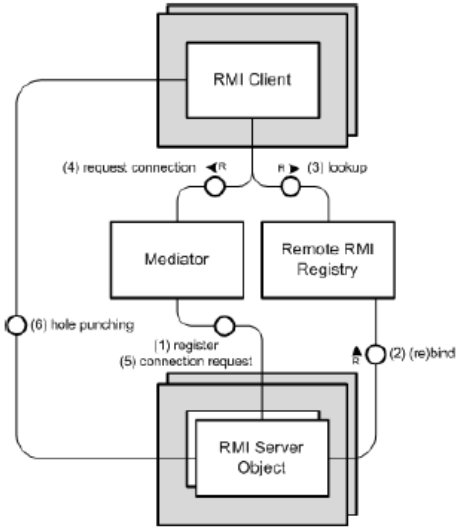


Java RMI

Sistemas Distribuidos

Rodrigo Santamaría

Java RMI

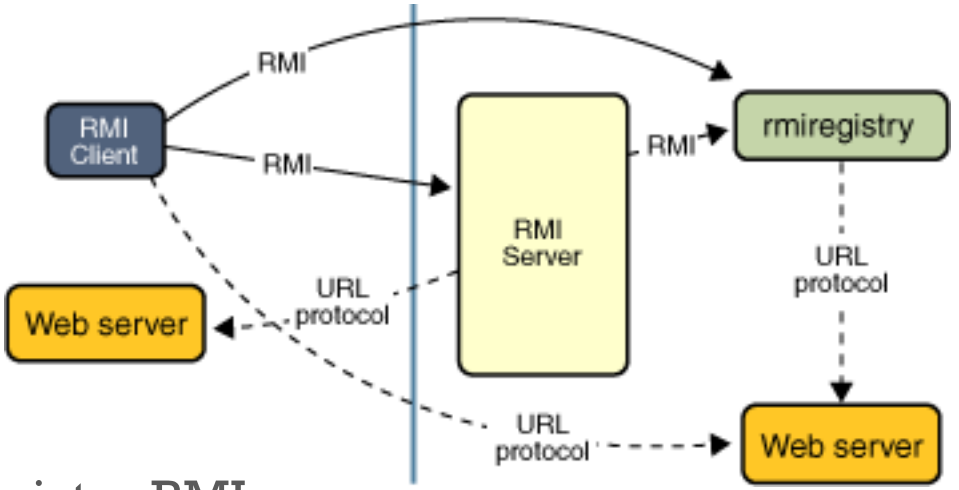
- RMI
- Java RMI

RMI

- **Remote Method Invocation**: middleware para que un objeto que se ejecuta en una JVM use métodos de otro objeto que se ejecuta en otra JVM (local o *remota*)
  - Un paso más allá de los sockets
- Introducción a RMI y tutorial:
  - http://download.oracle.com/javase/tutorial/rmi/overview.html
- Tutorial de sockets:
  - http://download.oracle.com/javase/tutorial/networking/sockets/clientServer.html

RMI

Aplicación



- Aplicación basada en RMI
- Dos fases fundamentales
  - Localizar objetos remotos
    - Registrados mediante el registro RMI
    - Pasados por referencia en invocaciones remotas
  - Comunicarse con objetos remotos
    - Gestionado por el servidor RMI, para el usuario es como llamar a métodos locales

RMI vs Sockets

RMI	Sockets
Invocación de objetos remotos	Invocación de métodos remotos
Sencillo	Complicado
No hay un protocolo	Necesidad de un protocolo
Genera mucho tráfico Stub+registro+objetos	Genera poco tráfico

En el fondo, RMI = Sockets + Serialización + Algunas utilidades

RMI

Implementación

1. Definir **interfaz** con los métodos remotos
   - Será conocida por cliente y servidor
2. Implementar el **servidor**
   1. El elemento que dará el servicio de la interfaz
3. Instanciar el servidor y registrarlo mediante un ***stub***:
   - Referencia remota al servidor generada por RMI para el uso de los clientes
4. Implementar el **cliente** que usará el servicio

