Ingeniería y Desarrollo
ISSN: 0122-3461
ingydes@uninorte.edu.co
Universidad del Norte
Colombia

Jabba Molinares, Daladier; Alies Fuentes, Maureen; Pinto Molina, Jhon; Buendía Rodríguez, Marirosa; Ceballos Salazar, Julio César

Conectividad de Java con bases de datos mediante invocación de objetos con métodos remotos (objetos RMI)

Ingeniería y Desarrollo, núm. 14, diciembre, 2003, pp. 93-106
Universidad del Norte
Barranquilla, Colombia

Conectividad de Java con bases de datos mediante invocación de objetos con métodos remotos (objetos RMI)

Daladier Jabba Molinares*, Maureen Alies Fuentes**,
Jhon Pinto Molina***, Marirosa Buendía Rodríguez****,
Julio César Ceballos Salazar*****

Resumen

El desarrollo de aplicaciones distribuidas está teniendo cada vez más auge entre las empresas del mundo; esto se debe a la gran importancia que ha adquirido el Internet en los últimos años. Para dar solución a esta necesidad surgen arquitecturas distribuidas como RMI (Invocación de Métodos Remotos). En este artículo se describe una de las principales arquitecturas empleadas para desarrollar aplicaciones basadas en objetos distribuidos, Java RMI. Además se presenta una comparación entre esta arquitectura y otras como CORBA, RPC y DCOM.

Palabras clave: Objetos distribuidos, sistemas distribuidos, Java RMI, sockets, CORBA (*Common Object Request Broker Architecture*), RPC (*Remote Procedure Call*), DCOM (*Distributed Component Object Model*), máquina virtual (MV).

Abstract

The Development of distributed applications is growing up among the companies in the world, it must for the big importance that Internet has become in the last years; for giving solutions to this necessity, architectures distributed appears as RMI (Remote Method Invocation). In this article we give a description about one of the main architectures used for developing applications based in distributed objects, Java RMI. Moreover, we presents a comparison among this architecture and others in the world, like: CORBA, RPC and DCOM.

Key words: Distributed objects, distributed systems, Java RMI, sockets, CORBA (*Common Object Request Broker Architecture*), RPC (*Remote Procedure Call*), DCOM (*Distributed Component Object Model*), virtual machine (VM).

Fecha de recepción: 1 de septiembre de 2003
Fecha de aceptación: 4 de noviembre de 2003

*Ingeniero de Sistemas, Universidad del Norte; Magíster en Ciencias Computacionales del convenio ITESM-CUTB. Docente del Departamento de Sistemas, Universidad del Norte *djabba@uninorte.edu.co*

**Ingeniero de Sistemas, Universidad del Norte, 2001. *maureen_alies@yahoo.com*

***Ingeniero de Sistemas, Universidad del Norte, 2001. *jhonpinto@tutopia.com*

****Ingeniero de Sistemas, Universidad del Norte, 2002. *mbuendia@unimail.uninorte.edu.co*

*****Ingeniero de Sistemas, Universidad del Norte, 2002. *jceballos@unimail.uninorte.edu.co*

1. INTRODUCCIÓN

En la actualidad, el cómputo distribuido ocupa un lugar preponderante tanto en las ciencias de la computación como en la industria, debido a que muchos de los problemas que éstas enfrentan son inherentemente distribuidos. De la misma manera, las tecnologías orientadas a objetos se han consolidado como una de las herramientas más eficaces en el desarrollo de *software*, debido principalmente a su capacidad de describir los problemas en el dominio del problema más que en el dominio de la solución. Dentro del ámbito del cómputo distribuido se incorpora fuertemente la tecnología orientada a objetos, debido a que en el paradigma basado en objetos el estado de un programa ya se encuentra distribuido de manera lógica en diferentes objetos, lo que hace a la distribución física de estos objetos en diferentes procesos o computadoras una extensión natural.

La invocación remota de métodos de Java (*Remote Method Invocation*, RMI) es un modelo de objetos distribuidos, diseñado específicamente para el lenguaje Java, por lo que mantiene la semántica del modelo de objetos locales de Java, facilitando de esta manera la implantación y el uso de objetos distribuidos.

