Capitulo 3

Remote Method Invocation: RMI

En este capitulo mencionamos los aspectos principales de RMI, capas y componentes, entre otras características.

3. Remote Method Invocation (RMI)

Los sistemas distribuidos requieren que las operaciones corran en diferentes espacios de direcciones, es decir sobre diferentes hosts. En un mecanismo de comunicación básico, el lenguaje java soporta sockets, los cuales son flexibles y suficientes para una comunicación general.

Una alternativa para los sockets es Remote Procedure Call (RPC) el cual extrae la interfaz de comunicación del nivel de una llamada a procedimiento. Sin embargo RPC, tiene algunas limitantes dentro de sistemas de objetos distribuidos.

El componente de Java, Remote Mehod Invocation (RMI, Invocación de métodos remotos en español), es un esquema para objetos distribuidos. Ahora ya es parte de la cubierta de java API. RMI ofrece algunos de los elementos críticos de un sistema de objetos distribuidos para java, además algunas otras características, que hacen posible que RMI sea un sistema únicamente java.

RMI tiene facilidades de comunicación de objetos que son similares a CORBA[11] y el mecanismo de serialización de objetos provee una forma para transferir o pedir una instancia de un objeto por valor desde un proceso remoto a otro.

Dado que RMI es un esquema de objetos distribuidos únicamente de Java, todas las interfaces de objetos son escrita en Java. Los stubs del cliente y los skeletons del servidor son generados a partir de esta interfaz

El mecanismo de RMI es básicamente un RPC orientado a objetos en el que existen tres procesos fundamentales:

- El cliente: proceso que invoca un método en un objeto remoto.
- El servidor: proceso que posee el objeto remoto
- Registro de objetos: se usa para el registro de objetos, para obtener acceso a objetos remotos utilizando su nombre.

3.1 Arquitectura.

El sistema RMI consta de tres capas:

- La capa de stub/skeleton.
- La capa de referencia remota.
- La capa de transporte.

La capa de aplicación esta situada en el nivel superior del sistema RMI. Una invocación remota de un cliente a un servidor de objetos remotos viaja a través de las capas del sistema RMI.

La relación entre estas capas se muestra en la figura 3.1.

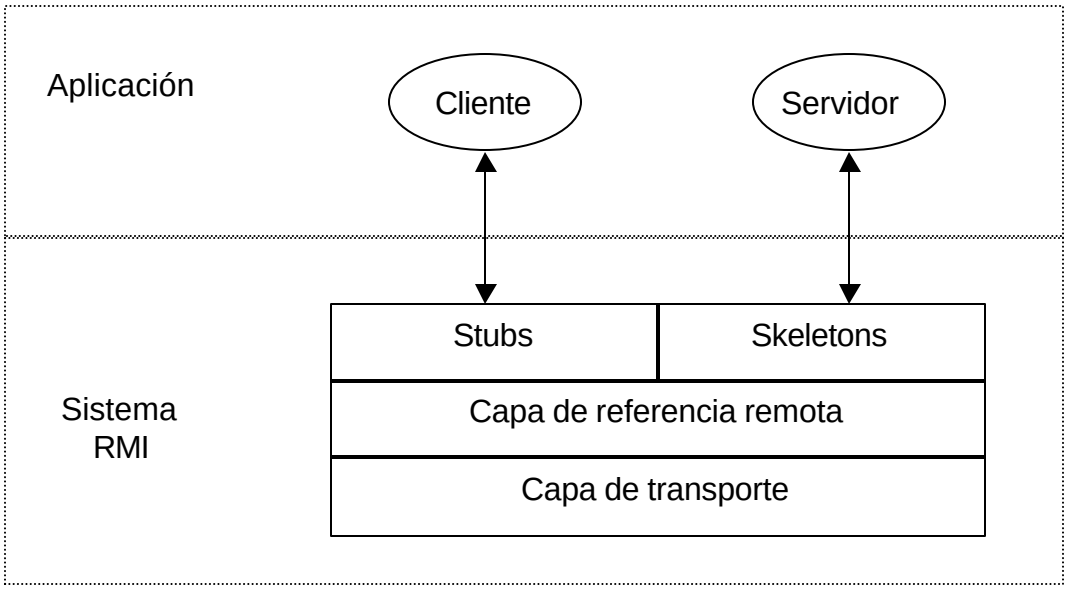

Fig. 3.1 Arquitectura del sistema RMI

3.1.1. La capa Stub/Skeleton

Esta capa es la interfaz entre la capa de aplicación y el resto del sistema RMI. Transmite datos desde la capa de referencia remota a través de un flujo ordenado. Este flujo emplea un mecanismo llamado serialización, el cual habilita los objetos java a ser transmitidos entre espacio de direcciones.

Un stub para un objeto remoto implementa todas las interfaces que son soportadas para la implementación de un objeto remoto[21]. Un stub del cliente es responsable de:

- Iniciar una llamada al objeto remoto
- Ordenar los argumentos del flujo
- Informar a la capa de referencia remota que la llamada debe ser hecha
- Retornar el valor o excepción de un flujo

- Informar a la capa de referencia remota que la llamada esta completa.