Java RMI

- RMI
- Java RMI

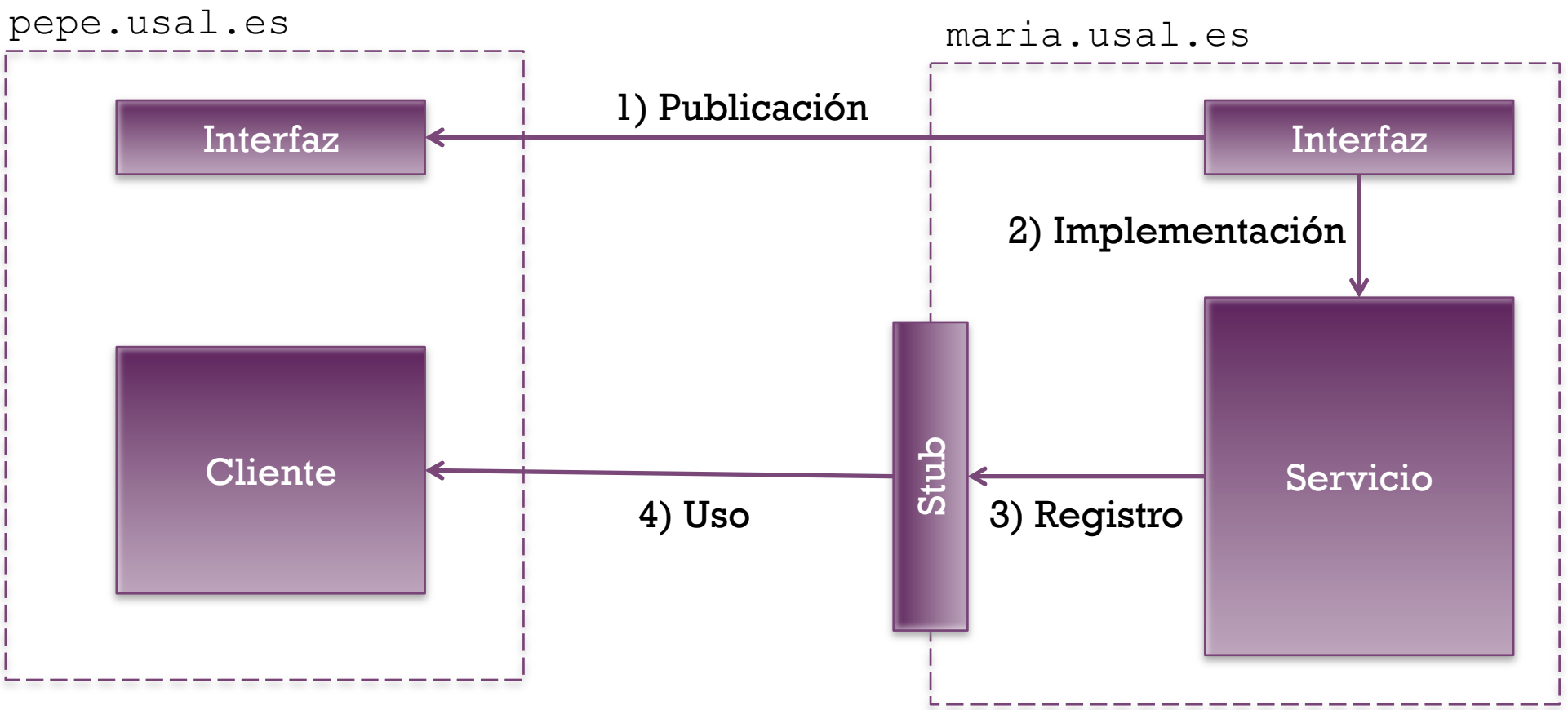
Java RMI

Implementación

1. **Interfaz**: clase que extiende java.rmi.Remote
2. **Servidor**: clase que implementa la interfaz
   - Puede tener más métodos que los de la interfaz
3. **Stub**: instancia de la interfaz asociada a un servidor
   - Sólo contiene los métodos de la interfaz
   - Es la que se registra en RMI
4. **Cliente:** cualquier clase que localice el stub y use su interfaz

Java RMI

Esquema



Java RMI

Interfaz

```
/**
 * Interfaz para un servicio RMI de corredores de bolsa
 * Debe heredar de java.rmi.Remote
 * Debe manejar RemoteException en sus métodos
 */
public interface Corredor extends Remote
 {
  String listarTitulos() throws RemoteException;
  void comprar(String nombre, int cantidad) throws RemoteException;
  void vender(String nombre, int cantidad) throws RemoteException;
  }
```

Java RMI

Interfaz: publicación

- La interfaz debe ser **accesible a cliente y servidor**
- La interfaz debe ser **idéntica** en ambos extremos
- Tres formas de publicación
  - Como clases sin compilar (ficheros .java)
    - Si los compiladores o los ficheros no son iguales, dará error
  - Como clases compiladas (ficheros .class)
    - Si las versiones de java no son iguales, puede dar error
  - Como archivo .jar
    - La mejor opción:

```
cd /home/usuario/workspace/rmiInterface/src
javac rmiInterface/Corredor.java
jar cvf bolsaInterface.jar rmiInterface/*.class
```

