

Paso de Mensajes con RMI

Tema 7 - Programación Concurrente y de Tiempo Real

Antonio J. Tomeu¹ Manuel Francisco²

¹Departamento de Ingeniería Informática
Universidad de Cádiz

²Departamento de CC. de la Computación e I.A.
Universidad de Granada

PCTR, 2019

Contenido

1. Conceptos de Programación Distribuida
2. Remote Method Invocation (RMI) en Java
3. El Nivel de Resguardos
4. La Responsable de que Todo Funcione: la interfaz
5. El precompilador rmic
6. Arquitectura Completa de una Aplicación
7. Evoluciones Reseñables

Conceptos de Programación Distribuida

- ▶ No existe memoria común
- ▶ Comunicaciones
- ▶ No existe un estado global del sistema
- ▶ Grado de Transparencia
- ▶ Escalables
- ▶ Reconfigurables

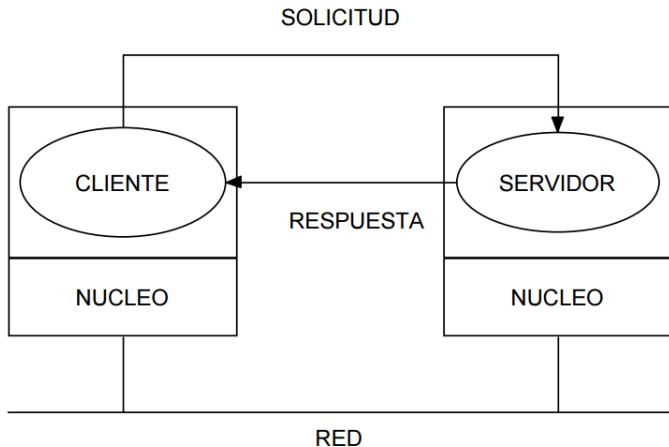
Modelos de Programación Distribuida

- ▶ Modelo de paso de mensajes
 - ▶ Operaciones send y receive
- ▶ Modelo RPC
 - ▶ Stubs de cliente y servidor
 - ▶ Visión en Java es RMI
- ▶ Modelos de Objetos Distribuidos
 - ▶ DCOM de Microsoft
 - ▶ Jini de SUN
 - ▶ CORBA de OMG

Modelo de Paso de Mensajes

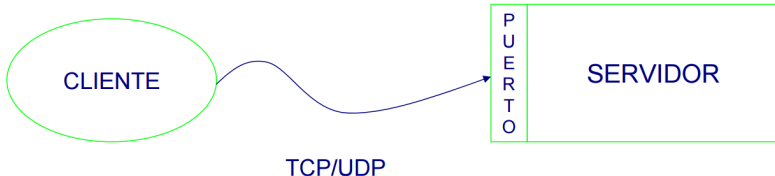
- ▶ Mecanismo para comunicar y sincronizar entidades concurrentes que no pueden o no quieren compartir memoria
- ▶ Llamadas send y receive
- ▶ Tipología de las llamadas:
 - ▶ Bloqueadas - No bloqueadas
 - ▶ Almacenadas - No almacenadas
 - ▶ Fiables - No Fiables
- ▶ En UNIX:
 - ▶ Pipes con y sin nombres
- ▶ En Java:
 - ▶ Sockets: Clases Socket y ServerSocket
 - ▶ Canales: CTJ (Communicating Threads for Java)

Modelo General

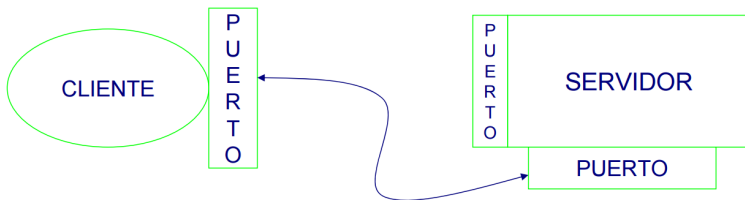


Puertos (Breve Repaso) I

- ▶ El protocolo TCP (y también UDP) utilizan los puertos para hacer llegar los datos de entrada a un proceso concreto que se ejecuta en una máquina



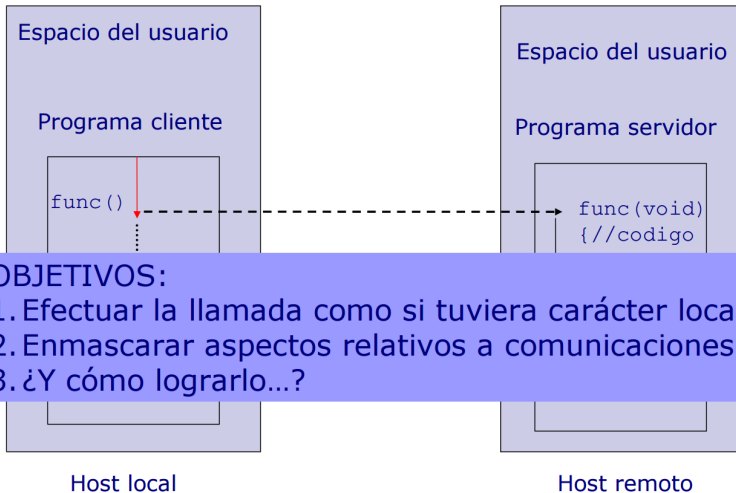
Puertos (Breve Repaso) II



Modelo Remote Procedure Call (RPC)

