COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

INTRODUCCIÓN

Objetos Remotos:

- Reciben invocaciones remotas de métodos
- Fallas Independientes → semánticas distintas
- Transparencia Total → no necesariamente deseable
- Empaquetado y Desempaquetado de argumentos (marshalling - unmarshalling)

COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

HISTORIA

- RPC
- RMI
- Basado en eventos

COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

MODELO

- Transparencia de ubicación
- Protocolos de Comunicación (*TCP - UDP*)
- HW → *empaquetado y desempaquetado de datos*
- Sistema Operativo → *independiente*
- Varios Lenguajes de Programación

RMI

COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

INTERFACES

- Especifica funciones/métodos accecibles
- Parámetros → por Valor
  → por Referencia ?
  → ~~Punteros~~

- **Interface Remota**

  Especifica los métodos de un objeto accecibles por otro objeto.

COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

IDL (Interface Definition Languaje)

- Objetos Remotos en Diferentes Lenguajes

Ejemplo Corba IDL:

```
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

COMUNICACIÓN ENTRE OBJETOS DISTRIBUIDOS

MODELO OBJETOS

OBJETOS DISTRIBUIDOS

- Problema partido en objetos → extensión natural
- Clinete-Servidor

MODELO OBJETOS DISTRIBUIDOS

- Invocación remota de métodos
- Objetos remotos → remota
  → local
- Referencia remota a objetos
- Interface Remota

MODELO
OBJETOS DISTRIBUIDOS

DECISIONES DE DISEÑO

- **Semántica**

  - Retransmitir pedidos
  - Filtrar duplicados
  - Retransmitir resultados

  * local es única → se ejecuta sólo una vez

SEMÁNTICAS (1/3)

Tal Vez (*Maybe*)

* El método Remoto se ejecuta una o ninguna vez.
  - Mensage de pedido o respuesta perdida
  - Caida del servidor
* Se utiliza cuando el sistema tiene fallas ocasionales

SEMÁNTICAS (2/3)

Al menos una Vez (*At-least-once*)

* El método Remoto se ejecuta al menos una vez
* Se retransmiten pedidos
* El cliente recibe un resultado o una excepción
  - Caida del servidor
  - Fallas arbitrarias (+ de 1 ejecución)

* Se utiliza cuando los métodos son idempotentes

SEMÁNTICAS (3/3)

A lo sumo una Vez (*At-most-once*)

* El cliente recibe un resultado (el método se ejecutó exactamente una vez) o una excepción (?)

* Se logra utilizando todas las medidas de tolerancia a fallos disponibles.

* Java RMI

Java RMI

RMI ("Remote Method Invocation") significa comunicar el objeto a través de la red.

RMI es un tipo de estructura o sistema que permite a un objeto que se ejecuta en una máquina virtual Java (Cliente) invocar métodos en un objeto que se ejecuta en otra máquina virtual Java (Servidor). Este objeto se denomina Objeto Remoto y tal sistema es También llamada RMI Distributed Application.