Java RMI

Interfaz: codebase

- Lugar desde el que se cargan las clases en la JVM
  - P. ej. el CLASSPATH es un codebase "local"
- El codebase "remoto" se usa para acceder a las clases desde applets o RMI, mediante URLs
- El codebase se puede modificar
  - Como argumento de la JVM: `-Djava.rmi.codebase="url"`
  - Desde el código:
    - `System.setProperty("java.rmi.server.codebase", "url");`
- En Java RMI, la url del codebase debe apuntar a la localización de las interfaces compartidas

Java RMI

Interfaz: URLs

- En el caso de una carpeta con clases (.java o .class), se referencia la carpeta:
  - "file:///Users/rodri/Documents/workspace/assoo/bin/"
- En el caso de un único fichero, lo referenciamos directamente
  - "file:///Users/rodri/Documents/workspace/interface.jar"
- En teoría, nuestra interfaz podría estar publicada y acceder a ella mediante una url remota:
  - "http://vis.usal.es/rodrigo/documentos/aso/rmi/interface.jar"
  - Sin embargo, muchas implementaciones de RMI no lo soportan o requieren reconfiguraciones del registro

Java RMI

Servidor y Stub

```
/**
 * Servidor de bolsa que implementa Corredor
 * Debe implementar una interfaz de java.rmi.Remote
 *
 */
public class Bolsa implements Corredor
 {
  String listarTitulos() throws RemoteException
      {...}
  void comprar(String nombre, int cantidad) throws RemoteException;
      {...}
  void vender(String nombre, int cantidad) throws RemoteException;
      {...}
  //Cualquier otra función interna que sea necesaria
  }
```

```
  //Registramos un objeto Bolsa de nombre “LaBolsa”
  String nombre="LaBolsa";
  Corredor motor=new Bolsa();
  Corredor stub=(Corredor) UnicastRemoteObject.exportObject(motor,0);
  Registry registro=LocateRegistry.getRegistry();
  registro.rebind(nombre,stub);
```

Java RMI

Stub: registro

- Para poder registrar el stub al nombre de servicio, debe estar activado el registro RMI (*rmiregistry*)
- Tres maneras:
  - Desde un terminal con
    - `rmiregistry [port]`
  - Desde java con
    - `Runtime.getRuntime().exec("rmiregistry");`
  - Desde java con
    - `LocateRegistry.createRegistry(int port);`
  - Cuidado: no iniciar un registro si ya hay otro corriendo

Java RMI

Stub: registro

- Dos modos de asociar un stub a un registro desde Java:
  - Mediante java.rmi.registry.Registry
    - `Registry registro=LocateRegistry.getRegistry();`
    - `registro.rebind(nombre,stub);`
  - Mediante java.rmi.Naming
    - `Naming.rebind(nombre, stub)`

Java RMI

Cliente

```
public class Cliente
 {
  public static void main(String args[])
   {
    try
     {
     String nombre="LaBolsa";
     //Instanciar el registro RMI
     Registry registro=LocateRegistry.getRegistry(args[0]);
     //Instanciar un objeto de la clase del servidor
     Corredor corredor=(Corredor) registro.lookup(nombre);
     //Uso del servicio
     ...
     }
    catch (Exception e)
      {
      System.err.println("Excepción en el cliente de la bolsa:");
      e.printStackTrace();
      }
   }
 }
```

Java RMI

Cliente: búsqueda

- Dos modos de buscar el stub de un servicio:
  - Mediante java.rmi.registry.Registry
    - `Registry registro=LocateRegistry.getRegistry();`
    - `registro.lookup(nombre);`
  - Mediante java.rmi.Naming
    - `Naming.lookup(nombre)`

Java RMI

Gestor de seguridad

- Si nuestra interfaz contiene métodos que requieren como argumentos o devuelven clases *distintas* del API de Java, hay que implementar un **gestor de seguridad**
  - Para evitar intrusiones de código maligno

- Para activar el gestor:

```
if (System.getSecurityManager()==null)
    System.setSecurityManager(new SecurityManager());
```

- Podemos modificar la política de seguridad de Java con un fichero de permisos con líneas como:

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

  - Y especificar el fichero con la opción de la JVM
    - `-Djava.security.policy=grantFilePath`