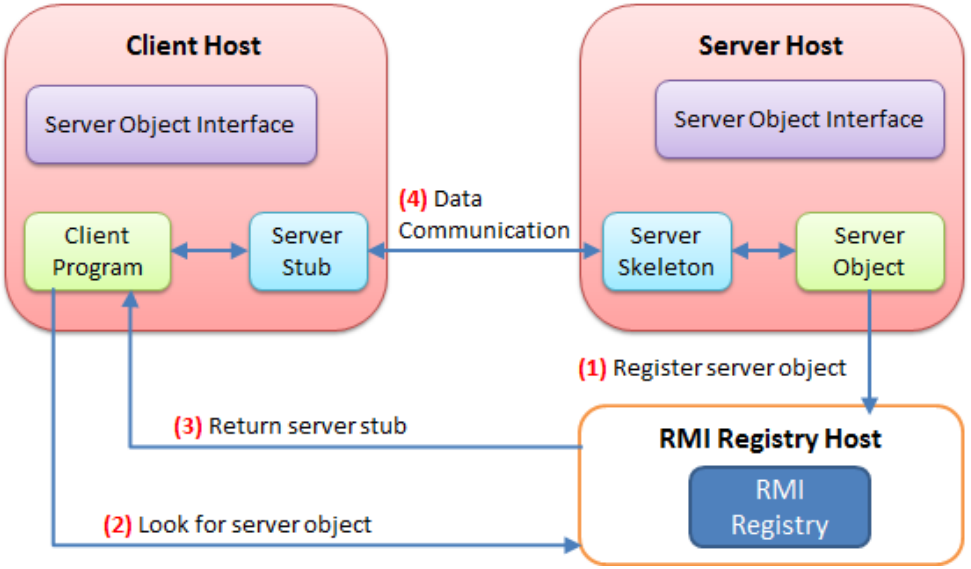
ESPAMMFL
ESCUELA SUPERIOR POLITÉCNICA AGROPECUARIA DE MANABÍ MANUEL FÉLIX LÓPEZ

Coordinación General de POSGRADO

MTI
Maestría en Tecnologías de la Información

RMI
Remote method invocation



René Guamán-Quinche
@rene5254
rene525456@gmail.com

Llamada a métodos remotos

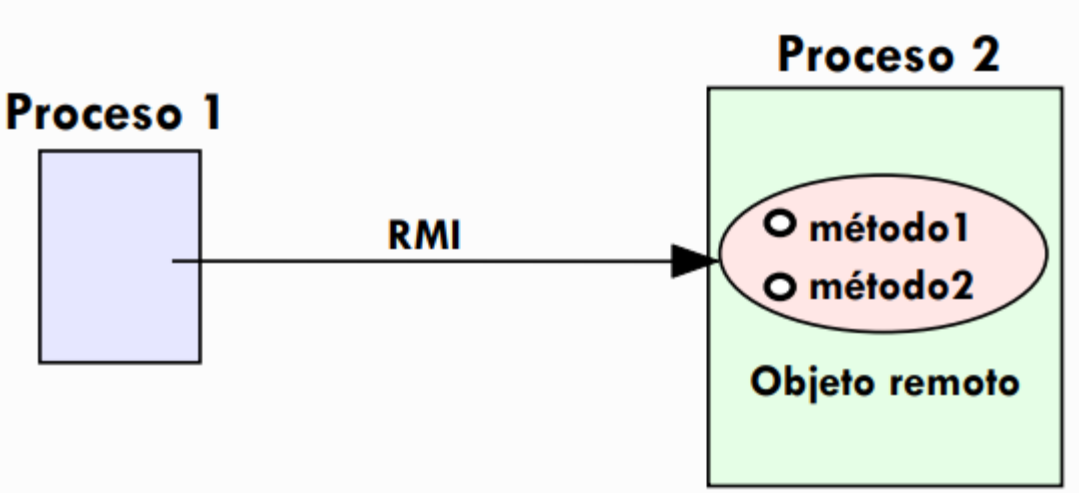
- Primera aproximación al uso de un modelo orientado a objetos sobre aplicaciones distribuidas
- Objetos distribuidos dentro de una red
  - Los objetos proporcionan métodos
  - los cuales dan acceso a los servicios

ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Llamada a métodos remotos

- Proceso invoca un método local de otro proceso
- Se envían tanto los argumentos del método como el valor devuelto por el mismo



ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Objetos distribuidos

Son los objetos que dinámicamente asumen el papel de clientes o servidores, según la necesidad

Un **objeto distribuido** es un objeto que puede ser accedido remotamente desde cualquier lugar en la red, del mismo modo que se haría si estuviese en la misma máquina

ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Objetos distribuidos

Los **objetos distribuidos** estarán "unidos" mediante algún mecanismo que permita saber a los clientes:

- ¿Dónde se encuentran los servidores?
- ¿Cómo acceder a ellos y ?
- ¿Qué se les puede pedir?