- ▶ Mayor nivel de abstracción
- ▶ Llamadas a procedimiento local o remoto indistintamente
- ▶ Necesita de stubs/skeletons (¡OCULTAMIENTO!)
- ▶ Registro del servicio (Servidor de Nombres)
- ▶ En Unix:
 - ▶ Biblioteca `rpc.h`
 - ▶ Represtación estándar XDR
 - ▶ Automatización parcial: especificación-compilador `rpcgen`
- ▶ En Java:
 - ▶ Objetos remotos. Serialización
 - ▶ Paquete `java.rmi`
 - ▶ Automatización parcial: interfaz-compilador `rmic`

Llamando a un Procedimiento Remoto



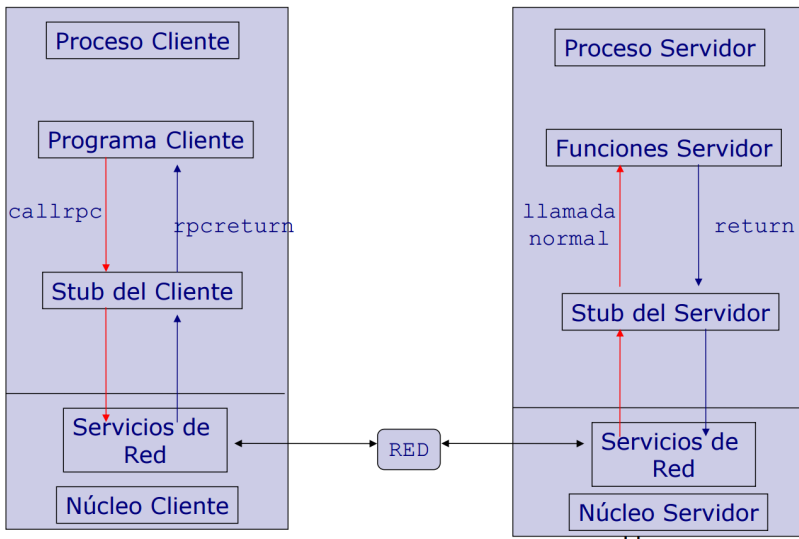
OBJETIVOS:

1. Efectuar la llamada como si tuviera carácter local
2. Enmascarar aspectos relativos a comunicaciones
3. ¿Y cómo lograrlo...?

Introducción del Nivel de Stubs

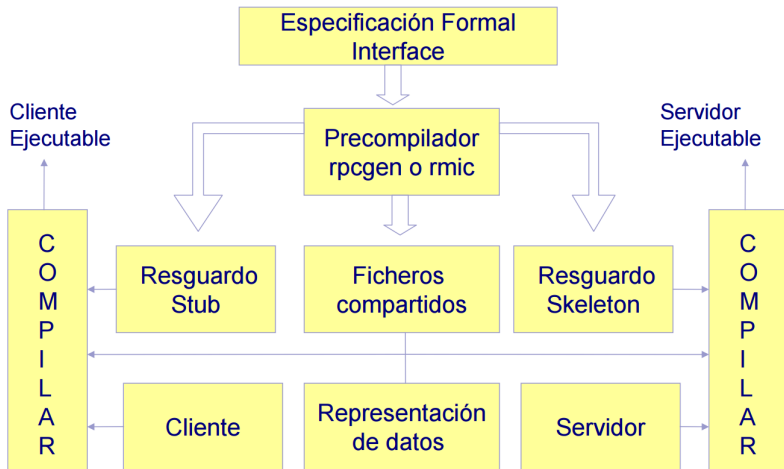
- ▶ Efectúan el marshalling/unmarshalling de parámetros.
- ▶ Bloquean al cliente a la espera del resultado.
- ▶ Transparencia de ejecución remota
- ▶ Interface con el nivel de red
 - ▶ TCP
 - ▶ UDP

Esquema RPC con Stubs

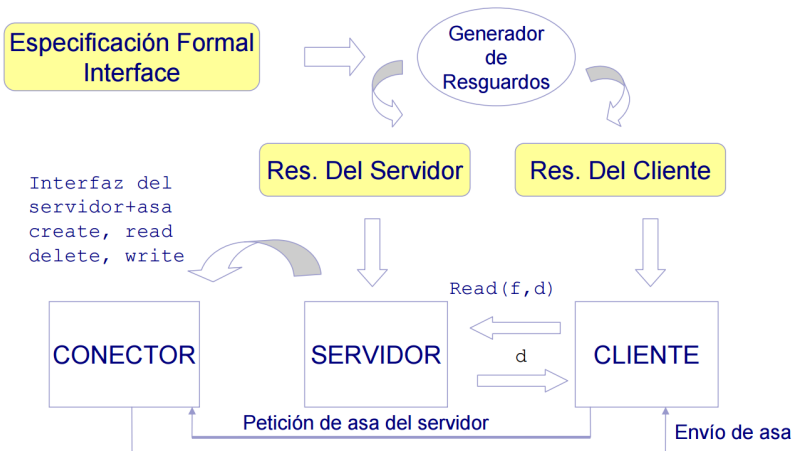


- ▶ De forma manual
- ▶ De forma automática:
 - ▶ Especificación formal de interfaz
 - ▶ Software específico
 - ▶ rpcgen (C-UNIX) / rmic (Java)

