SISTEMA DE RESERVACIONES DE SALONES USANDO COMO HERRAMIENTAS LIBRERIAS JAVA EN UN ESQUEMA DISTRIBUIDO

Proyecto de Sistemas Distribuidos

Alejandro Blanco-Uribe

Jorge Machado

UNIVERSIDAD CATOLICA ANDRES BELLO

FACULTAD DE INGENIERIA

ESCUELA DE INFORMATICA

Caracas Junio 11 de junio de 2009.

SISTEMA DE RESERVACIONES DE SALONES USANDO COMO HERRAMIENTAS LIBRERIAS JAVA EN UN ESQUEMA DISTRIBUIDO

El objetivo de este proyecto es el de desarrollar un sistema distribuido por medio de la invocación a métodos de objetos remotos (Java RMI) que permita a los usuarios de los salones de la Universidad Católica Andrés Bello reservarlos en una fecha y tiempo determinados para su uso, siempre y cuando los salones estén disponibles. Los usuarios podrán reservar los salones tanto dentro como fuera del campus de la universidad.

PLANTEAMIENTO DEL PROBLEMA

La reservación de salones es un proceso que requiere de una disponibilidad de tiempo y que se lleva actualmente de manera automatizada, el sistema actual se maneja en un enfoque centralizado donde el usuario debe de ir personalmente al lugar donde se presta el servicio (la escuela asociada) de manera de poder reservar el salón y dada la necesidad de este usuario es decir, el salón requerido, se busca en el sistema si el mismo se encuentra libre a las horas en la solicitadas por el usuario.

Este enfoque incurre en que no solamente el usuario debe transportarse al sitio indicado anteriormente, sino que también el sistema no responde ante ciertas necesidades de los usuarios, por ejemplo, si tienen aire acondicionado, video beam, capacidad de puestos, etc. Además el problema del sistema centralizado es que si se cae el servidor entonces los requerimientos de los usuarios no podrán ser atendidos.

Es por esto, que se plantea un sistema bajo un enfoque distribuido en donde el usuario desde su casa pueda realizar esta solicitud sin necesidad de transportarse, además de agregarle otras especificaciones que puedan ser importantes para el usuario (Video beam, aire acondicionado, capacidad, etc.) de manera de mejorar el rendimiento de este proceso y traduciéndose en un mejor tiempo de respuesta. El enfoque distribuido de este sistema se basa en la necesidad de tener varios servidores que puedan responder en caso de que alguno de los mismos se caiga, con el fin de darle respuesta al usuario aunque existan problemas en algunos de ellos y convertir así el sistema que ya existía en uno más robusto.

**INTRODUCCION**

La necesidad de los usuarios ha utilizar sistemas que le permitan realizar sus actividades diarias desde la comodidad de su casa ha crecido notablemente, trayendo como consecuencia la necesidad de mayores y mejores sistemas enfocadas a un modelo cliente servidor. Estos sistemas o aplicaciones con un enfoque cliente - servidor muchas veces traen problemas si se usa un servidor único, ya que si el mismo deja de funcionar (cumplir con las necesidades del cliente) se convierte en una tarea bastante tediosa y en algunos casos, caótica. Aquí entra en juego la necesidad de cambiar el enfoque de un sistema centralizado a uno distribuido de manera que en la mayor cantidad de casos cumplir con las necesidades de los usuarios ya sea balanceando las cargas (las tareas que se deban cumplir) o como respaldo, en donde un servidor pueda con la carga del que dejo de funcionar, siendo esto transparente para el usuario final.

Otro punto importante es la arquitectura del sistema en la que intervienen la forma en que se guarda la información que puede ser realizada de diversas maneras Base de Datos, XML y la plataforma de desarrollo de la aplicación o sistema, es decir, el lenguaje de programación utilizado. Puntos importantes para el sistema ya que los mismos determinan el alcance y la portabilidad del sistema o aplicación y que su definicion requiere de los requerimientos dados por el cliente o simplemente por el levantamiento de información hechos en donde se pueda en base a los mismos tomar dichas decisiones de arquitectura.

En el Sistema implementado el usuario definió parte de la arquitectura del sistema quedando de parte del equipo de desarrollo la mejor implementación de los mismos. Las especificaciones del cliente para este sistema era la realización del sistema en Java y usando como herramienta para el almacenamiento de la información XML. Todo esto bajo un enfoque de servidores distribuidos en donde el sistema sea robusto a caídas

El siguiente trabajo contempla la implementación de las herramientas necesarias bajo la arquitectura solicitada por el cliente buscando cuál de ellas se adapta mejor al problema y resuelvan el mismo de una manera efectiva y eficiente.