Por otra parte, ODBC, una interfaz basada en C y aplicada a motores de bases de datos basados en SQL, provee una interfaz para comunicarse con una base de datos y para acceder a los metadatos (información sobre la base de datos, cómo es guardada la información, etc.) de una base de datos. Cada vendedor pone a disposición del usuario controladores específicos para su sistema manejador de base de datos. Gracias a ODBC y SQL es posible conectarse a una base de datos y manipularla de una forma estándar. Java posee un conjunto de librerías, entre las cuales se encuentra JDBC, la cual puede ser vista como la versión para JAVA de ODBC; en la actualidad existe un controlador que funciona como interfaz entre JAVA y ODBC, lo cual permite la comu-nicación con bases de datos que no tienen conocimiento de la existencia de JAVA.

Para permitir la invocación de Métodos Remotos que accedan y ejecuten acciones a una base de datos es necesaria la utilización de otra herramienta denominada RMI, la cual mencionamos con anterioridad; ésta permite que un objeto que se ejecuta bajo el control de una JVM (*Java Virtual Machine*) pueda invocar métodos de un objeto que se encuentra en ejecución bajo el control de una JVM diferente.

2. REMOTE METHOD INVOCATION

¿Qué es un Objeto Remoto?

En el modelo de objetos distribuidos de Java, un objeto remoto es aquel cuyos métodos pueden ser invocados por objetos que se encuentran en una máquina virtual (MV) diferente. Los objetos de este tipo se describen por una o más interfaces remotas que contienen la definición de los métodos del objeto que es posible invocar remotamente.

¿Qué es RMI (*Remote Method Invocation*)?

RMI es un término que se usa para describir llamadas a métodos de objetos que por lo general no se encuentran localizados en la misma computadora; soporta no sólo la transferencia de control entre dos computadoras, sino también la transferencia de objetos tanto por paso por referencia como por valor. Utiliza conceptos de inter comunicación entre procesos de muy alto nivel, lo cual la hace una alternativa atractiva a la comunicación entre procesos usando *sockets*.

Las aplicaciones que usan RMI por lo general están compuestas por dos programas separados: Un servidor y un cliente. Una aplicación servidor típica crea los objetos remotos, hace accesible referencias hacia ellos y espera a que los clientes invoquen métodos de estos objetos remotos. Una aplicación cliente típica obtiene una referencia remota a uno o más objetos remotos en el servidor y luego invoca métodos de estos objetos.

RMI provee el mecanismo por medio del cual el servidor y el cliente se comunican y se pasan información del uno al otro. Estas aplicaciones son conocidas como Aplicaciones de Objetos Distribuidos, las cuales necesitan:

- *Localizar objetos remotos*. Las aplicaciones pueden usar uno de dos mecanismos para obtener referencias a objetos remotos. Una aplicación puede registrar en el rmizregistry sus objetos remotos utilizando el método Naming de RMI, o la aplicación puede pasar y devolver referencias a objetos remotos como parte de su operación normal.
- *Comunicarse con los Objetos Remotos*. Los detalles de la comunicación entre los objetos remotos son manejados por RMI. Para el programador, la comunicación remota es como la invocación estándar de métodos en Java.
- *Cargar el código de las clases de los objetos que son transmitidos*. Debido a que RMI permite que un invocador pase objetos a los objetos remotos, provee los mecanismos necesarios para cargar el código de los objetos, así como también transmitir sus datos.

La figura 1 describe una aplicación distribuida RMI que usa el registro para obtener una referencia a un objeto remoto. El servidor llama al registro para asociar un nombre con un objeto remoto. El cliente busca el objeto remoto por su nombre en el registro del servidor y luego invoca un método del objeto remoto.

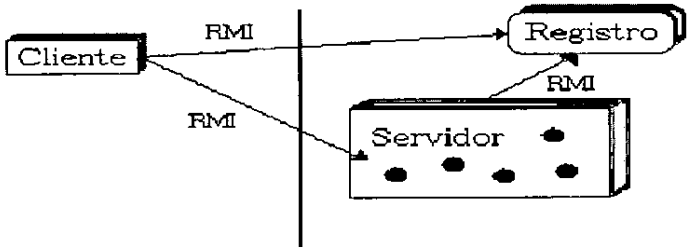

Figura 1. Aplicación distribuida RMI

2.1. *Arquitectura RMI de Java*

RMI está construida por tres capas abstractas:

- La capa stub/skeleton (Stub & Skeleton)
- La capa de referencia remota
- La capa de transporte

Cada capa es independiente de las otras y tiene definido su propia interfaz y protocolo. Al utilizar una arquitectura con capas, cada una de ellas podría ser mejorada o reemplazada sin afectar al resto del sistema. Por ejemplo, la capa de transporte podría ser reemplazada por una capa UDP/IP sin afectar las capas superiores (ver figura 2).