Generación Automática de Stubs



Dynamic Binding



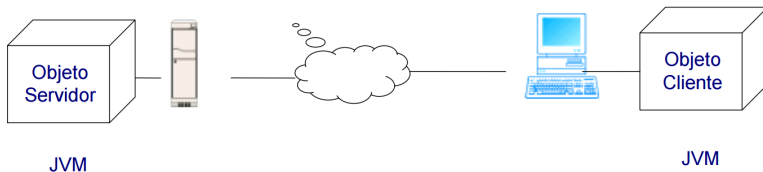
- Especificación de la interfaz del servidor remoto en un fichero de especificación .x

```
/*rand.x*/  
program RAND_PROG{  
    version RAND_VER{  
        void inicia_aleatorio(long)=1;  
        double obtener_aleat(void)=2;  
    }=1;  
}=0x3111111;
```

- Generación automática de resguardos: rpcgen rand.x
- Distribuir y compilar ficheros entre las máquinas implicadas
- Necesario utilizar representación XDR. Biblioteca xdr.h

RMI (Remote Method Invocation) en Java

- ▶ Permite disponer de objetos distribuidos utilizando Java
- ▶ Un objeto distribuido se ejecuta en una JVM diferente o remota
- ▶ **Objetivo:** lograr una referencia al objeto remoto que permita utilizarlo como si el objeto se estuviera ejecutando sobre la JVM local
- ▶ Es similar a RPC, si bien el nivel de abstracción es más alto
- ▶ Se puede generalizar a otros lenguajes utilizando JNI
- ▶ RMI pasa los parámetros y valores de retorno utilizando serialización de objetos.



RPC vs. RMI

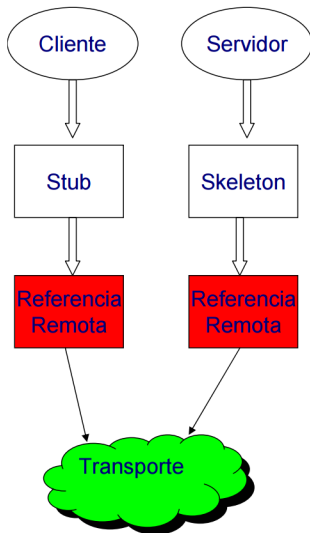
RPC	RMI
Carácter y Estructura de diseño procedimental	Carácter y Estructura de diseño orientada a objetos
Es dependiente del lenguaje	Es dependiente del lenguaje
Utiliza la representación externa de datos XDR	Utiliza la serialización de objetos en Java
El uso de punteros requiere el manejo explícito de los mismos	El uso de referencias a objetos locales y remotos es automático
NO hay movilidad de código	El código es móvil, mediante el uso de bytecodes.

Llamadas Locales Vs. Llamadas Remotas

- ▶ Un objeto remoto es aquél que se ejecuta en una JVM diferente, situada potencialmente en un host distinto.
- ▶ RMI es la acción de invocar a un método de la interfaz de un objeto remoto.

```
1 //ejemplo de llamada a metodo local
2 int dato;
3 dato = Suma (x,y);
4 //ejemplo de llamada a metodo remoto (no completo)
5 IEjemploRMI1 ORemoto =
6 (IEjemploRMI1)Naming.lookup("//sargo:2005/EjemploRMI1");
7 ORemoto.Suma(x,y);
```

Arquitectura RMI I

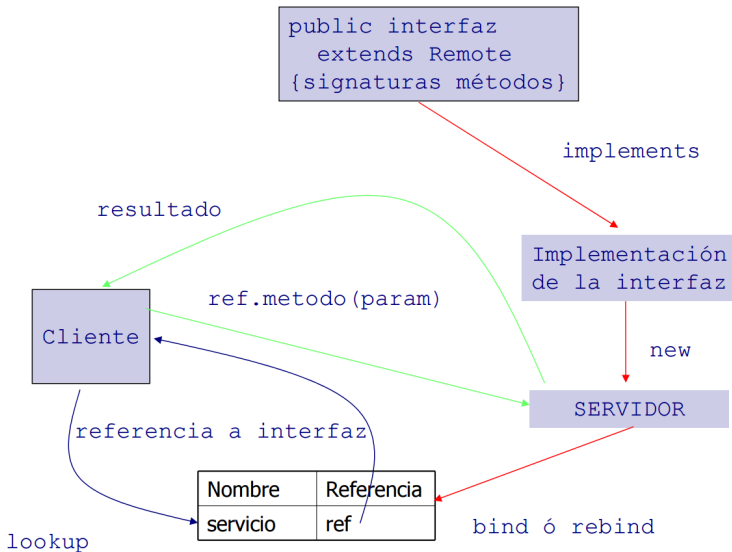


- ▶ El servidor debe extender `RemoteObject`
- ▶ El servidor debe implementar una interfaz diseñada previamente
- ▶ El servidor debe tener como mínimo un constructor (nulo) que lanzará la excepción `RemoteException`
- ▶ El método `main` del servidor debe lanzar un gestor de seguridad
- ▶ El método `main` crea los objetos remotos
- ▶ El compilador de RMI (`rmic`) genera el stub y el skeleton.
- ▶ Los clientes de objetos remotos se comunican con interfaces remotas (diseñadas antes de...)
- ▶ Los objetos remotos son pasados por referencia
- ▶ Los clientes que llaman a métodos remotos deben manejar excepciones.