Objetos distribuidos

Es transparente

Se localiza los **objetos** en los sistemas remotos y transforma las peticiones para que se entiendan independientemente a la máquina sobre la que se están ejecutando o en el lenguaje en el que están escritos

Objetos distribuidos

El concepto global de **objetos distribuidos** puede verse como una red global de clientes y servidores heterogéneos

Tecnología

El mecanismo más usado en el modelo procedimental es la llamada a procedimiento remoto (Remote procedure call, RPC) que es idéntico a una llamada a un procedimiento sólo que origen y destino son procesos distintos.

Tecnología

¿Y qué es RMI?

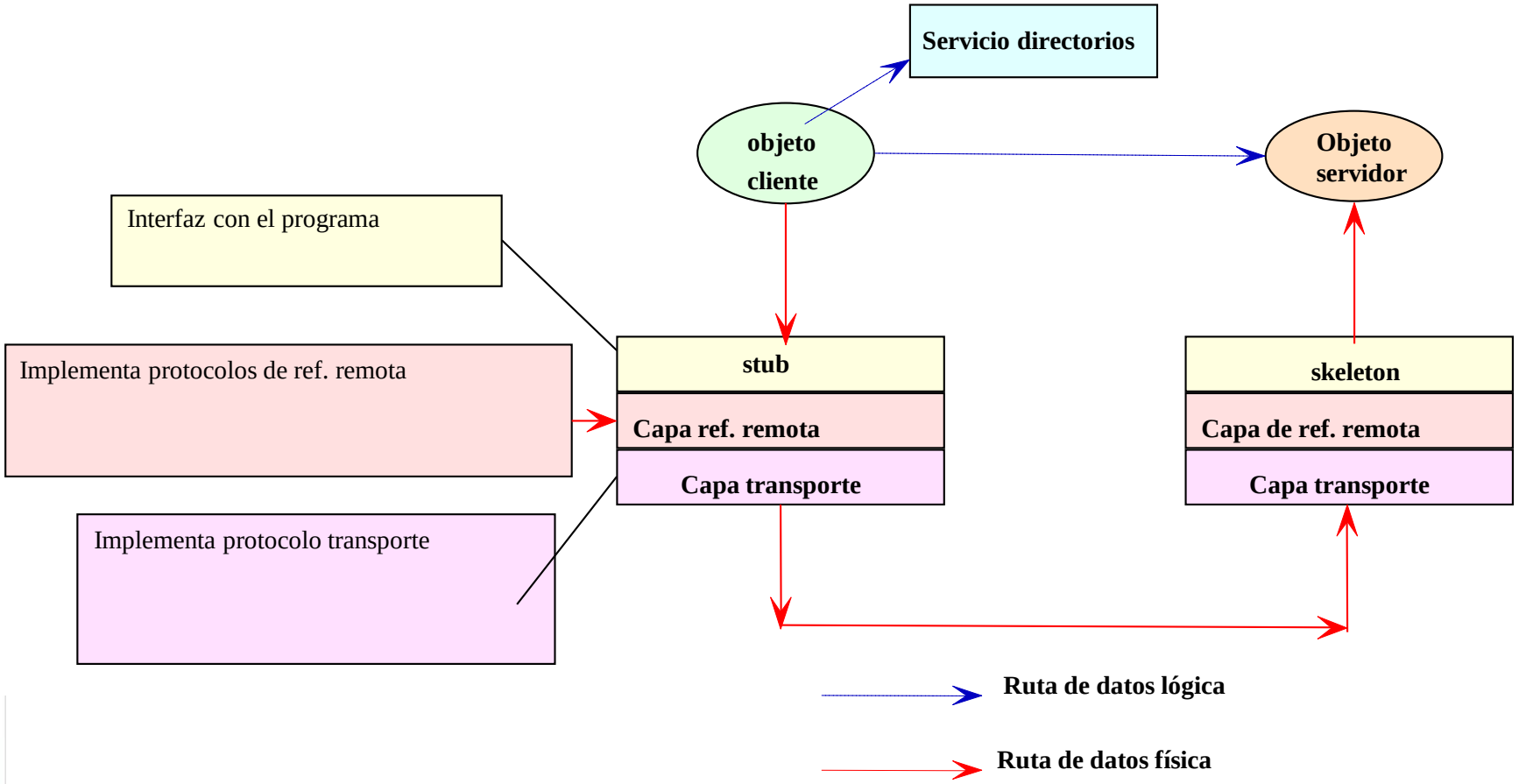
- **RMI** permite la invocación de métodos de objetos que residen en otras máquinas.
- El *Remote Method Invocation* aparece como parte integrante del *JDK* (*Java Development Kit*) de *Java* a partir de la versión 1.1
- En python se usa el paquete PYRO