RMI proporciona la comunicación remota entre los programas escritos en el JAVA

Java RMI

Capas

1.- Application

2.- Proxy

3.- Referencia Remota

4.- Transporte

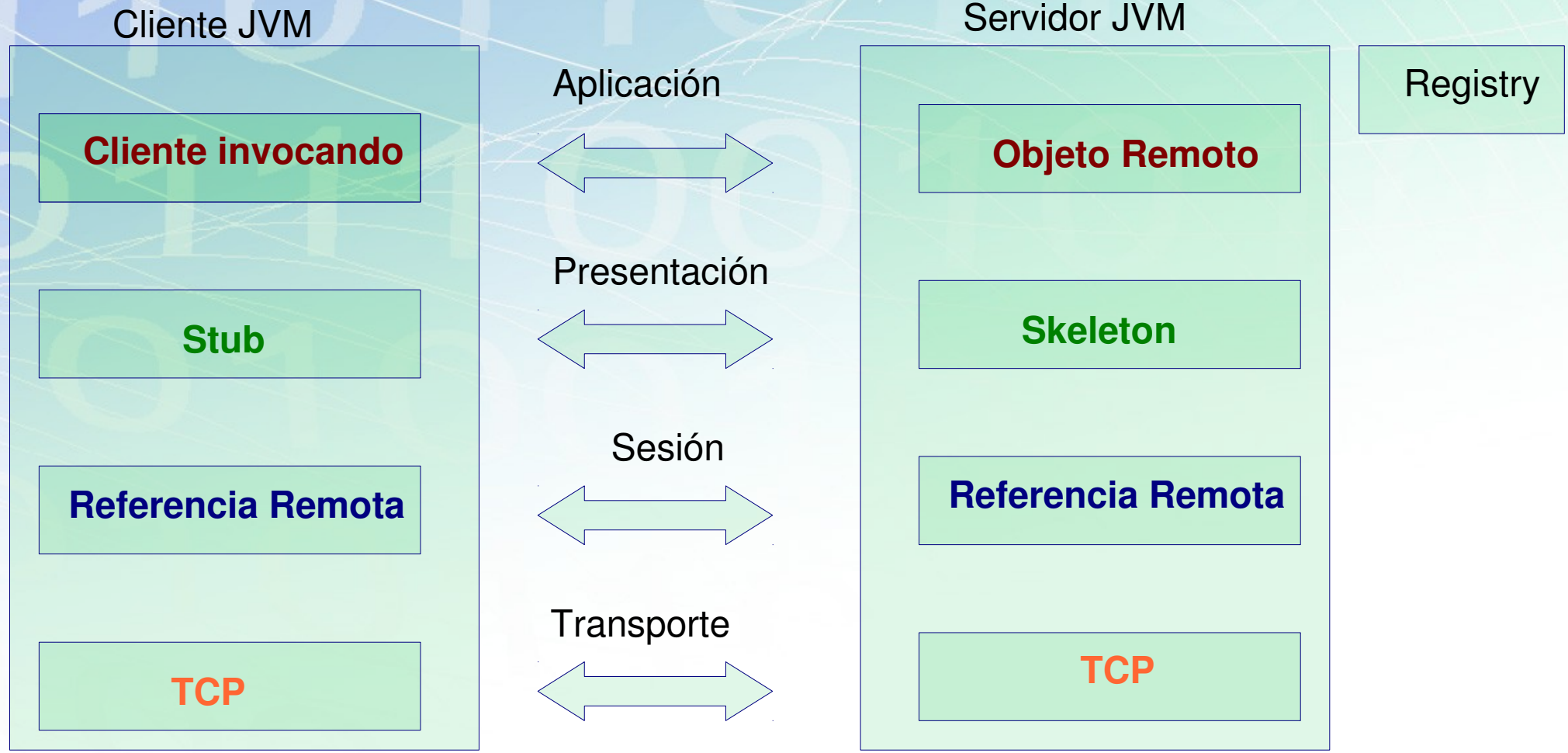
Componentes

1.- RMI Server

2.- RMI Client

3.- RMI Registry

Java RMI


Cliente JVM
Cliente invocando
Stub
Referencia Remota
TCP
Aplicación
Presentación
Sesión
Transporte
Servidor JVM
Objeto Remoto
Skeleton
Referencia Remota
TCP
Registry

Capas RMI

1.- Capa de aplicación:

2.- Capa Proxy ("Stub / Skeleton"):
Una clase Stub es un proxy del lado del cliente que maneja los objetos remotos que se obtienen de la referencia.
Una clase Skeleton es un proxy del lado del servidor que establece la referencia a los objetos que se comunican con el Stub.

3.- Capa de referencia remota (RRL):
Es un responsable de gestionar las referencias hechas por el cliente al objeto remoto en el servidor para que esté disponible en ambas JVM (cliente y servidor).
El RRL del lado del cliente recibe la solicitud de métodos del Stub que se transfiere al proceso de flujo de bytes denominado serialización (Marshaling) y luego estos datos se envían al RRL del lado del servidor.
El lado del servidor RRL haciendo proceso inverso y convertir los datos binarios en objeto. Este proceso se llama deserialización o desmaralización y luego se envía a la clase Skeleton.

4.- Capa de transporte:

Componentes RMI

1.- Servidor RMI:
Crea objetos remotos y aplica la referencia a estos objetos en el Registro.

2.- Cliente de RMI:
El Cliente RMI obtiene la referencia de uno o más objetos remotos desde el Registro con la ayuda del nombre del objeto.

3.- Registro de RMI:
En el lado del servidor se aplica la referencia del objeto (que se invoca remotamente).

RMI Registry

Es un servicio de nombres.

Los programas servidores RMI utilizan este servicio para enlazar el objeto java remoto con los nombres.

Los clientes que se ejecutan en máquinas locales o remotas recuperan los objetos remotos por su nombre registrado en el registro de RMI y luego ejecutan métodos en los objetos.

Para trabajar con el registro de RMI …

1. Iniciar el Registro
2. Ejecutar el programa del servidor.
3. Ejecutar el programa cliente.

RMI – Desarrollo (1/5)

(1) Definir la interfaz remota

(2) Definir la clase e implementar la interfaz remota (métodos) en esta clase

(3) Definir la clase del lado del servidor

(4) Definir la clase del lado del cliente

(5) Compile los cuatro archivos fuente (java)

(6) Iniciar el registro remoto RMI

(7) Ejecute la clase del lado del servidor

(8) Ejecute la clase de cliente (en otra JVM)

RMI – Desarrollo (2/5)

(1) Definir la interfaz remota

Esta es una interfaz en la que declaramos los métodos según nuestra lógica y más adelante estos métodos serán llamados usando RMI.

Calculadora.java

