RMI (Remote Method Invocation)
o Invocación de Métodos Remotos)

1. Introducción

Al igual que con RPC (Remote Procedure Call o Llamada a Procedimientos Remotos) de C, es posible hacer que un programa en java llame a métodos de objetos que están instanciados en otro proceso y/o otra máquina de la red.

En base a lo expuesto, desarrollaremos un ejemplo básico de RMI para que sirva de modelo e introducirnos en cómo funciona. Esto conlleva a los siguientes pasos:

1. Se desarrollará una clase capaz de sumar dos enteros.
2. Haremos que esta clase quede registrada como objeto remoto, por lo que podrá ser llamada desde otros procesos.
3. Haremos un cliente capaz de llamar a esa clase para efectuar una suma mediante invocación a un método remoto.

Veamos qué clases necesitamos para hacer nuestros clientes y servidores de rmi. Debemos tener en cuenta:

- ✓ **InterfaceRemota**. Es una interface java con todos los métodos que queramos invocar de forma remota, es decir, los métodos que queremos llamar desde el cliente, pero que se ejecutarán en el servidor. Esta interface juega el rol de la interface lograda con el lenguaje XDR en RPC, pues es allí donde se especifica el llamado a la función remota, los argumentos que transfiere y el resultado que recibe.
- ✓ **ObjetoRemoto**. Es una clase con la implementación de todos los métodos de InterfaceRemota. A esta clase sólo la ve el servidor de rmi.
- ✓ **ObjetoRemoto_Stubs**. Es una clase que implementa InterfaceRemota. Cada método se encarga de hacer una llamada a través de la red al ObjetoRemoto del servidor, esperar el resultado y devolverlo. **Esta clase la ve el cliente y no necesitamos codificarla, java lo hace automáticamente a partir de ObjetoRemoto**.

Por último, necesitamos también un archivo de política de seguridad. En este archivo se indica al servidor de rmi y al cliente de rmi, qué conexiones pueden o no establecerse. **Debe haber un fichero de política de seguridad en el Servidor de rmi y otro en el Cliente.**

2. Implementación

En el servidor de rmi deben correr dos programas:

- **rmiregistry**. Este es el enlazador que nos proporciona java. Cumple un rol similar al que cumple PORTMAP en RPC. Una vez arrancado, admite que registremos en él objetos para que puedan ser invocados remotamente y admite peticiones de clientes para ejecutar métodos de estos objetos.
- **Servidor**. Este es el ejecutable que "instancia" objetoRemoto y lo registra en el rmiregistry. Una vez registrado objetoRemoto, el servidor queda activo. Cuando un cliente llame a un método de objetoRemoto, el código de ese método se ejecutará en el proceso del servidor.

En el cliente debe correr el programa:

- **Cliente**. Este programa pide al rmiregistry del servidor una referencia remota al Objeto Remoto. Una vez que la consigue (en realidad obtiene un ObjetoRemoto_Stbus), puede hacer las llamadas a sus métodos. Los métodos se ejecutarán en el Servidor y el Cliente quedará bloqueado hasta que Servidor termine de ejecutar el método.

3. Programación

a. Interface de la clase remota

Tenemos que hacer es una interface con los métodos que queremos que se puedan llamar remotamente. Esta interface es la siguiente:

```
import java.rmi.Remote;

public interface InterfaceRemota extends Remote
{
   public int suma (int a, int b) throws java.rmi.RemoteException;
}
```

Tiene que heredar de la interface **Remote** de java, si no el objeto no será remoto. Añade además los métodos que queramos, pero todos ellos deben lanzar la excepción **java.rmi.RemoteException**, que se producirá si hay algún problema con la comunicación entre los dos computadores o cualquier otro problema interno de rmi.

Todos los parámetros y valores devueltos de estos métodos deben ser tipos primitivos de java o bien clases que implementen la interface **Serializable** de java. De esta forma, tanto los parámetros como el resultado podrán transitar por la red, del cliente al servidor y viceversa.

b. El objeto remoto

Se debe hacer una clase que implemente **InterfaceRemota**, es decir, que tenga los métodos que queremos llamar desde un cliente **rmi**. El servidor de **rmi** se encargará de instanciar esta clase y de ponerla a disposición de los clientes. Esa clase es la que llamamos objeto remoto.