Tecnología

Actualmente, algunas de las tendencias principales que se están utilizando dando soporte a la distribución de objetos son:

- ***RMI*** de *Sunsoft*
- ***CORBA*** del *Object Management Group*
- ***DCOM*** (*Distributed Component Object Model*) de *Microsoft*

Arquitectura RMI



ESPAMMFL

MTI
Maestría en Tecnologías de la Información

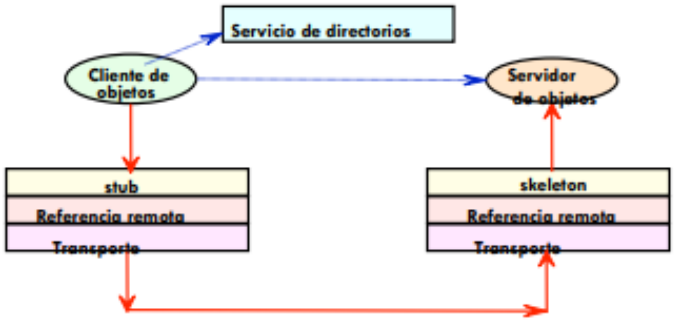
Arquitectura de RMI

Nivel de resguardo o stub

- Se encarga del aplanamiento de los parámetros.
- ***Stub***: resguardo local. Cuando un cliente realiza una invocación remota, en realidad hace una invocación de un método del resguardo local.

Nivel de gestión de referencias remotas

- Interpreta y gestiona las referencias a objetos remotos.
- Invoca operaciones de la capa de transporte.