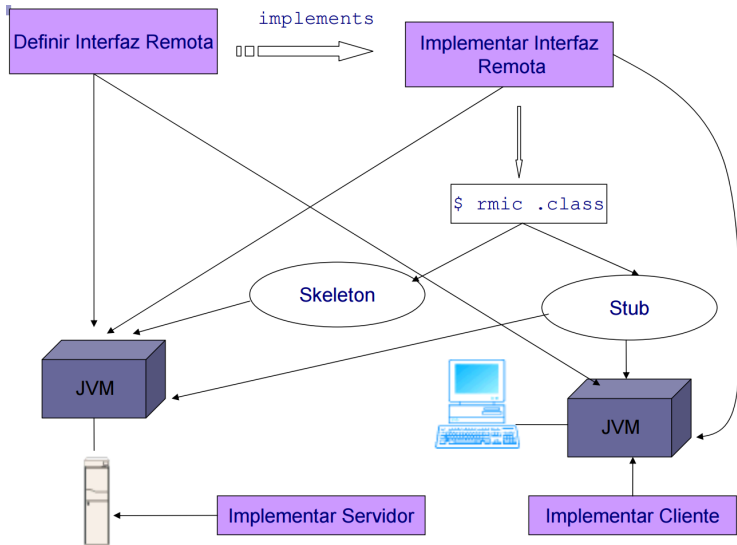
La Clase java.rmi.Naming

```
1 public static void bind(String name, Remote obj)
2 public static String[] list(String name)
3 public static Remote lookup(String name)
4 public static void rebind(String name, Remote obj)
5 public static void unbind(String name)
```

¿Cómo Lograrlo? I



¿Cómo Lograrlo? II



Fases de Diseño de RMI (1 - Interface)

- ▶ Escribir el fichero de la interfaz remota.
- ▶ Debe ser public y extender a Remote.
- ▶ Declara todos los métodos que el servidor remoto ofrece, pero NO los implementa. Se indica nombre, parámetros y tipo de retorno.
- ▶ Todos los métodos de la interfaz remota lanzan obligatoriamente la excepción RemoteException
- ▶ El propósito de la interface es ocultar la implementación de los aspectos relativos a los métodos remotos.
- ▶ De esta forma, cuando el cliente logra una referencia a un objeto remoto, en realidad obtiene una referencia a una interfaz.
- ▶ Los clientes envían sus mensaje a los métodos de la interfaz

Ejemplo: Implementación Interfaz y Servidor I

Código 1: codigos_t7/EjemploRMI1.java

```
1  /**
2   * Ejemplo de implementacion del interfaz remoto para un RMI
3   * @author Antonio Tomeu
4   */
5
6  //se importan los paquetes necesarios
7  import java.rmi.*;
8  import java.rmi.server.*;
9  import java.rmi.registry.*;
10 import java.net.*;
11
12 public class EjemploRMI1
13 extends UnicastRemoteObject //el servidor debe siempre extender
    esta clase
14 implements IEjemploRMI1 //el servidor debe siempre implementar
    la interfaz
15 //remota definida con caracter previo
16 {
17     public int Suma(int x, int y)
```

Ejemplo: Implementación Interfaz y Servidor II

```
18     throws RemoteException {
19         return x + y;
20     }
21
22     public int Resta(int x, int y)
23     throws RemoteException {
24         return x - y;
25     }
26
27     public int Producto(int x, int y)
28     throws RemoteException {
29         return x * y;
30     }
31
32     public float Cociente(int x, int y)
33     throws RemoteException {
34         if (y == 0) return 1 f;
35         else return x / y;
36     }
37
38     //es necesario que haya un constructor (nulo) como minimo, ya
        que debe
```

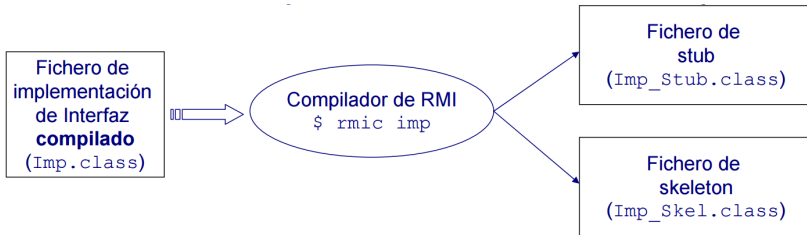
Ejemplo: Implementación Interfaz y Servidor III

```
39 //lanzar RemoteException
40 public EjemploRMI1()
41 throws RemoteException { //super(); invoca automaticamente al
    constructor de la superclase
42 }
43
44 //el metodo main siguiente realiza el registro del servicio
45
46 public static void main(String[] args)
47 throws Exception {
48     //crea e instala un gestor de seguridad que soporte RMI.
49     //el usado es distribuido con JDK. Otros son posibles.
50     //En esta ocasion trabajamos sin el.
51     //System.setSecurityManager(
52     //new RMISecurityManager());
53
54     //Se crea el objeto remoto. Podriamos crear mas si interesa.
55     IEjemploRMI1 ORemoto = new EjemploRMI1();
56
57     //Se registra el servicio
58     Naming.bind("Servidor", ORemoto);
59
```

Ejemplo: Implementación Interfaz y Servidor IV

```
60     System.out.println("Servidor Remoto Preparado");  
61 }  
62 }
```

Fases de Diseño RMI (3 - STUB y SKELETON) I



```
rmic -vcompat EjemploRMI1
```

Fases de Diseño RMI (3 - STUB y SKELETON) II

- ▶ Es necesario que en la máquina remota donde se aloje el servidor se sitúen también los ficheros de stub y skeleton. (Cambios a partir de 1.5)
- ▶ Con todo ello disponible, se lanza el servidor llamando a JVM del host remoto, previo registro en un DNS
- ▶ En nuestro caso, el servidor remoto se activó en hercules.uca.es sobre el puerto 1099.

```
java EjemploRMI1 &
```