Esta clase remota debe implementar la interface remota que hemos definido (y por tanto implementará también la interface **Remote** de java). También debe hacer otras cosas como definir métodos como **hashCode()**, **toString()**, **equals()**, etc., de forma adecuada a un objeto remoto. También debe tener métodos que permita obtener referencias remotas. Es decir, una serie de elementos más o menos complejos y de las que no tenemos que preocuparnos si se hace que nuestra clase herede de **UnicastRemoteObject**. El código construido sería el siguiente:

```
import java.io.Serializable;

public class ObjetoRemoto extends UnicastRemoteObject implements InterfaceRemota
{
   public int suma(int a, int b)
   {
      System.out.println ("sumando " + a + " + " + b);
      return a+b;
   }
}
```

Otra opción es no hacerlo heredar de **UnicastRemoteObject**, pero luego la forma de

registrarlo varía un poco y además debemos encargarnos de implementar adecuadamente todos los métodos indicados (y algunos más).

c. La clase de stubs

Una vez compilado y que obtenemos nuestro archivo **ObjetoRemoto.class**, necesitamos crear la "clase de stubs", para que desde un proceso se pueda llamar a un método de una clase que reside en otro proceso. Se debe enviar un mensaje por red indicando que se quiere invocar un método de una clase y además pasar los parámetros de dicha invocación. Una vez ejecutado el método, el servidor que lo ha ejecutado debe enviar un mensaje con el resultado. La clase de stubs es una clase con los mismos métodos que nuestro **ObjetoRemoto**, pero en cada uno de esos métodos está inmersa la serialización del mensaje por la red tanto de la invocación como de la respuesta.

Java nos proporciona una herramienta llamada **rmic** a la que le pasamos la clase **ObjetoRemoto** y nos devuelve la clase de stubs **ObjetoRemoto_stubs**. Sólo tenemos que poner en la variable de entorno CLASSPATH el directorio en el que está nuestra clase **ObjetoRemoto** y ejecutar **rmic.**

```
$ set CLASSPATH=C:\java\rmisum
$ rmic ObjetoRemoto
```

Esto generarará un **ObjetoRemoto_stubs.class** y un **ObjetoRemoto_Skel.class**. El primero debe estar visible tanto por el cliente como por el servidor, es decir, deben aparecer en el **CLASSPATH** de ambos. Eso implica que debe estar situado en el servidor en un sitio público al que el cliente tenga acceso o que se debe suministrar una copia al cliente. El **ObjetoRemoto_Skel.class** se generará por defecto y sólo es útil para clientes con java anterior a la versión 1.2. Para java más moderno no tiene utilidad.

d. Servidor de rmi

Debe Instanciar una clase remota y luego registrarla en el servidor de rmi. Eso es sencillo.

```
ObjetoRemoto objetoRemoto = new ObjetoRemoto();
Naming.rebind ("ObjetoRemoto", objetoRemoto);
```

La instanciación no tiene problemas. Para registrarla hay que llamar al método estático **rebind()** de la clase **Naming**. Se le pasan dos parámetros. Un nombre para poder identificar el objeto y una instancia del objeto. El nombre que hemos dado debe conocerlo el cliente, para luego poder pedir la instancia por el nombre. El método **rebind()** registra el objeto. Si ya estuviera registrado, lo sustituye por el que acabamos de pasarle.

e. Cliente de RMI

Ahora tenemos que hacer el programa que utilice este objeto de forma remota. Los pasos que debe realizar este programa son los siguientes:

- Pedir el objeto remoto al servidor de rmi. El código para ello es:

  InterfaceRemota objetoRemoto = (InterfaceRemota) Naming.lookup

("//localhost/ObjetoRemoto");

Simplemente se llama al método estático **lookup()** de la clase **Naming**. Se le pasa a este método la URL del objeto. Esa URL es el nombre (o IP) del host donde está el servidor de rmi (por ejemplo **localhost**) y por último **el nombre** con el que se registró anteriormente el objeto (en este caso **ObjetoRemoto**).

- Este método devuelve un **Remote**, así que debemos hacer un "cast" a **InterfaceRemota** para poder utilizarlo. El objeto que recibimos aquí es realmente un **ObjetoRemoto_Stubs**.