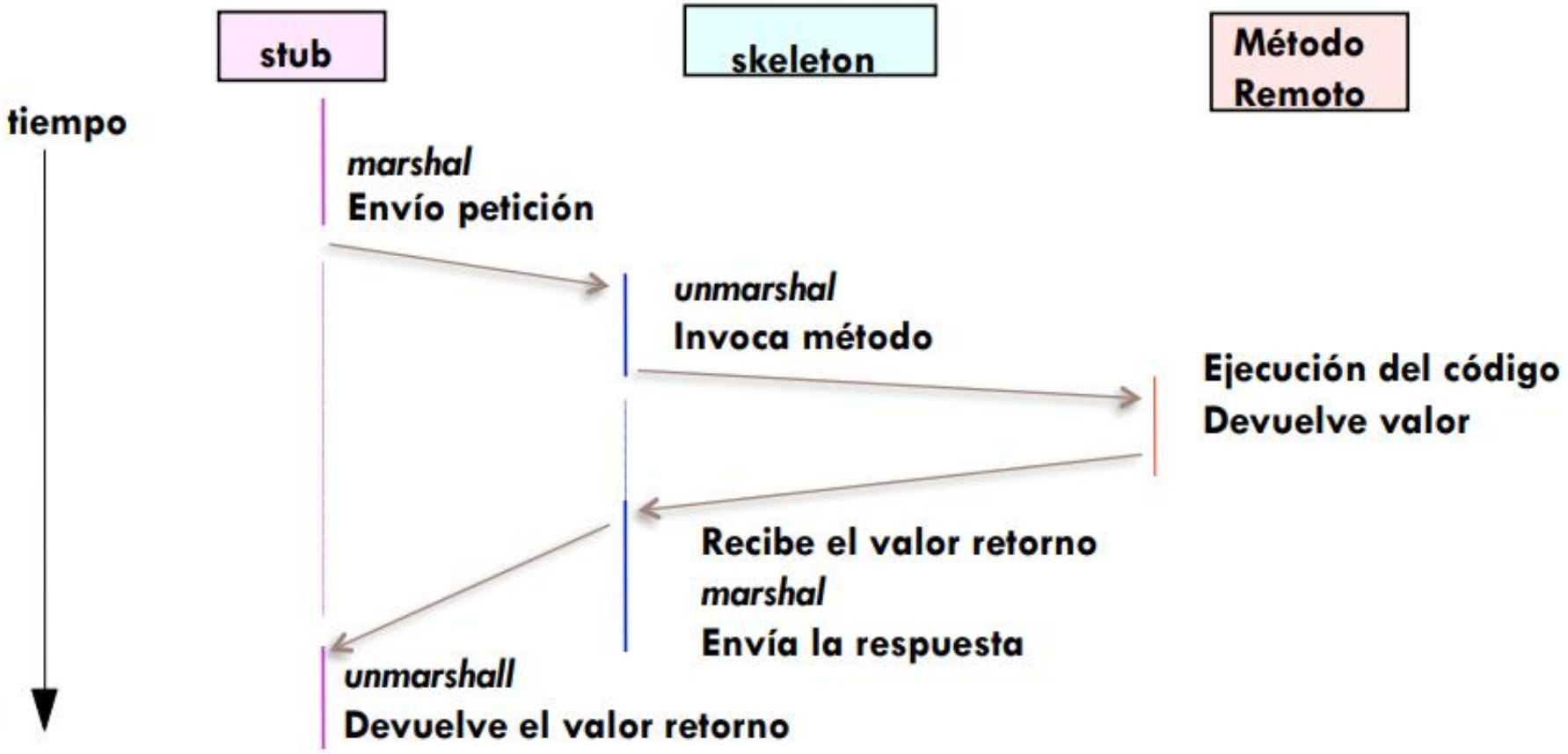
Nivel de transporte

- Se encarga de las comunicaciones y de establecer las conexiones necesarias.
- Basada en protocolo TCP.