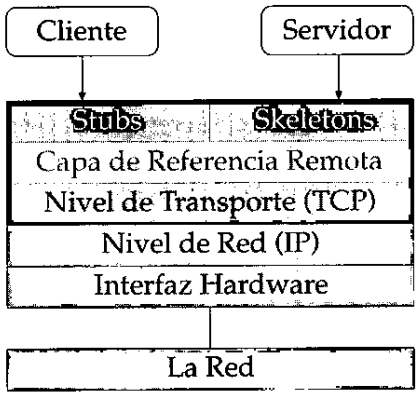

Figura 2. Arquitectura RMI

2.1.1. *La capa Stub/Skeleton (Stub & Skeleton)*

Se encuentra ubicada debajo de la vista del desarrollador. Esta capa intercepta las llamadas a métodos hechas por el cliente a la variable de referencia de la interfaz y redirecciona estas llamadas a un servicio RMI remoto.

El punto de contacto de la aplicación cliente con el objeto remoto se hace por medio del *stub* (delegado) local. Los *stubs* actúan como mediadores en la comunicación y son los responsables de traducir los objetos a una representación apropiada para entonces realizar la llamada al método remoto. Para todos los efectos, el *stub* es la representación local del objeto remoto. Entre sus responsabilidades se destacan:

- Inicializar las llamadas a los objetos remotos
- Serializar los argumentos para enviarlos por la red
- Deserializar los argumentos devueltos en las llamadas

Del lado del servidor, el *skeleton* (esqueleto) es el equivalente al *stub* en el cliente. Se encarga de traducir las invocaciones que provienen de la capa de referencia remota, así como de gestionar las respuestas. Entre sus actividades se destacan:

- Deserializar los argumentos
- Hacer las llamadas a los métodos de la implantación del objeto remoto
- Serializar los valores de retorno

2.1.2. *La capa de referencia remota*

La capa de referencia remota descansa debajo de la capa Stub/Skeleton. Está formada por dos entidades distintas, el cliente y el servidor, que se comunican a través de la capa de transporte. Es responsable de implementar la política de comu-nicación, que puede ser de distintos tipos:

- Invocación unicast punto-punto
- Invocación a grupos de objetos
- Estrategias de reconexión

Esta capa entiende cómo interpretar y manejar las referencias hechas por los clientes a los servicios remotos. En JDK 1.1, esta capa conecta clientes con servicios remotos que se están ejecutando y están exportados en un servidor. La conexión es uno a uno. En Java 2 SDK, esta capa se mejoró para soportar la activación de objetos remotos inactivos vía Activación de Objetos Remotos.

2.1.3. *La capa de transporte*

La capa de transporte es responsable del establecimiento y mantenimiento de la conexión, la cual proporciona una canal de comunicación fiable entre las capas del referencia remota del cliente y del servidor. Sus principales responsabilidades son:

- Establecimiento y mantenimiento de la conexión
- Atender a llamadas entrantes
- Establecer la comunicación para las llamadas entrantes

Se basa en conexiones TCP/IP entre máquinas de una red. Provee conectividad básica, así como también estrategias de penetración (*firewall*). La capa de transporte de RMI , en la realidad, está implementada por medio de *sockets*.

2.2. Características de RMI

Dentro de las características de los objetos remotos encontramos las siguientes:

- Sencillez
- Transparencia
- Paso de objetos por valor (como parámetros de los métodos)
- Implementación 100% JAVA
- Independencia del protocolo de comunicación
- Permite la comunicación entre objetos situados (creados y ejecutados) en máquinas diferentes
- Cada objeto remoto implementa un interfaz remota que especifica cuales métodos pueden ser invocados por los clientes
- Los clientes pueden invocar métodos remotos casi exactamente igual que se invocan métodos locales
- La invocación remota de un método (RMI) es la acción de invocar un método de una interfaz remota de un objeto remoto. La invocación de un método de un objeto remoto tiene exactamente la misma sintaxis de invocación que la de un objeto local.

