PUCP

PROGRAMACIÓN 3 (1INF30)

RMI

2024

Prof. Freddy Paz

AGENDA

1. Concepto de RMI
2. Arquitectura de RMI
3. Implementación de RMI
4. Referencias

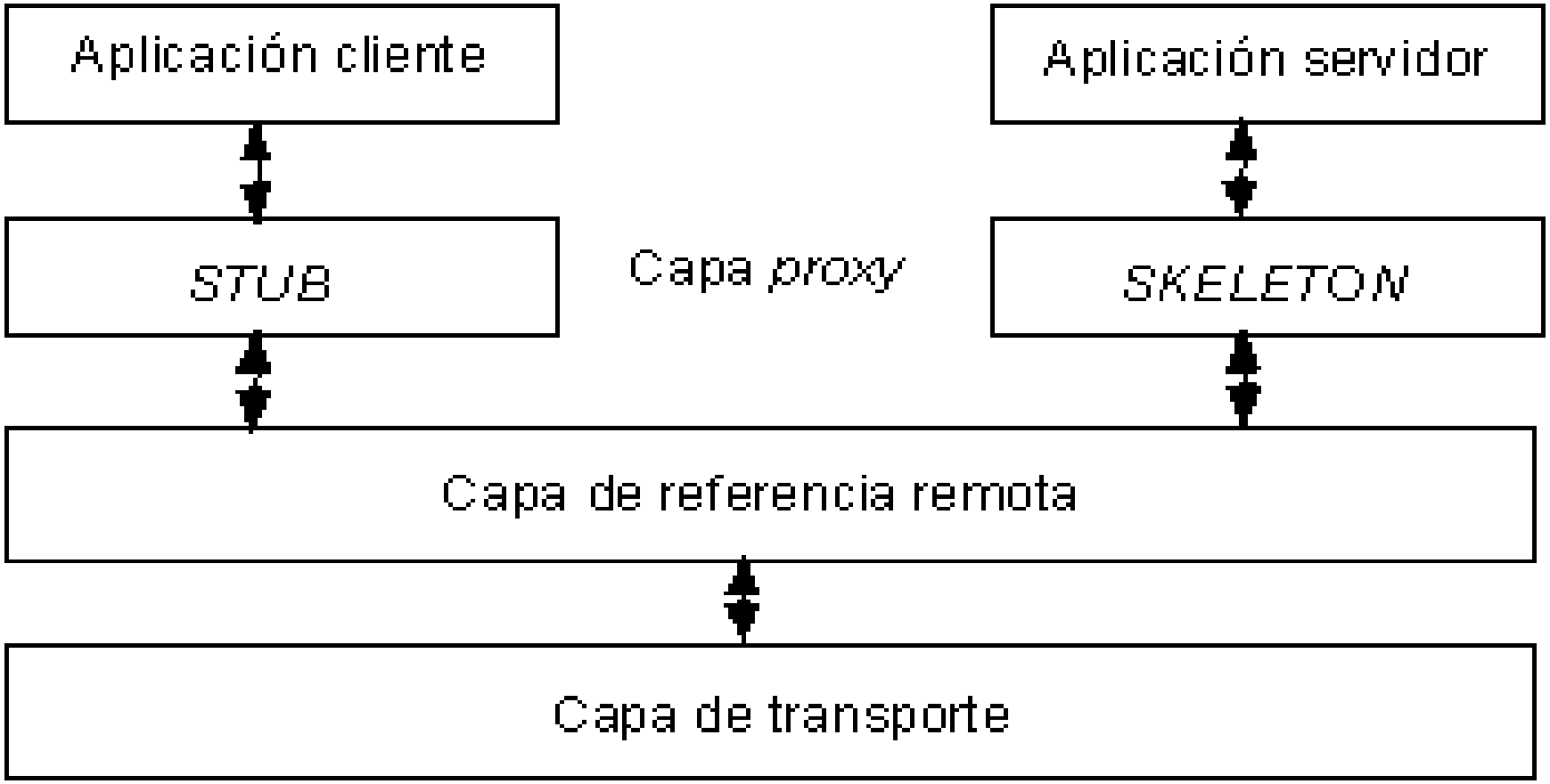
Resultado de Aprendizaje

- **RA1: Desarrolla aplicaciones informáticas empleando conocimientos de programación orientada a objetos.**
- **RA2: Aplica principios de concurrencia y programación distribuida para desarrollar soluciones informáticas.**
- **RA3: Despliega aplicaciones de software que se conectan a diferentes motores de base de datos y a las cuales puedan acceder los usuarios finales.**

Concepto de RMI

- **RMI** (**Remote Method Invocation**) es un mecanismo que permite realizar llamadas a métodos de objetos remotos situados en distintas (o la misma) máquinas virtuales de Java, compartiendo así recursos y carga de procesamiento a través de varios sistemas.

