Object obj = orb.string_to_object(ior);
      X objX = XHelper.narrow(obj);
```

2) En realidad se obtiene una referencia o representante local (stub) del objeto remoto. Cuando se invoca un método del stub, éste pasa la petición a un stub servidor que invoca localmente al objeto que implementa el método. Este objeto (Impl) o bien estará esperando peticiones o será activado automáticamente al llegar una invocación (similar a un servlet) por un "Adaptador de Objetos" como el BOA (Básico) o más recientemente el POA (Portable). El resultado de la invocación sigue el mismo camino de vuelta.

De cara al objeto cliente, todo es bastante invisible: ha invocado un método de un objeto local (el stub), y ha obtenido una respuesta local. Lo que ha ocurrido entre stub cliente – stub servidor y objeto implementación es invisible y mágico para él.

En lugar de crear un objeto (local) `X objX = new X();` el objeto se ha creado al otro lado, y el cliente en su lugar ha de localizar y obtener una referencia a la instancia remota.

En la realidad, el proceso puede ser diferente dependiendo de qué servicios CORBA se utilicen para contactar con las instancias de objetos remotos o cómo se invoquen las operaciones. Por ejemplo, si se utiliza el interface de invocación dinámica (DII) se pueden invocar operaciones desconocidas para el cliente cuando se escribió y compiló el programa cliente. La forma de hacerlo es muy diferente al caso no dinámico: hay que rellenar una estructura de datos y pasarlos a un método de invocación remota.

Puede usarse también el servicio de nombres que permiten obtener una referencia de un objeto remoto a partir de nombres más "cómodos" y descriptivos que un IOR: iiploc://máquina/servicio , o servicios que permiten seleccionar entre varios objetos servidor en función de características de cada instancia: el servicio de "trading" y otros más.

Corba usa clases e interfaces del paquete: `org.omg.CORBA`. A continuación se especifica en java el interface de un programa ejemplo que se va a mostrar:

```
module mX {

  exception Unknown{};

  interface X {
     long incr(in long a);          // Incrementar un long ...
     string mesg(in string a) raises(Unknown); // Devolver mensaje
  };
};
```

El código del programa cliente:

```
import java.io.*;
import org.omg.CORBA.*;
import mX.*;

public class XCliente {
  public static void main(String args[]) {
    if (args.length!=2) {
      System.err.println("Faltan datos: fichero_con_ior mensaje");
      return;
    }
    try {
      ORB orb = ORB.init(args, null);     // crear e inicializar ORB

      // Leer ref. a objeto (ior) servidor creado por el servidor
      BufferedReader in =new BufferedReader(new FileReader(args[0]));
      String ior = in.readLine();
      in.close();

      // ior -> objeto
      org.omg.CORBA.Object obj = orb.string_to_object(ior);
      // objeto -> clase X
      X objX = XHelper.narrow(obj);

      // llamar los métodos del objeto remoto
      System.out.println("incr(2)=" + objX.incr(2));
      System.out.println("mesg(\""+ args[1] +"\")=\""
                  + objX.mesg(args[1]) + "\"");

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

El código de la clase que implementa el servicio:

```
import mX.*;

public class XImpl extends mX._XImplBase {

    private String _mensaje=null;

    public XImpl(String mensaje) {
        super();
        _mensaje = mensaje;
    }
    public int incr(int a) {
      return a+1;
    }
    public long incr(long a) {
      return a+1;
    }
    public String mesg(String m) throws Unknown {
      if (m==null) throw new Unknown();
      return (_mensaje + m + "!\n");
    }
}
```

El código de un programa servidor que instancia la clase anterior y "publica" la referencia, en forma de IOR en este caso:

```
import java.io.*;
import mX.*;

public class XServidor {
  public static void main(String[] args) {
    if (args.length!=2) {
      System.err.println("Faltan datos: fichero_ior mensaje");
      return;
    }
    try {
      // Initialize the ORB.
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

      // Crear objeto X.
      XImpl objX = new XImpl(args[1]);
      orb.connect(objX);

      // escribir referencia objeto en la pantalla
      System.out.println(orb.object_to_string(objX));
      // y en un fichero ...
      PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(args[0])));
      out.println(orb.object_to_string(objX));
      out.close();

      // esperar peticiones de los clientes
      java.lang.Object sync = new java.lang.Object();
      synchronized (sync) {
      sync.wait();
      }

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

En resumen, los pasos a seguir para diseñar una aplicación distribuida mínima con CORBA son:

1. Definir las funciones de la clase remota como un interface idl.
2. Escribir la clase implementación.
   a. Declarar que importa el módulo definido en el idl: `import mX.*;`
   b. Declarar que extiende la clase *_módulo*`ImplBase` definida automáticamente por el compilador de idl: `public class XImpl extends mX._XImplBase`.
   c. Proporcionar implementaciones para el constructor y métodos del interface.
3. Escribir la clase servidor que instancia y anuncia el objeto que implementa el interface.
   a. Crear una o más instancias del objeto remoto.
   b. Publicar o registrar esas instancias en algún servicio para que sea localizado o hacer llegar al cliente el IOR del objeto.
4. Escribir un programa cliente que contacte con el orb: `ORB orb = ORB.init(args, null);` obtenga una referencia a objeto servidor e invoque sus métodos.

Una vez comprobado que la JVM funciona (se puede ejecutar `javac y idlj`) se puede copiar el programa anterior y probar que funciona correctamente. Puede encontrarse en el web del laboratorio de AAD (http://www.ac.upc.es/docencia/FIB/AAD/lab)

Pasos a seguir:

1) Crear un directorio de trabajo. Traer y descompactar el paquete zip con el programa de ejemplo.

2) Compilar el código Java: `javac *.java`

3) Generar y compilar los stub cliente y servidor y otras clases de soporte del objeto implementación:
```
idlj -fall x.idl
cd mX
javac *.java
```

4) Tras ello, se ha de encontrar lo siguiente:

Origen	*Resultado*
`XImpl.java`	`XImpl.class`
`XServidor.java`	`XServidor.class`
`XCliente.java`	`XCliente.class`
`Idlj -fall` `x.idl`	`UnknownHelper.java`
Los resultados quedarán en el directorio `mX` (nombre del `module` idl)	`UnknownHolder.java`
	`Unknown.java`
	`_XImplBase.java`
	`XHolder.java`
	`XHelper.java`
	`XStub.java`
	`X.java`
	`XOperations.java`

5) Ejecutar el programa servidor, que instancia y anuncia un objeto de la clase XImpl:
```
java XServidor ior.txt "S>" &   (unix)
start java XServidor ior.txt "S>"   (windows)
```

6) Ejecutar el programa cliente, que localiza una instancia remota, e invoca sus métodos (en realidad los del objeto stub local):
```
java XCliente ior.txt "Hola desde cliente" &   (unix)
start java XCliente ior.txt "Hola desde cliente"   (windows)
```

Debería responder con el mensaje que se haya programado en la clase `XImpl`.