2.3. **Metas del sistema RMI de Java**

Las metas que se pretenden alcanzar al soportar objetos distribuidos en Java son:

- Proporcionar invocación remota de objetos que se encuentran en JVM diferentes
- Soportar llamadas a los servidores desde los *applets*
- Integrar el modelo de objetos distribuidos en el lenguaje Java de una manera natural, conservando, en la medida de lo posible, la semántica de los objetos Java.

- Hacer tan simple como sea posible la escritura de aplicaciones distribuidas
- Preservar la seguridad proporcionada por el ambiente Java
- Proporcionar varias semánticas para las referencias de los objetos remotos

2.4. **Ventajas de** RMI

Trabajar con objetos remotos mediante RMI ofrece las siguientes ventajas:

- Hace parte del estándar del lenguaje Java
- Aprovecha las ventajas del lenguaje Java
- Los detalles de comunicación son transparentes para el programador
- Permite el desarrollo rápido y fácil de objetos distribuidos
- Es una plataforma amigable para empezar en el área de aplicaciones distribuidas
- La habilidad de descargar los *bytecodes* (o simplemente, código) de una clase de un objeto si la clase no está definida en la máquina virtual del servidor

2.5. **Desventajas de** RMI

- No permite la fácil integración con sistemas heredados
- No es rápido
- Tiene algunas limitaciones debido a su estrecha integración con Java; la principal de ellas es que esta tecnología no permite la interacción con aplicaciones escritas en otro lenguaje.

2.6. **Métodos comparables y/o equivalentes con** RMI

La tendencia actual es el desarrollo de aplicaciones distribuidas a través de entornos Web; es por esto que se encuentran comúnmente métodos para implementar este tipo de arquitecturas que permitan la comunicación en un ambiente distribuido de varios equipos y aplicaciones independientemente de la localización de éstos. RMI es uno de los tantos métodos que se utilizan en la actualidad para implementar este tipo de aplicaciones a través de objetos remotos.

Entre los métodos más ejemplares y conocidos comparables a RMI se encuentran:

- Sockets
- RPC (*Remote Procedure Call*)
- DCOM (*Distributed Component Object Model*)
- CORBA (*Common Object Request Broker Architecture*)

Por sus destacables cualidades y ventajas cabe destacar las características de DCOM y CORBA.

2.6.1. DCOM *(Distributed Component Object Model)*

Está basado en el modelo de componentes COM (*Object Component Model*) de Microsoft. Soporta la interoperabilidad de componentes construidos con diversas herramientas Microsoft. Utiliza el protocolo *Object Remote procesadure Call* (ORPC), construido sobre mecanismos DCE RPC (*Distribuited Computing Enviroment-Remote Procedure Call*).

A diferencia de RMI, DCOM está restringido a plataformas Microsoft, aunque hay proyecciones de llevar DCOM a otras plataformas. Otra característica es que puede contar con varias interfaces. Cada interfaz puede contener unos métodos y propiedades que no tienen por qué ser igual en todas las interfaces. Además, los componentes de DCOM pueden ser escritos en diversos lenguajes de programación: C++, Java, Object Pascal (Delphi), Visual Basic e incluso COBOL. Quizás el mayor inconveniente de DCOM es que está íntimamente ligado a Windows. Además, el modelo Orientado a Objeto de DCOM es menos flexible que el de CORBA o los componentes RMI de Java.

2.6.2. CORBA *(Common Object Request Broker Architecture)*

CORBA define una especificación para un ambiente de computación orientado a objetos, heterogéneo y distribuido. La especificación incluye un lenguaje de definición de interfaces (*Interface Definition Language*, IDL) para describir las interfaces que las implementaciones de CORBA deben implementar. CORBA no provee todos los servicios sino solamente aquellos que razonablemente se puede esperar que se implementen en cualquier lenguaje y plataforma.

CORBA soporta la activación automática de objetos, la cual no se permite en Java (es necesario crear una instancia de la clase antes de invocar uno de sus métodos).