MARCO TEORICO

Algunos de los conceptos necesarios para las aplicaciones de las arquitecturas

XML

La definición de Wikipedia "Siglas en inglés de Extensible Markup Language («lenguaje de marcas extensible)," 1. Es un metalenguaje es decir, un lenguaje basado en lenguajes algo análogo al concepto de metada, que son datos que se construyen en base a datos. Permiten definir configuraciones de cualquier tipo (para configurar web services, datos, etc), asi como guardar información como una especie de base de datos, asignándoles atributos y valores. De los cuales se pueden hacer consultas.

**Ventajas**

* Es Adaptable : como se le pueden definir nuevos tipos de etiquetas se puede adaptar
* Como todo xml necesito de cualquier analizador disponible, se pueden evitar problemas futuras, convirtiéndose en un ahorro de tiempo cuando se realizan aplicaciones.
* Es bastante legible, por su forma de estructurar las jerarquías, la forma en que se guardan estos documentos puede dar un formato bastante claro de lo que se esta haciendo con los datos.

Los documentos xml vienen dados por elementos y los identificadores de estos que son las etiquetas, en donde los elementos es la porcion de dato definida entre las etiquetas que a su vez son quienes le dan nombre al elemento.

Documento XML bien formados:

"Los documentos denominados como "bien formados" (del inglés well formed) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (parser) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

\* Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.

\* Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.

\* Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.

\* El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.

\* Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.

\* Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos "entendibles" por las personas."1

Partes de un documento XML

* Prologo: son las primeras líneas del documento que se refieren a la versión del documento, el tipo.
* Cuerpo: como su nombre lo dice contiene un solo elemento raíz y contiene en su interior todos los elementos, es único.
* Elementos como ya se nombró contiene información, o a su vez atributos que lo definen.
* Atributos son las características que definen a un elemento.

**DTD (Document Type Definition)**

Tomado del wikipedia "algunas limitaciones para combinarlos. Los documentos XML que se ajustan a su DTD son denominados válidos.

#### **Declaraciones tipo elemento**

Los elementos deben ajustarse a un tipo de documento declarado en una DTD para que el documento sea considerado como válido.

#### **Modelos de contenido**

Un modelo de contenido es un patrón que establece los subelementos aceptados, y el orden en que se aceptan.

#### **Declaraciones de lista de atributos**

Los atributos se usan para añadir información adicional a los elementos de un documento.

#### **Tipos de atributos**

· Atributos CDATA y NMToken

· Atributos enumerados y notaciones

· Atributos ID e IDREf

#### **Declaración de entidades**

XML hace referencia a objetos que no deben ser analizados sintácticamente según las reglas XML, mediante el uso de entidades. Las entidades pueden ser:

· Internas o externas

· Analizadas o no analizadas

· Generales o parame trizadas

#### **Espacios de nombres**

Los espacios de nombre XML permiten separar semánticamente los elementos que forman un documento XML."1

**XML esquemas**

Es un lenguaje para diseñar un esqueleto de un XML, nos permite definir como será nuestro XML de manera de poder generar múltiples XML que cumplan con ese esqueleto. Fue desarrollado por la W3c y alcanzo el reconocimiento en el 2001.

Estos esqueletos están basando en el lenguaje XML, es decir los mismos se escriben en lenguaje XML, y son una método alternativo a los DTD vistos anteriormente, así mismo los XML esquemas se realizaron pensando en las debilidades y problemas que trae el uso de los DTD, además que incorpora una variedad de tipos de datos que aporta una flexibilidad bastante alta a la hora de colocar la información o en todo caso de darle sentido, por ejemplo colocando datos de tipo Time.

Namespace

Cumplen la misma función que los Namespace del lenguaje .net y los packages de java es decir la información que está dentro de ellos se encuentra en ese espacio de trabajo.

Ventajas de XML Esquemas sobre DTD

Básicamente las ventajas de los XML esquemas sobres los dtd que son mas notables son las siguientes

* XML esquemas se escribe en lenguaje XML
* Proporcionan tipos de datos complejos.
* Se pueden hacer referencias a otros esquemas de manera de armar una jerarquía bastante completa que a veces podría ser compleja.

**DOM**

Básicamente es una librería que permite el uso y manipulación por medio de una interfaz para representar objetos de tipo HTML y XML, esta manipulación permite insertar y actualizar su contenido, estructura o estilo.