Fases de Diseño RMI (4 - 'Registro)

- ▶ Método `Naming.bind("Servidor",ORemoto);` para registrar el objeto remoto creado requiere que el servidor de nombres esté activo.
- ▶ Dicho servidor se activa con `start rmiregistry` en Win32
- ▶ Dicho servidor se activa con `rmiregistry &` en Unix.
- ▶ Se puede indicar como parámetro el puerto que escucha.
- ▶ Si no se indica, por defecto es el puerto 1099.
- ▶ El parámetro puede ser el nombre de un host como en `Naming.bind("//sargo.uca.es:2005/Servidor",ORemoto);`

Fases de Diseño RMI (5 - Cliente) I

- ▶ El objeto cliente procede siempre creando un objeto de interfaz remota.
- ▶ Posteriormente efectúa una llamada al método `Naming.lookup` cuyo parámetro es el nombre y puerto del host remoto junto con el nombre del servidor.
- ▶ `Naming.lookup` devuelve una referencia que se convierte a una referencia a la interfaz remota.
- ▶ A partir de aquí, a través de esa referencia el programador puede invocar todos los métodos de esa interfaz remota como si fueran referencias a objetos en la JVM local.

Fases de Diseño RMI (5 - Cliente) II

Código 2: codigos_t7/ClienteEjemploRMI1.java

```
1  /**
2   * Ejemplo de implementacion de un cliente para RMI
3   * @author Antonio Tomeu
4   */
5   import java.rmi.*;
6   import java.rmi.registry.*;
7
8   public class ClienteEjemploRMI1 {
9       public static void main(String[] args)
10          throws Exception {
11           int a = 10;
12           int b = -10;
13
14           //En esta ocasion trabajamos sin gestor de seguridad
15           //System.setSecurityManager(new RMISecurityManager());
16
17           //Se obtiene una referencia a la interfaz del objeto remoto
18           //SIEMPRE debe convertirse el retorno del metodo Naming.lookup
19           //a un objeto de interfaz remoto
20
21           IEjemploRMI1 RefObRemoto =
```

Fases de Diseño RMI (5 - Cliente) III

```
22      (IEjemploRMI1) Naming.lookup("//localhost/Servidor");
23
24      System.out.println(RefObRemoto.Suma(a, b));
25      System.out.println(RefObRemoto.Resta(a, b));
26      System.out.println(RefObRemoto.Producto(a, b));
27      System.out.println(RefObRemoto.Cociente(a, b));
28
29  }
30 }
```

Distribución de Archivos

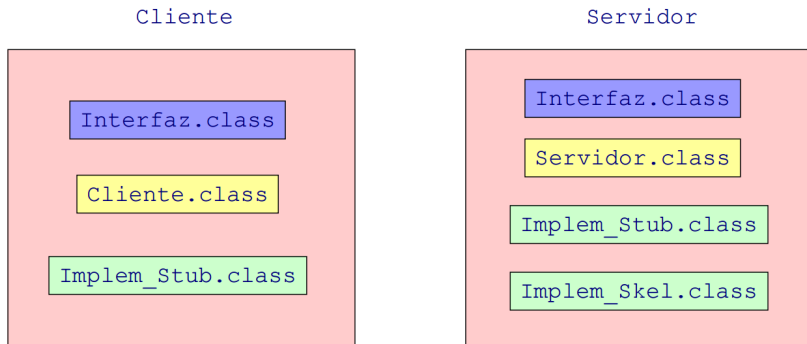


Figura: A partir de la JDK 1.2 las clases SKELETON no son necesarias

- ▶ Descargue los ficheros IEjemploRMI1.java, EjemploRMI1.java y ClienteEjemploRMI1.java
- ▶ Siga el guión auxiliar de prácticas (ver bloque del tema actual) y desarrolle la arquitectura en local
- ▶ Distribuya ahora la aplicación con un compañero
- ▶ Escriba una interfaz de operaciones para números complejos, utilizando tipos primitivos, llamada IComplejo.java
- ▶ Impleméntela en Complejo.java
- ▶ Escriba código de servidor y cliente que hagan uso de la misma. Guárdelos en ServerComplejo.java y ClientComplejo.java
- ▶ Distribuya nuevamente la aplicación
- ▶ Acuerden una interfaz llamada interfaz_grupo_x.java

Ejercicio II

- ▶ Un miembro del grupo escribirá el `cliente_grupo.java` y el otro el `servidor_grupo_x.java` de manera independiente. Una vez hecho, lancen la arquitectura rmi.
- ▶ Suban el Espacio de los alumnos un fichero `rmi_grupo_x.rar` que contengan los tres ficheros.

- ▶ A partir del JDK 1.2 la implementación de RMI no necesita skeletons (con ese nombre tenían poco futuro). El stub estará en ambos lados.
- ▶ Recompila los ejemplos anteriores sin el flag `-vcompat`
- ▶ A partir del JDK 1.5 se incorpora Generación Dinámica de Resguardos

- ▶ Pruebe los ejemplos anteriores sin skeleton
- ▶ No obstante, ¿necesita algo el servidor a pesar de todo?