2.6.3. *Cuadro comparativo entre CORBA, DCOM y Java RMI*

Algunas de las diferencias entre CORBA, DCOM y Java RMI podemos apreciarlas en la siguiente tabla:

Característica	CORBA	DCOM	Java RMI
- Protocolo utilizado en la ivocación de métodos remotos	Internet Inter - ORB protocol (IIOP)	object Remote Procedure Call (ORCP)	Java Remote Method Protocol (JRMP)
- Plataforma	Cualquiera	Windows	Cualquiera
- Lenguaje soportado	Cualquiera	Cualquiera bajo Microsoft	Java

- Pueden especificar excepciones en el Interface Definition Language (IDL)	Si	No	Sí
- Utiliza un archivo .java para definir la interface remota	No	No	Sí
- Soporta múltiple herencia en el nivel de la interface	Si	No	Sí
- Responsabilidad de localizar la implementación de un objeto	Object Adapter	Service Control Manager	Java Virtual Machine
- Parámetros pasados entre el cliente y el servidor se definen	Si son tipo Interface se pasan por referencia, el resto por valor	En el IDL	Los objetos remotos implementados que extienden a java.rmi.Remote se pasan por referencia remota, el resto por valor

2.7. **Pasos que se deben seguir para desarrollar una aplicación RMI**

- *Definir una interfaz remota*: El servidor remoto de objetos debe declarar sus servicios por medio de una interfaz, extendiendo la interfaz java.rmi.Remote. Cada método de la interfaz remota debe lanzar una excepción: java.rmi.RemoteException.

- *Implementar la interfaz remota*: El servidor remoto debe implementar la interfaz, derivando la clase de java.rmi.UnicastRemoteObject.

Las reglas generales de una clase que implementa una interfaz remota son las siguientes:

- *Compilar las clases servidoras*: El servidor debe compilarse usando *javac* (i.e. javac server.java).

- *Correr el generador de stubs y skeletons*: El generador (o compilador) de *stubs* que acompaña a RMI es *rmic* (Ej. rmic server.class). Este generador se aplica al código compilado (.class) para generar los delegados de los clientes y los *esqueletos* (*skeletons*) de los servidores. El compilador rmic toma los mismos parámetros de la línea de comandos que toma javac.

- *Comenzar el registro RMI sobre el servidor*: RMI define interfaces para un servicio de nombramiento no persistente llamado el registro (*Registry*). RMI tiene una implementación de este objeto remoto que permite recuperar y registrar servidores usando nombres simples. Cada servidor puede soportar su propio registro o tener un solo registro independiente que admita a todas las maquinas virtuales disponibles en la computadora del servidor. Para arrancar un objeto registro en el servidor se lanza el comando RMI (ejemplo, *RMI registry*). Puesto que la bases de datos del registro está vacía cuando el servidor comienza, todos los objetos remotos que se construyan se deben insertar.

- *Iniciar los objetos servidores*: Para comenzar la interacción, se deben cargar todas las clases de los servidores, y entonces crear las instancias de los objetos remotos.

- *Registrar el objeto remoto con el registro*: Todas las instancias de los objetos se deben registrar ante el registro RMI de modo que puedan ser conocidos por los clientes. Para lograrlo se deben usar los métodos de la clase java.rmi.Naming, la cual asocia un nombre al servidor. Esta clase es la infraestructura de registro RMI para almacenar los nombres. Los servidores, una vez registrados, ya pueden ser conocidos (e invocados) por los clientes.

- *Escribir el código del cliente*: El cliente debe usar la clase java.rmi.Naming para localizar al objeto remoto. Entonces, el cliente puede invocar los servicios de los objetos remotos entablando comunicación a través del delegado (*stub*) que actúa como un representante del servidor ante el cliente.

- *Compilar el código del cliente*: El cliente debe compilarse usando javac (i.e. javac client.java).

- *Iniciar la ejecución del cliente*: Antes de comenzar la ejecución se deben cargar las clases del cliente, así como los delegados de los servidores. RMI también ofrece mecanismos de seguridad para descargar por demanda a diferentes representantes provenientes del servidor.