- Ahora podemos llamar al método de suma().

```
System.out.print ("2 + 3 = ");
System.out.println (objetoRemoto.suma(2, 3));
```

Para que el código del cliente compile necesita ver en su classpath a **InterfaceRemota.class**. Para que además se ejecute sin problemas necesita además ver a **ObjetoRemoto_Stubs.class**, por lo que estas clases deben estar accesibles desde el servidor o bien tener copias locales de ellas.

d. El archivo de política de seguridad

Debemos crear, si no lo tenemos, un archivo de permisos. Este archivo de permisos le dirá al servidor de rmi qué conexiones debe o no aceptar y al cliente qué conexiones puede o no establecer.

El fichero de permisos por defecto debe llamarse **java.policy** y estar en el **HOME** del usuario que lanza el servidor o el cliente de rmi. Por ejemplo en Windows, en el **HOME C:\java**, se coloca **java.policy** con el siguiente contenido:

```
grant {
        permission java.security.AllPermission;
};
```

Con este contenido, se dan todos los permisos a todo el mundo. Si bien no es la opción más segura, es la adecuada en esta prueba. Es posible que se tenga problemas de permisos porque al ejecutar el programa no se encuentre este archivo. Una forma de obligar a que se tome el fichero que le indiquemos es cambiar la propiedad **"java.security.policy"** de la siguiente manera:

```
System.setProperty ("java.security.policy", "c:/java/java.policy");
```

Una propiedad no es más que un nombre que lleva asociado un valor. Asi, por ejemplo, la propiedad **"java.version"** dice cual es la versión de java en la que está corriendo el programa, **"os.name"** nos devuelve el nombre del sistema operativo, **"user.home"** nos devuelve en directorio por defecto del usuario, etc, etc. **System.getProperties()** nos devuelve una lista de todas las propiedades disponibles. De hecho, en la api de java, para este método, sale una lista con bastantes de las propiedades existentes. **System.setProperty()** fija el valor para una propiedad y **System.getProperty()** permite obtener el valor de una propiedad.

4. Ejecución

a. Rmiregistry

Antes de registrar el objeto remoto, debemos lanzar, desde una consola en Windows o linux el programa **rmiregistry**. Este programa viene con java y está en el directorio **bin** de donde tengamos instalado java y símplemente tenemos que arrancarlo sin parámetros. Es importante al arrancarlo que la variable **CLASSPATH** no tenga ningún valor que permita encontrar nuestros objetos remotos, por lo que se aconseja borrarla antes de lanzar **rmiregistry**. Si **rmiregistry** esta en el **PATH** de búsqueda de ejecutables (o nos hemos situado en el directorio en el que está), se lanzaría de la siguiente manera.

Desde Windows

```
c:\> set CLASSPATH=
c:\> rmiregistry
```

Desde linux

```
$ unset CLASSPATH
$ rmiregistry
```

Una vez arrancado **rmiregistry**, podemos registrar en él nuestros objetos remotos.

b. Servidor de rmi

Se ejecutará el servidor que instancie y registre el objeto remoto. Este programa java debe hacer lo siguiente

- Indicar cual es el path en el que se puede encontrar la clase correspondiente al objeto remoto. En nuestro caso, el path en el que se puede encontrar **ObjetoRemoto.class**. Dicho path se da en formato URL, por lo que no admite espacios ni caracteres extraños (Quizás admita los típicos %20% que se ven en el navegador cuando hay espacios en la url). En el ejemplo dejamos las clases **ObjetoRemoto.class**, **InterfaceRemota.class** y **ObjetoRemoto_Stubs.class** en **c:\java\remisum.** El path, en formato URL, puede ser algo como esto "**file://localhost/java/remisum**". En lugar de localhost puede ponerse la dirección IP de otro computador.

  El código para indicar esto es el siguiente:

  ```
  System.setProperty ("java.rmi.server.codebase", "file://localhost/java/remisum/");
  ```

  Consiste en fijar una propiedad de nombre **java.rmi.codebase** con el path donde se encuentran los ficheros **.class** remotos.

- Asegurarse que hay un gestor de seguridad. Para ello se comprueba si existe y si no hay, se añade. El código para ello es este

  ```
  if (System.getSecurityManager()==null)
  ```

```
System.setSecurityManager(new RMISecurityManager());
```

c. Cliente de RMI

Se ejecutará el cliente que invoque el método remoto. Como argumento de la línea de comandos se debe utilizar la URL del servidor o “localhost” si se trata de cliente y servidor en un mismo computador.

Luego de la ejecución el cliente devolverá la salida de la suma efectuada.