- ▶ Serialización de Objetos
- ▶ Gestión de Seguridad (policytool)
- ▶ *Callback* de Cliente
- ▶ Descarga Dinámica de Clases

- ▶ Serializar un objeto es convertirlo en una cadena de bits, posteriormente restaurada en el objeto original
- ▶ Es útil para enviar objetos complejos en RMI
- ▶ Tipos primitivos se serializan automáticamente
- ▶ Clases contenedoras también
- ▶ Objetos complejos deben implementar la interfaz Serializable (es un flag) para poder ser serializados

Código 3: codigos_t7/Arbol.java

```
1  /**
2   * @(#)Arbol.java
3   *
4   *
5   * @author El Pupas
6   * @version 1.00 2009/5/21
7   */
8
9
10 import java.io.*;
11
12 public class Arbol
13     implements java.io.Serializable
14 {
15     public Arbol izq;
16     public Arbol der;
17     public int id;
18     public int nivel;
19     private static int cont = 0;
20
21     public Arbol(int l) {
```

Serialización III

```
22     id = cont++;
23     nivel = l;
24     if (l > 0) {
25         izq = new Arbol(l-1);
26         der = new Arbol(l-1);
27     }
28 }
29 }
```

Código 4: codigos_t7/IArbol.java

```
1  /**
2   * @(#)IArbol.java
3   *
4   *
5   * @author Antonio Tomeu
6   * @version 1.00 2009/5/21
7   */
8
9  import java.rmi.*;
10 public interface IArbol
11     extends Remote
```

Serialización IV

```
12 {  
13     public void Listado_Remoto (Arbol t, int n) throws  
14         RemoteException;  
15 }
```

Código 5: codigos_t7/Serv_arbol.java

```
1  /**  
2   * @(#)Serv_arbol.java  
3   *  
4   *  
5   * @author Antonio Tomeu  
6   * @version 1.00 2009/5/21  
7   */  
8  
9  import java.rmi.*;  
10 import java.rmi.server.*;  
11 import java.rmi.registry.*;  
12 import java.net.*;  
13  
14 public class Serv_arbol
```

Serialización V

```
15     extends UnicastRemoteObject
16     implements IArbol
17 {
18
19     public Serv_arbol()
20     throws RemoteException
21     {super();}
22
23     public void Listado_Remoto (Arbol t, int n)
24     throws RemoteException
25     {
26
27         for (int i = 0; i < t.nivel; i++)
28             System.out.print("  ");
29         System.out.println("nodo " + t.id);
30
31         if (t.nivel <= n && t.izq != null)
32             Listado_Remoto(t.izq, n);
33
34         if (t.nivel <= n && t.der != null)
35             Listado_Remoto(t.der, n);
36     }
```

Serialización VI

```
37
38
39 public static void main(String[] args)
40     throws Exception
41 {
42     Serv_arbol ORemoto = new Serv_arbol();
43     Naming.bind("Servidor", ORemoto);
44     System.out.println("Servidor Remoto Preparado");
45 }
46
47 }
```

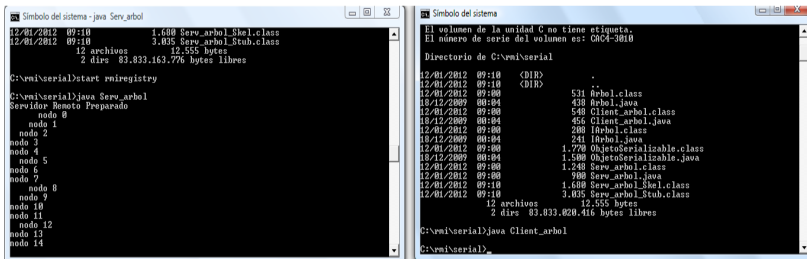
Código 6: codigos_t7/Client_arbol.java

```
1  /**
2   * @(#)Client_arbol.java
3   *
4   *
5   * @author Antonio Tomeu
6   * @version 1.00 2009/5/21
7   */
8
```

Serialización VII

```
9  import java.rmi.*;
10 import java.rmi.registry.*;
11
12 public class Client_arbol
13 {
14     public static void main(String[] args)
15         throws Exception
16     {
17         int niveles = 3;
18         Arbol arb = new Arbol (niveles);
19
20         IArbol RefObRemoto =
21             (IArbol)Naming.lookup("//localhost/Servidor");
22
23         RefObRemoto.Listado_Remoto (arb, niveles);
24
25     }
26 }
```


Serialización VIII



```
Símbolo del sistema - java Serv_arbol
12/01/2012 09:10 1.600 Serv_arbol_Skel.class
12/01/2012 09:10 3.035 Serv_arbol_Stub.class
                12 archivos 12.555 bytes
                2 dirs 83.833.163.776 bytes libres

C:\rmi\serial>start rmiregistry

C:\rmi\serial>java Serv_arbol
Servidor Remoto Preparado
nodo 0
nodo 1
nodo 2
nodo 3
nodo 4
nodo 5
nodo 6
nodo 7
nodo 8
nodo 9
nodo 10
nodo 11
nodo 12
nodo 13
nodo 14

Símbolo del sistema
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: CAC4-3010

Directorio de C:\rmi\serial

12/01/2012 09:10 <DIR>
12/01/2012 09:10 <DIR>
12/01/2012 09:00 531 Arbol.class
18/12/2009 00:04 438 Arbol.java
12/01/2012 09:00 548 Client_arbol.class
18/12/2009 00:04 456 Client_arbol.java
12/01/2012 09:00 288 IArbol.class
18/12/2009 00:04 241 IArbol.java
12/01/2012 09:00 1.770 ObjetoSerializable.class
18/12/2009 00:04 1.500 ObjetoSerializable.java
12/01/2012 09:00 1.248 Serv_arbol.class
12/01/2012 09:00 988 Serv_arbol.java
12/01/2012 09:10 1.600 Serv_arbol_Skel.class
12/01/2012 09:10 3.035 Serv_arbol_Stub.class
                12 archivos 12.555 bytes
                2 dirs 83.833.020.416 bytes libres

C:\rmi\serial>java Client_arbol

C:\rmi\serial>
```