"El DOM define la manera en que objetos y elementos se relacionan entre sí en el navegador y en el documento. Cualquier lenguaje de programación adecuado para el desarrollo de página web puede ser utilizado. En el caso de javascript, cada **objeto** tiene un nombre, el cual es exclusivo y único. Cuando existen más de un objeto del mismo tipo en un documento web, estos se organizan en un vector."2

Javadocs

“Es un funcionalidad de Sun Microsystems que genera automáticamente la documentación de APIS en una salida HTML a partir de un código fuente escrito en java, se utilizan ciertos tags al principio o en la cabecera de las funciones que permiten la generación de estas APIS, poseen su formato definido y etiquetas para hacer referencias a cada elemento de la documentación, algunas etiquetas o tags”3

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag** | **Descripción** | **Uso** | **Versión** |
| @author | Nombre del desarrollador. | nombre\_autor | 1.0 |
| @deprecated | Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores. | descripción | 1.0 |
| @param | Definición de un parámetro de un método, es requerido para todos los parámetros del método. | nombre\_parametro descripción | 1.0 |
| @return | Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void". | descripción | 1.0 |
| @see | Asocia con otro método o clase. | referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()). | 1.0 |
| @throws | Excepción lanzada por el método | nombre\_clase descripción | 1.2 |
| @version | Versión del método o clase. | versión | 1.0 |

*Tomado de http://es.wikipedia.org/wiki/Javadoc*

Java RMI

Es una implementación en java para hacer llamadas a métodos de manera remota, lo cual sirve como base para la implementación de aplicaciones distribuidas. Sus características esenciales vienen dadas por la capacidad de pasar objetos por referencia y por claridad a la hora de implementar su código fuente.

Arquitectura de RMI



Tomado de "http://www.proactiva-calidad.com/java/rmi/introduccion.html"

La arquitectura RMI puede verse como un modelo de cuatro capas adaptado a tcp/ip:

"**Primera capa**

La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una interfaz que extienda java.rmi.Remote. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete java.rmi.server), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

**Segunda capa**

La capa 2 es la capa proxy, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

**Tercera capa**

La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream (stream-oriented connection) desde la capa de transporte.

**Cuarta Capa**

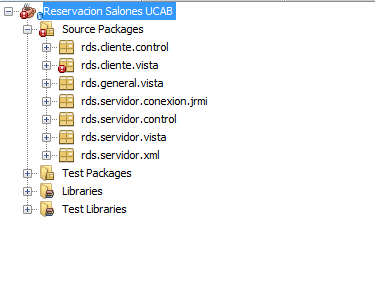
La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol), que solamente es "comprendido" por programas Java."4

JDOM

Es una librería de java que permite el manejo de XML, la librería presenta un API de muchas funcionalidades que te permiten desde crear un XML hasta leerlo y modificarlo, su funcionalidad es bastante parecida a la librería DOM, pero se diferencia en que dom fue pensado para manipulación también de lenguaje de HTML. JDOM es una librería que se creó específicamente para java.

SOLUCION AL PROBLEMA

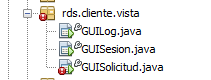
Para plantear una solución al problema se diseñó una aplicación creada en java usando las librerías de la misma, la aplicación se diseñó usando RMI para invocación de objetos de remotos por lo tanto tiene un cliente y un servidor. La arquitectura del sistema quedo de la siguiente manera en la figura 1.



*Figura 1. Paquetes*

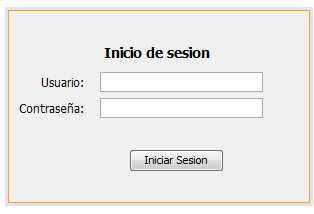
En el cliente control se encuentra el archivo cliente.java (figura 2) donde se encuentra el código del cliente necesario para hacer las llamadas a los servicios remotos y la llamada a la primera ventana que necesita ver el cliente

Posteriormente el paquete cliente vista contiene todas las interfaces de usuario que serán utilizadas por el mismo