```
// interface que contiene los métodos del servicio

import java.rmi.*;


public interface Calculadora extends Remote {
    public int suma (int a, int b) throws RemoteException;
    public int resta (int a, int b) throws RemoteException;
    public int div (int a, int b) throws RemoteException;
    public int mul (int a, int b) throws RemoteException;

}
```

RMI – Desarrollo (3/5)

(2) Definir la clase e implementar la interfaz remota (métodos) en esta clase:

CalculadoraImp.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CalculadoraImp extends UnicastRemoteObject implements Calculadora
{
    protected CalculadoraImp() throws RemoteException    { super(); }
    public int suma(int a, int b) throws RemoteException     { return a+b; }
    public int resta(int a, int b) throws RemoteException      { return a-b; }
    public int div(int a, int b) throws RemoteException     { return a/b; }
    public int mul(int a, int b) throws RemoteException    { return a*b; }
   private static final long serialVersionUID = 75264722956227761 47L;
}
```

RMI – Desarrollo (4/5)

(3) Definir la clase del lado del servidor:

Servidor.java

```
import java.rmi.*;
class Servidor {
  static public void main(String args[]) {
        if (args.length!=1) { System.err.println("Uso: Servidor Puerto"); return; }
         if (System.getSecurityManager() == null) {
           // System.setSecurityManager(new RMISecurityManager());
           System.setProperty("java.rmi.server.hostname","10.0.0.199");}
      try {
         Calculadora calc = new CalculadoraImp();
         Naming.rebind("rmi://localhost:" + args[0] + "/CalculadoraImp", calc);  }
      catch (RemoteException e) {
         System.err.println("Error de comunicacion: " + e.toString());
         System.exit(1); }
      catch (Exception e) {
         System.err.println("Excepcion en Servidor:");
         e.printStackTrace();
         System.exit(1); }
      } }
```

RMI – Desarrollo (5/5)

(4) Definir la clase del lado del cliente

Cliente.java

```
iimport java.rmi.*;

public class Cliente {
    public Cliente()
    {
        try {
      Calculadora calc = (Calculadora)Naming.lookup
("//10.0.0.199:50000/CalculadoraImp");
           System.out.print ("2 + 3 = "); System.out.println (calc.suma(2,3));
          System.out.print ("7 - 3 = "); System.out.println (calc.resta(7,3));
          System.out.print ("9 / 3 = "); System.out.println (calc.div(9,3));
          System.out.print ("2 * 3 = "); System.out.println (calc.mul(2,3));    }
        catch (Exception e)  {
           e.printStackTrace();   }
    }
    public static void main(String[] args) {
     new Cliente();
    }
}
```

Serialización en Java RMI

La serialización de Java es el proceso de convertir un objeto en un flujo de bytes. Representa la clase del objeto, la versión del objeto y el estado interno del objeto. La “des-serealización” es el proceso de reconstrucción del conjunto de bytes en un objeto.

USOS

- **Stashing**
  En lugar de tener un objeto grande en la memoria, es mejor que la cacheé a un archivo local a través de serialización
- **Transmisión**
  Para serializar un objeto a través de una red mediante RMI (Remote Method Invocation)
- **Persistencia**
  Si desea conservar el estado de una operación en particular en una base de datos, simplemente serializarla a una matriz de bytes y guardarla en la base de datos para su uso posterior.
- **Clonado Profundo**
  Serializar el objeto en una matriz de bytes y luego deserializarlo a otro objeto cumplirá el propósito.
- **Cruzar la comunicación JVM**
  La serialización funciona de la misma manera en diferentes JVMs independientemente de las arquitecturas en las que se estén ejecutando.

serialVersionUID

El serialVersionUID en las clases Java nos previene de errores en la "deserialización"

Es posible que en distintas versiones de nuestro programa la clase objeto_serializable cambie, de forma que es posible que un lado tenga una versión más antigua que en el otro lado. Si sucede esto, la reconstrucción de la clase en el lado que recibe es imposible.

¿Cómo generar un <SerialVersionUID>?

- **Utilizar el comando <serialver>**
- **Usar Eclipse IDE**
- **Asignar un valor a gusto**

Empaquetado y Desempaquetado

Durante la comunicación entre dos máquinas a través de RMI, los parámetros se envasan en un mensaje y luego se envían a través de la red.

Este empaquetamiento de parámetros en un mensaje se llama ordenación (Marshalling).

En el otro lado estos parámetros embalados se desempaquetan del mensaje (Unmarshalling).