Figura: Ejecución del Ejemplo en el Modo Local

- ▶ Descargue los ficheros contenidos en la subcarpeta Serialización (carpeta de códigos del tema), compile y pruebe
- ▶ Haga lo propio distribuyendo la aplicación con un compañero

Gestionando la Seguridad: policytool I

- ▶ Java tiene en cuenta la seguridad

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

- ▶ Se ajusta la seguridad
 - ▶ Construyendo un objeto `SecurityManager`
 - ▶ Llamando al método `setSecurityManager` (clase `System`)
- ▶ Clase `RMISecurityManager`
- ▶ Programador ajusta la seguridad mediante la clase `Policy`
 - ▶ `Policy.getPolicy()` permite conocer la seguridad actual.
 - ▶ `Policy.setPolicy()` permite fijar nueva política
 - ▶ Seguridad reside en fichero específico
- ▶ Java 2 incorpora una herramienta visual: `policytool`

Gestionando la Seguridad: policytool II

► Ajuste de la política de seguridad

► Crearlas con policytool: ejemplo de fichero .policy

```
1  /* AUTOMATICALLY GENERATED ON Fri May 28
2  19:23:13 CEST 2004*/
3  /* DO NOT EDIT */
4  grant {
5  permission java.net.SocketPermission
6  "*:1024-65535", "connect,accept, listen,
7  accept, connect, listen, resolve";
8  permission java.net.SocketPermission "*:80",
9  "connect, accept";
10 permission java.security.AllPermission;
11 };
```

► Activarlas

► Ejecutar ajustando la política con

```
1  java -Djava.security=fichero.policy servidor|cliente
```

► O bien crear objeto RMISecurityManager y ajustar SetSecurityManager sobre el objeto creado

- ▶ Si el servidor de RMI debe notificar eventos a clientes, la arquitectura es inapropiada
- ▶ La alternativa estándar es el sondeo (*polling*), donde cliente:

```
1  Iservidor RefRemota= (IServidor)
2      Naming.lookup (URL);
3  while (!(Ref.Remota.Evento.Esperado())){}
```

RMI con Polling: Ejemplo de Interfaz

Código 7: codigos_t7/ejemploPolling.java

```
1  /**
2   * @(#)ejemploPolling.java
3   * @author A.T.
4   * @version 1.00 2012/1/11
5   */
6
7  import java.rmi.*;
8  public interface ejemploPolling
9  extends Remote {
10     public void datoInc() throws RemoteException;
11     public boolean igualDiez() throws RemoteException;
12
13 }
```

RMI con Polling: Ejemplo de Servidor I

Código 8: codigos_t7/servPolling.java

```
1  /**
2   * @(#)servPolling.java
3   * @author A.T.
4   * @version 1.00 2012/1/11
5   */
6
7  import java.rmi.*;
8  import java.rmi.server.*;
9  import java.rmi.registry.*;
10
11 public class servPolling
12 extends UnicastRemoteObject
13 implements ejemploPolling {
14     private static int dato = 0;
15
16     public void datoInc() throws RemoteException {
17         dato++;
18         System.out.println(dato);
19     }
20 }
```

RMI con Polling: Ejemplo de Servidor II

```
20
21 public boolean igualDiez() throws RemoteException {
22     return (dato == 10);
23 }
24
25 public servPolling() throws RemoteException {}
26
27 public static void main(String[] args)
28     throws Exception {
29     servPolling contRemoto = new servPolling();
30     Naming.bind("Servidor_Polling", contRemoto);
31     System.out.println("Servidor Remoto Preparado");
32 }
33 }
```


RMI con Polling: Ejemplo de Cliente I

Código 9: codigos_t7/clientPolling.java

```
1  /**
2   * @(#)clientPolling.java
3   * @author A.T.
4   * @version 1.00 2012/1/11
5   */
6
7  import java.rmi.*;
8  import java.rmi.registry.*;
9
10 public class clientPolling {
11
12     public static void main(String[] args)
13     throws Exception {
14         ejemploPolling RefObRemoto = (ejemploPolling)
15             Naming.lookup("//localhost/Servidor_Polling");
16
17         while (!RefObRemoto.igualDiez()) {
18             System.out.println("Incremento remoto del contador");
19             RefObRemoto.datoInc();
20         }
21     }
22 }
```