Arquitectura de RMI



Capa de Aplicación

- Implementación real de las aplicaciones cliente y servidor.
- Llamadas a alto nivel para acceder y exportar objetos remotos.
- Se declaran métodos en una interfaz que herede de **java.rmi.Remote**.
- Una vez que los métodos han sido implementados, el objeto debe ser exportado:
  - El objeto debe heredar de la clase **UnicastRemoteObject** (paquete java.rmi.server)

Capa de Proxy o 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.
- Se crea un objeto "stub" en el lado del cliente que representa el objeto remoto. El stub es responsable de enviar las invocaciones de métodos del cliente al objeto remoto a través de la red.
- Se crea un objeto "skeleton" en el lado del servidor que recibe las invocaciones de métodos del "stub" y las pasa al objeto remoto.

Capa de Referencia Remota

- 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.
- Maneja la comunicación y la invocación de métodos entre el cliente y el servidor.

Capa 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 entendido por programas Java.
- Maneja la conexión de red y la comunicación de bajo nivel entre el cliente y el servidor.
- Utiliza TCP/IP para la comunicación entre los diferentes componentes.

Crear una aplicación con RMI

- Toda aplicación RMI normalmente se descompone en 2 partes:
- **Un servidor**, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- **Un cliente**, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

Implementar la Aplicación Distribuida

- Se decide la arquitectura de la aplicación y se determina qué componentes son objetos locales y cuáles deberían ser accesibles remotamente.
- Este paso incluye:
  - **1.** Definir las interfaces remotas.
  - **2.** Implementar los objetos remotos.
  - **3.** Implementar el servidor.
  - **4.** Implementar el cliente.

1. Definir las interfaces remotas

- La interfaz debe ser pública.
- Debe heredar de la interfaz **java.rmi.Remote**, para indicar que puede llamarse desde cualquier Máquina Virtual Java.
- Cada método remoto debe lanzar la excepción **java.rmi.RemoteException**, además de las excepciones que pueda manejar.

1. Definir las interfaces remotas

```
public interface InterfazRemota extends java.rmi.Remote {

    public void metodo1()
    throws java.rmi.RemoteException;

    public String metodo2(String nombre) throws
    java.rmi.RemoteException;

}
```

2. Implementar los objetos remotos

```
public class ClaseRemota
   extends java.rmi.server.UnicastRemoteObject implements InterfazRemota{
   public ClaseRemota() throws java.rmi.RemoteException {
   }

   public void metodo1() throws java.rmi.RemoteException{
       System.out.println("Método 1");
   }

   public String metodo2(String nombre) throws java.rmi.RemoteException{
       System.out.println("Método 2");
       return "Hola " + nombre;
   }
   public void metodo3(){
       System.out.println("Este metodo no puede llamarse remotamente");
   }
}
```

3. Implementar el servidor

```
public class Principal{
        public static void main(String[] args){
                try{
                        System.setProperty("java.net.preferIPv4Stack" , "true");
                        System.setProperty("java.rmi.server.hostname", "127.0.0.1");
                        String puerto = "1234";
                        LocateRegistry.createRegistry(1234);
                InterfazRemota ir = new ClaseRemota();
java.rmi.Naming.rebind("//"+java.net.InetAddress.getLocalHost().getHostAddress()+ ":" +
puerto + "/rmi", ir);
} catch (Exception e){
          System.out.println(e.getMessage());
          }
  }
}
```

4. Implementar el cliente

```
public class ClienteRMI {
	public static void main(String[] args) {
		try {
	InterfazRemota ir = (InterfazRemota) java.rmi.Naming.lookup("//" +
args[0] + ":" + args[1] + "/rmi");
	//Se invoca al método 1 y al método 2
	ir.metodo1();
	System.out.println(ir.metodo2("Karla Perez"));
	} catch (Exception e) {
		e.printStackTrace();
		}
	}
}
```

Referencias

- https://docs.oracle.com/javase/tutorial/rmi/
- https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html
- https://www.oracle.com/technetwork/java/rmi-141556.html

¡Gracias!

PUCP