3. JAVA DATABASE CONNECTIVITY

JDBC es un API de Java que permite al programador ejecutar instrucciones en lenguaje estándar de acceso a bases de datos, SQL. Para que una aplicación pueda hacer operaciones en una base de datos, debe tener una conexión con ella, que se establece a través de un *driver* que convierte el lenguaje de alto nivel a sentencias de bases de datos. Por lo tanto, las tres acciones principales que realiza JDBC son:

- Establecer la conexión a una base de datos
- Enviar sentencias SQL a la base de datos
- Procesar los resultados obtenidos de la base de datos

3.1. **Conectividad JDBC**

JDBC está diseñado teniendo en mente la comunicación con bases de datos; es por esto que especifica una serie de clases y métodos para que cualquier programa desarrollado en Java tenga acceso a sistemas de bases de datos de forma homogénea. Este acceso se realiza a través de *drivers*, que son los que implementan la funcionalidad especificada en JDBC.

A pesar de la existencia de ODBC es necesario JDBC, debido a que ODBC es una interfaz escrita en lenguaje C, que al no ser un lenguaje portable haría que las aplicaciones desarrolladas en Java pierdan la portabilidad.

JDBC permite escribir aplicaciones que accedan a datos a través de sistemas de bases de datos incompatibles, corriendo en plataformas distintas, basándose en que Java se puede ejecutar sobre plataformas *hardware* y sistemas operativos diferentes.

3.2. **Puente JDBC -- ODBC**

Este *driver* proporciona acceso a bases de datos desde JDBC a través de uno o más *drivers* ODBC, aprovechando así la configuración ya hecha en el ODBC. Es muy útil cuando ya existen *drivers* ODBC instalados en la máquina en la cual se ejecuta la aplicación. Sin embargo resulta inadecuado cuando se trata de aplicaciones que requieren una alta velocidad de respuesta, ya que el rendimiento se reciente enormemente al tener que realizarse la conversión de transacciones de JDBC a ODBC. Además este *driver* no soporta todas las características de Java.

El *driver* ODBC se carga de forma local, lo cual impide el acceso a través de una red. Para utilizar los *drivers* JDBC en un entorno de red hay que recurrir a RMI, que replica una conexión local en una base de datos remota, resintiendo aún más el rendimiento de la aplicación.

3.3. **Similitudes y diferencias entre el modelo de objetos locales y distribuidos de Java**

Como se mencionó anteriormente, el modelo de objetos distribuidos de Java se desarrolló teniendo como meta acercarlo lo más posible al modelo de objetos locales de Java. Como es de esperar, un objeto remoto no puede ser exactamente igual a uno local, pero es similar en dos aspectos muy importantes:

- Se puede pasar una referencia a un objeto remoto, como argumento o como valor de retorno en la invocación de cualquier método, ya sea local o remoto.

- Se puede forzar una conversión de tipos de un objeto remoto a cualquier interfaz remota, mediante la sintaxis normal de Java que existe para este propósito.

Sin embargo, el modelo de objetos distribuidos difiere con el modelo de objetos locales en los siguientes aspectos:

- Los clientes de los objetos remotos interactúan con las interfaces remotas y nunca con las clases que implementan dichas interfaces.

- Los argumentos de los métodos remotos, así como los valores de retorno, son pasados por copia y no por referencia.

- Los objetos remotos se pasan por referencia y no mediante la copia de la implantación del objeto.

- La semántica de algunos métodos definidos por la clase java.lang.Object está especializada para el caso de los objetos remotos.

- Los clientes deben tener en cuenta excepciones adicionales referentes a la invocación remota de los métodos.

4. UTILIZACION DE RMI Y JDBC COMBINADOS

- Protege la base de datos de posibles errores, centralizando en la aplicación el acceso a la base de datos vía SQL (ver figura 3).

- Mayor eficiencia (mejor tiempo de respuesta), pues sólo se hacen pedidos SQL a nivel local.

- Simplifica el desarrollo de las aplicaciones que utilizan sólo los métodos permitidos sobre los objetos de información sin conocer la estructura de las BD que los implantan (sigue mejor el paradigma OO).