ESPAMMFL

MTI
Maestría en Tecnologías de la Información

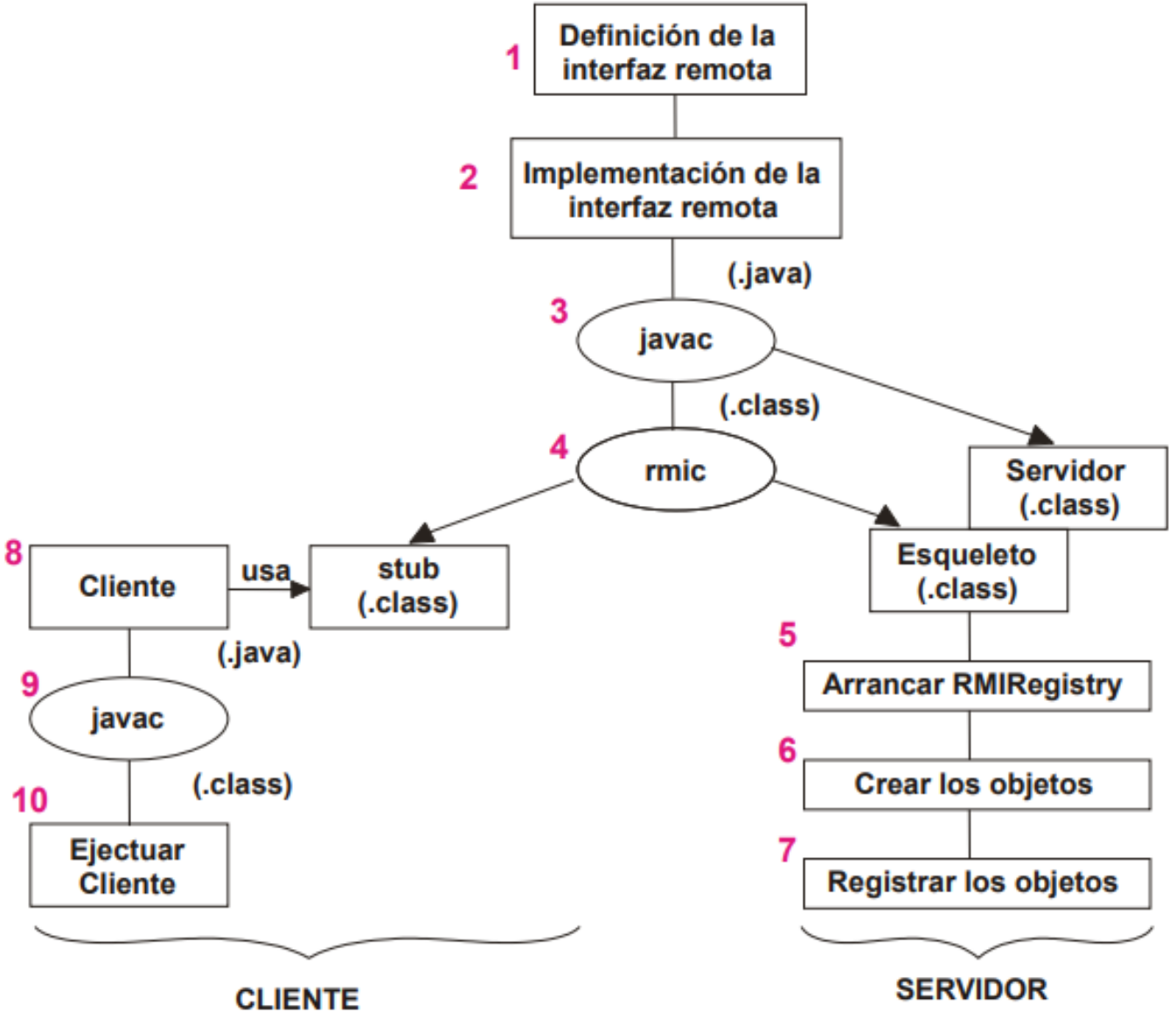
Arquitectura de RMI



Comparaciones RMI y Socket

- Ventajas:
  - Los programas RMI son más sencillos de diseñar
  - Servidor RMI concurrente
- Inconvenientes:
  - Sockets tienen menos sobrecarga
  - RMI sólo para plataformas Java, python.

Diseño RMI


1
Definición de la interfaz remota
2
Implementación de la interfaz remota
(.java)
3
javac
(.class)
4
rmic
Servidor (.class)
Esqueleto (.class)
stub (.class)
8
Cliente
usa
(.java)
9
javac
(.class)
10
Ejectuar Cliente
5
Arrancar RMIRegistry
6
Crear los objetos
7
Registrar los objetos
CLIENTE
SERVIDOR

Ejemplo de RMI

Se realizará una aplicación para enviar un mensaje de saludo al servidor. El servidor recibirá el mensaje y lo imprimirá