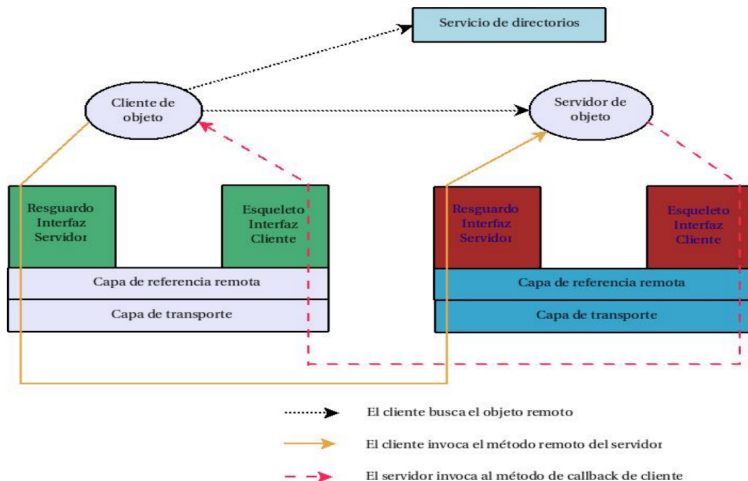
RMI con Polling: Ejemplo de Cliente II

```
19     }  
20     System.out.print("El contador remoto llego a diez y se  
        sale...");  
21  
22     }  
23 }
```

Características del Callback

- ▶ Clientes interesados **se registran** en un objeto servidor para que les sea notificado un evento
- ▶ Cuando el evento se produce, el objeto servidor notifica al cliente su ocurrencia
- ▶ ¿Qué necesitamos para que funcione?

Arquitectura RMI para Callback

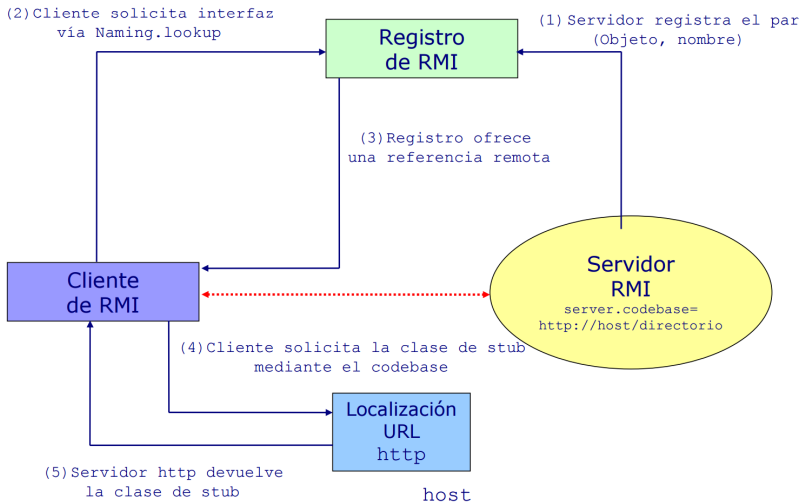


- ▶ Hay que duplicar la arquitectura
- ▶ Se requieren dos interfaces
- ▶ Habrá dos niveles de *stubs*
 - ▶ Cliente (*stub+skel*) y servidor (*stub+skel*)
- ▶ Es necesario proveer en el servidor medios para que los clientes registren sus peticiones de callback.

- ▶ Lea el guión de *callback* de cliente
- ▶ Compile y active el código ejemplo de *callback* que figura en la subcarpeta *callback* de la carpeta de códigos RMI (utilice *stubs* y *skeletons*).
- ▶ Haga lo propio con subcarpeta *callback_simple*
- ▶ Distribuya la aplicación con su compañero de grupo

- ▶ Permite cambios libres en el servidor
- ▶ Elimina la necesidad de distribuir (resguardos) si los hay y nuevas clases
- ▶ Dinámicamente, el cliente descarga todas las clases que necesita para desarrollar una RMI válida
- ▶ Mediante HTTP o incluso FTP

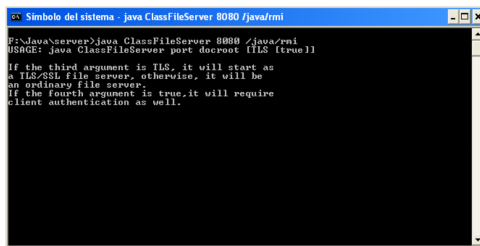
Descarga Dinámica de Clases II



- ▶ Mini-servidor de HTTP de Sun
 - ▶ Clases ClassServer y ClassFileServer
 - ▶ Se compilan conjuntamente

```
java ClassFileServer 8080 /rmi/clases
```
- ▶ Servidor HTTP estándar
 - ▶ Apache
 - ▶ Otros

Servidor HTTP para Descarga Dinámica II



```
Simbolo del sistema - java ClassFileServer 8080 /java/rmi
F:\Java\server>java ClassFileServer 8080 /java/rmi
USAGE: java ClassFileServer port docroot [TLS {true}]

If the third argument is TLS, it will start as
a TLS/SSL file server, otherwise, it will be
an ordinary file server.
If the fourth argument is true, it will require
client authentication as well.
```

Figura: Ejemplo de servidor básico de sun activo en Windows

- ▶ Colocar resguardos y clases para carga dinámica en /java/rmi
- ▶ Ajustar la propiedad `java.rmi.server.codebase`

Novedades a partir de JDK 1.5

- ▶ Soporta Generación Dinámica de Resguardos
- ▶ Prescinde del generador `rmic`
- ▶ Pruebe ahora los ejemplos anteriores sin pasar por la etapa de generación de resguardos (`rmic`). ¿Qué observa?

Bibliografía



Eckel, B.

Thinking in Java

Prentice Hall, 2006



Hilderink et al.

Communicating Threads for Java

Draft



Vinoski, S.

CORBA: Integrating Diverse...

IEEE Communications, 1997



Grosso, W.

Java RMI

O'Reilly, 2001