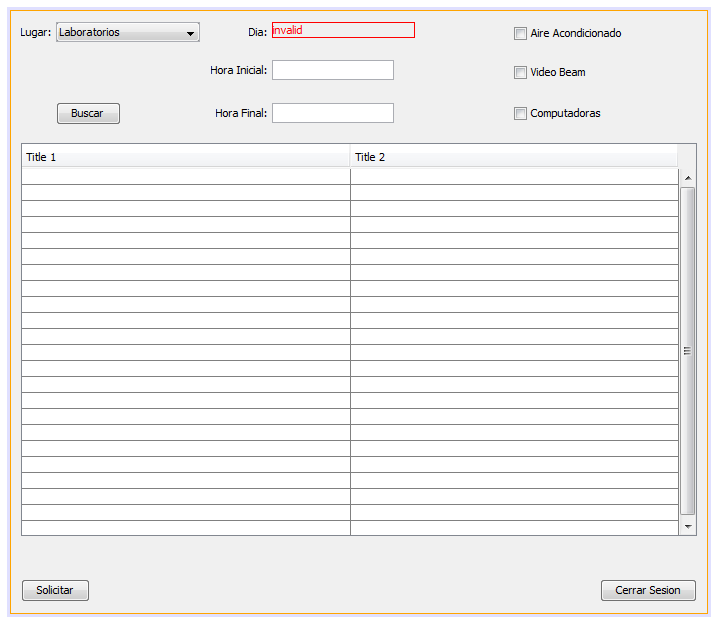
*Figura 2. Cliente Vista*

Las ventanas son las siguientes:



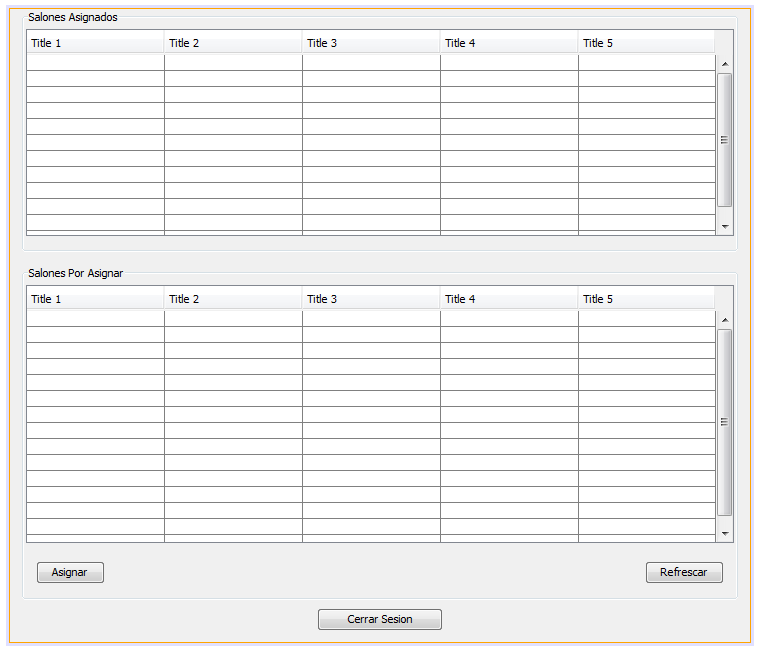
*Figura 3. Ventana para iniciar sesión*

Para loggear el usuario en el sistema (figura 3)



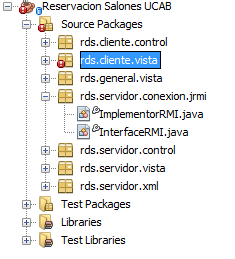
*Figura 4. Ventana para solicitar salones*

Para la solicitud del salón el usuario de ser este un profesor o estudiante, de ser un administrador de sistema le aparecerá la siguiente ventana donde se despliegan todas las solicitudes hechas por los otros usuarios y donde el administrador deberá aceptar la solicitud de salón echa por algún profesor o estudiante.



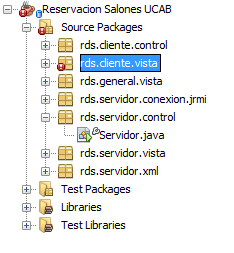
*Figura 4. Ventana para aceptar una solicitud*

En el paquete servidor conexión rmi (Figura 5), se encuentran los archivos implementos e interface rmi que es donde está toda la lógica de programación del servidor



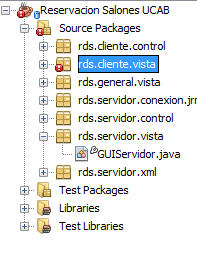
*Figura 5. Servidor Control*

En el paquete servidor control es donde se encuentra todo el código necesario para levantar el servidor que provee los servicios a pedir por el cliente



*Figura 6. Cliente vista*

Por ultimo en servidor vista (Figura 6) se realizó una interface de usuario para ver que el servidor se haya levantado de manera correcta.



En la implementación de la solución se usó como manejador de XML la librería Jdom, se tomó esta decisión de usar JDOM porque es una herramienta echa específicamente para java lo cual es una ventaja ya que provee una alta compatibilidad y significa que la forma en la que se realizan los comandos para usar la librería son nativos y con un formato conocido a lo que se viene trabajando con java.