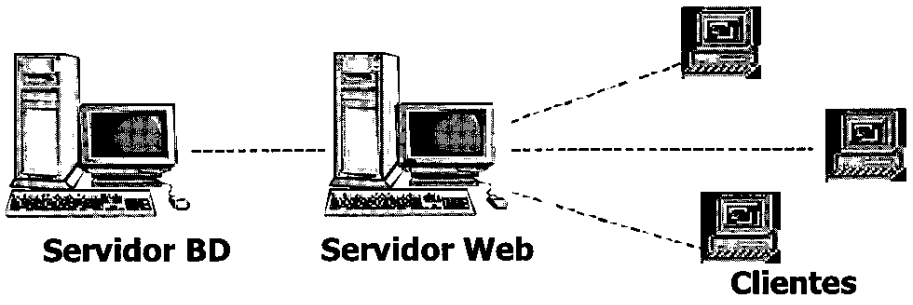

Figura 3. RMI y JDBC

CONCLUSIONES

Es bastante claro que el futuro de las aplicaciones informáticas está orientada a implementar su desarrollo bajo arquitecturas distribuidas dirigidas bien sea para Internet o Intranet. RMI de JavaSoft es una de las soluciones de que se dispone actualmente para desarrollar este tipo de aplicaciones. RMI posee todas las características de seguridad que hereda de la plataforma Java misma, pero no posee una arquitectura que le proporcione independencia del lenguaje de programación. Esto quiere decir que a diferencia de CORBA, que posee una arquitectura que proporciona independencia del lenguaje de programación, RMI está diseñada exclusivamente para Java, y esto puede ser considerado una gran desventaja.

Java es el lenguaje que se debe utilizar en el lado del cliente (indirectamente) y del servidor en la Web. Por lo tanto, es importante evaluar cómo cada medio se integra con Java. Pero uno de los grandes problemas de Java es que los objetos deben ser capaces de comunicarse con todos los objetos de la red, también con los escritos en C++ y con los objetos *Smalltalk* (herencia de los sistemas COBOL). Así que tendremos que evaluar la capacidad de integración de RMI con otros lenguajes y sistemas operativos.

La invocación remota de métodos en Java parte del hecho de correr sobre cualquier plataforma. Está diseñada para tomar ventaja de esta característica, lo que le permite presentar propiedades que otros modelos de objetos no poseen. Ejemplo de esto lo tenemos en la capacidad que tiene RMI de migrar dinámicamente a las implantaciones de los objetos, lo que le puede permitir a un cliente enviar un objeto para que se ejecute en una máquina con mayor poder de cómputo.

GLOSARIO

- DBMS (*Data Base Management System*): Sistema Manejador de Base de Datos. Consiste en una colección de datos interrelacionados y una colección de programas para acceder a esos datos. El objetivo principal de un DBMS es proporcionar un entorno en el que pueda almacenarse y recuperarse información de forma conveniente y eficiente.

- HTML (*HyperText Markup Language*): Lenguaje de marcas hipertextuales. Lenguaje de computadora empleado para especificar el contenido y el formato de un documento de hipermedios en World Wide Web. Es poco usual que los usuarios se encuentren con el HTML, ya que éste es un detalle interno.

- SQL (*Structured Query Language*): Lenguaje estructurado de consultas. Lenguaje de bases de datos relacional estándar.

Bibliografía

[1] TANENBAUM, Andrew, *Redes de computadoras*, 3ª ed. Prentice Hall.
[2] CHAN, Mark, *1001 Tips para programar con Java*. McGraw-Hill.
[3] FROUFE, Agustín, *JAVA 2, Manual de usuario y tutorial*, 2ª ed. Alfaomega Ra-Ma.
[4] DEITEL & DEITEL, *Cómo programar en Java*. Pearson Educación.

[5] http://www.lab.dit.upm.es/~labscom/almacen/sld/rmi/sld003.htm

[6] http://delta.cs.cinvestav.mx/~oolmedo/CBR/JavaRMI.html

[7] http://www.it.uc3m.es/~jvl/si/rmi/index.html

[8] http://my.execpc.com/~gopalan/misc/compare.html

[9] http://www.javaworld.com/javaworld/jw-05-1996/jw-05-shah.html

[10] http://java.sun.com

[11] http://www.desarrolloweb.com/

[12] http://www.revista.unam.mx/

[13] http://www.lacompu.com/