Un Skeleton para un objeto remoto en el servidor contiene un método que envía las llamadas a la actual implementación del objeto remoto. El skeleton es responsable de:

- Decodificar los datos del flujo de datos
- Hacer la llamada de la implementación del objeto remoto actual
- Retornar el estado de la llamada o excepción (si esta ocurre) dentro de un flujo.

3.1.2. La capa de referencia remota

La capa de *referencia remota* es responsable de llevar a cabo la especificación de un protocolo de referencia remota, el cual es independiente del stub del cliente y del skeleton del servidor.

La capa de referencia remota tiene dos componentes cooperativos: el lado del cliente y el lado del servidor. El componente del cliente contiene información especifica al servidor remoto y vías de comunicación para transportarse al lado del cliente. La capa de referencia remota transmite datos a la capa de transporte.

3.1.3. La capa de Transporte

En general la capa de transporte de RMI es responsable de:

- Establecer conexiones al espacio de direcciones

- Manejar las conexiones
- Monitorear las conexiones actuales
- Escuchar las llamadas
- Mantener una tabla de objetos remotos que residen en el espacio de direcciones
- Establecer una conexión para una llamada

3.2. Fases para crear un ejemplo usando RMI

Primero que nada debemos definir la interface remota, la cual tiene las siguientes características:

- La inteface remota debe ser pública
- La interface remota hereda de java.rmi.Remote.
- Un objeto remoto que pase un argumento debe ser declarada como interface remota no como implementación de la clase.

Enseguida definimos y escribimos la implementación de la clase remota, la cual necesita:

- Especificar la interface remota siendo implementada
- Definir el constructor para el objeto remoto
- Proveer la implementación de los métodos que pueden ser invocados remotamente
- Crear e instalar un administrador de seguridad
- Crear una o más instancias del objeto remoto

Posteriormente generamos un servidor y un cliente. En el servidor creamos y registramos los objetos remotos para que sus métodos puedan ser invocados remotamente, y en el cliente obtenemos las referencias a los objetos remotos que creamos previamente en el servidor.

Para lograr esto debemos generar el Stub y skeleton con una instrucción muy simple que usa únicamente la implementación de las clases. Esto es corriendo el compilador *rmic* con los nombres de la implementación de las clases.

Finalmente iniciamos el registro de RMI, el cual almacena los clientes remotos para hacer referencia a un objeto remoto. Para iniciar el registro sobre el servidor se utiliza el comando *rmiregistry {#puerto}* y posteriormente se corre el programa que funcionará como servidor.

Un ejemplo de una clase remota y su interfaz remota lo podemos ver en el *Apéndice B*, en donde también mostramos el programa cliente y el programa servidor.

3.3 Seguridad

La primera acción de un programa RMI es instalar el administrador de seguridad, con lo cual asegura que las clases cargadas cumplan con la características de seguridad en java. Una aplicación puede definir sus propias políticas de seguridad en vez de utilizar el administrador de RMI.

3.4. Conclusión

RMI es un sistema que nos permite el intercambio de objetos. Este intercambio se realiza de manera transparente de un espacio de dirección a otro, puesto que utiliza una técnica de serialización. Además que nos permite el llamado de los métodos remotamente, sin tener la necesidad de tener los métodos localmente.

Debido a que los sistemas necesitan manejo de datos en sistemas distribuidos cuando estos residen en direcciones de distintos Host, los métodos de invocación remota son una buena alternativa para solucionar estas necesidades. RMI es un sistema de programación para la distribución e intercambio de datos entre distintas aplicaciones existentes en un entorno distribuido.

Hemos mencionado, a lo largo de este capitulo las características más importantes de RMI. Por estas características elegimos a RMI como arquitectura o medio de compartir datos a través de la red. Sin embargo podemos mencionar otras características tales como la facilidad de usar y programar, ya que es un sistema cien por ciento java, además que permite realizar sistemas distribuidos orientados a objetos.

Como ya hemos mencionado, RMI se utiliza para realizar aplicaciones distribuidas, por lo que es de gran utilidad para el desarrollo de nuestra aplicación Mediante la invocación de métodos remotos podemos extraer datos geográficos de la base de datos a través de la red de una forma rápida y segura. Podemos tener métodos remotos que nos permitan extraer

datos para la actualización de nuestra base de datos de índices, y es su caso para la recuperación de los datos geográficos.

Tenemos otras alternativas por medio de las cuales podemos realizar el intercambio de datos Una alternativa y posiblemente muy similar a RMI en muchas características, es CORBA que es un estándar para la interoperación de objetos en ambientes cliente-servidor. Los objetos CORBA pueden residir en cualquier parte de la red, empaquetados como componentes binarios que los clientes remotos pueden accesar vía invocación de métodos, esto es totalmente transparente al cliente.