Otra decisión que se tomó para la solución del sistema es que para guardar la información de los usuarios y de los salones solicitados, fueron los XML esquemas esta solución se tomó en base a que los XML esquemas se escriben en código XML, además que netbeans(que fue el IDE utilizado) posee un plugin que te permite construir el XML esquema de una manera sencilla, además que luego de generar el XML esquema la herramienta te permite generar el XML donde vas a guardar la información respectiva. En general se crearon los siguientes XML esquemas:

Para el usuario, para el servidor, para las horas en específico y salones que usa una referencia al XML de las horas.

En las decisiones de diseño se usó para la implementación de los objetos remotos RMI, pero en la arquitectura del diseño de rmi solo se usaron las clases de implementación, la interface y el servidor, es decir no se usó el Helper.

En la clase Implementor es donde se encuentran todas las funciones necesarias para que el servidor preste el servicio al cliente. En la interface se encuentra la cabeceras de todos los procedimientos que están en el implementor y que pueden ser invocadas por el cliente.

Descripción de las funciones del implementor:

Public Boolean AutenticarUsuario(String nombreusuario, String clave) throws RemoteException;

Es una función que devuelve un booleano pidiendo el nombre de usuario y la contraseña del cliente si se pudo autenticar la persona, es decir si existe en la base de datos.

Public List<String> DatosUsuario(String nombreusuario,String clave) throws RemoteException;

Devuelve una lista en donde cada posición es un dato del usuario, dada el nombre de usuario y la contraseña del cliente.

Public void SolicitarSalon(String usuario, String fecha, String hora, String salon, String tipoUsuario) throws RemoteException;

Recibe los datos del salón y del usuario de manera de ingresar en el log la información del salón y quien fue el usuario que lo solicito.

public List<String> BuscarSalonDisponible(String salon,String dia,String pedidoInicial,String pedidoFinal,String videoBeam,String aa,String computadora) throws RemoteException;

Dado los datos del salón busca en la base de datos si se encuentra un salón disponible con la información requerida.

Public String getTipoUsuario(String usuario) throws RemoteException;

Dado el nombre de usuario te devuelve su tipo de usuario es decir si es un profesor, estudiante o un administrador del sistema.

RESULTADOS OBTENIDOS

Básicamente el problema se resolvió, con la solución propuesta se puede buscar en base a los parámetros solicitados por el usuario un salón, también se cumple con los requerimientos de tener tipos de usuarios y controles que permitan regular su acceso al sistema, todo esto haciendo llamadas a objetos remotos. Se logró generar un log que tiene la salida de quien pidió el salón. El problema con el log es que se sobrescribía cada vez que se solicitaba el salón, es decir no se logró agregar cada nueva solicitud.

Lo que no se logro fue colocar el sistema en un esquema distribuido, es decir, que si se cae el servidor el sistema no puede seguir funcionando, esto se debió a la dependencia del cliente con el servidor usando rmi, es decir la dirección que usa el cliente para conectarse con el servidor esta cableada y es siempre dependiente del servidor. La solución a este problema era la inclusión de un helper para que dirección y la conexión remota no dependiera directamente de la dirección de un servidor.

CONCLUSIONES

El uso de XML como sistema para almacenar y manejar los datos es una buena opción cuando se trata de aplicaciones pequeñas y donde las entidades involucradas no tienen relaciones complejas, sin embargo a veces su implementación de puede complicar dependiendo de que forma se utilice para su creación.

La tecnología Java Rmi nos da grandes ventajas, es sencilla de implementar y permite ejecutar de manera remota cualquier operación que necesitemos ahorrándonos el uso de servicios web que requieren de una implementación más compleja. Sin embargo el desconocimiento del alcance de las herramientas y el tiempo son factores determinantes para la implementación de sistemas, quizás como recomendaciones futuras seria el uso de esquemas XML y como manejador el JDOM, pudiéndose explicar la compilación para crear el .jar y su implementación al sistema, ya que la investigación para el uso de esta herramienta puede influir en el tiempo de entrega.

Por último el desarrollo de este sistema fue básicamente un trabajo arduo para la implementación de los salones ya que el uso de ciertas librerías para el manejo de horas, puede ser complejo y el manejo de las horas como enteros es una tarea larga más no imposible.

Referencias Bibliográficas

1 <http://es.wikipedia.org/wiki/XML>

2 <http://es.wikipedia.org/wiki/DOM>

3 <http://es.wikipedia.org/wiki/Javadoc>

4 <http://www.proactiva-calidad.com/java/rmi/introduccion.html>