Ejemplo de RMI

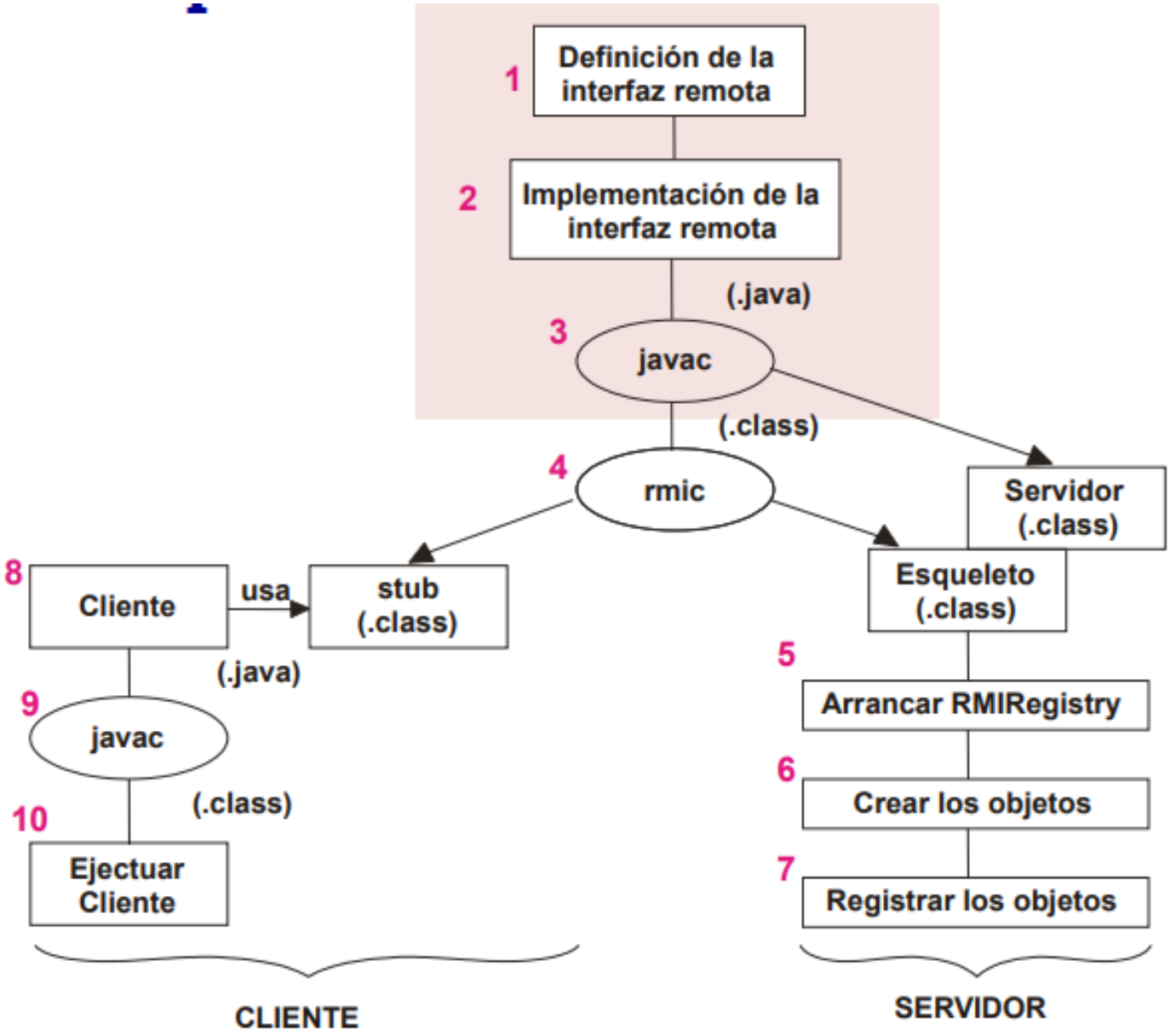
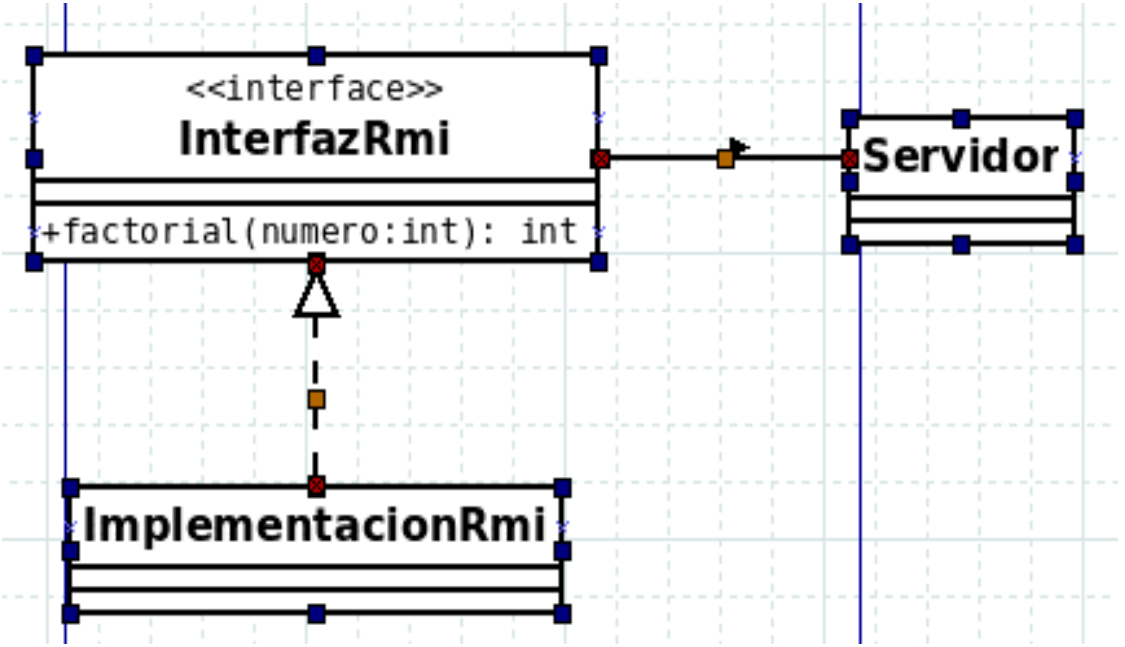


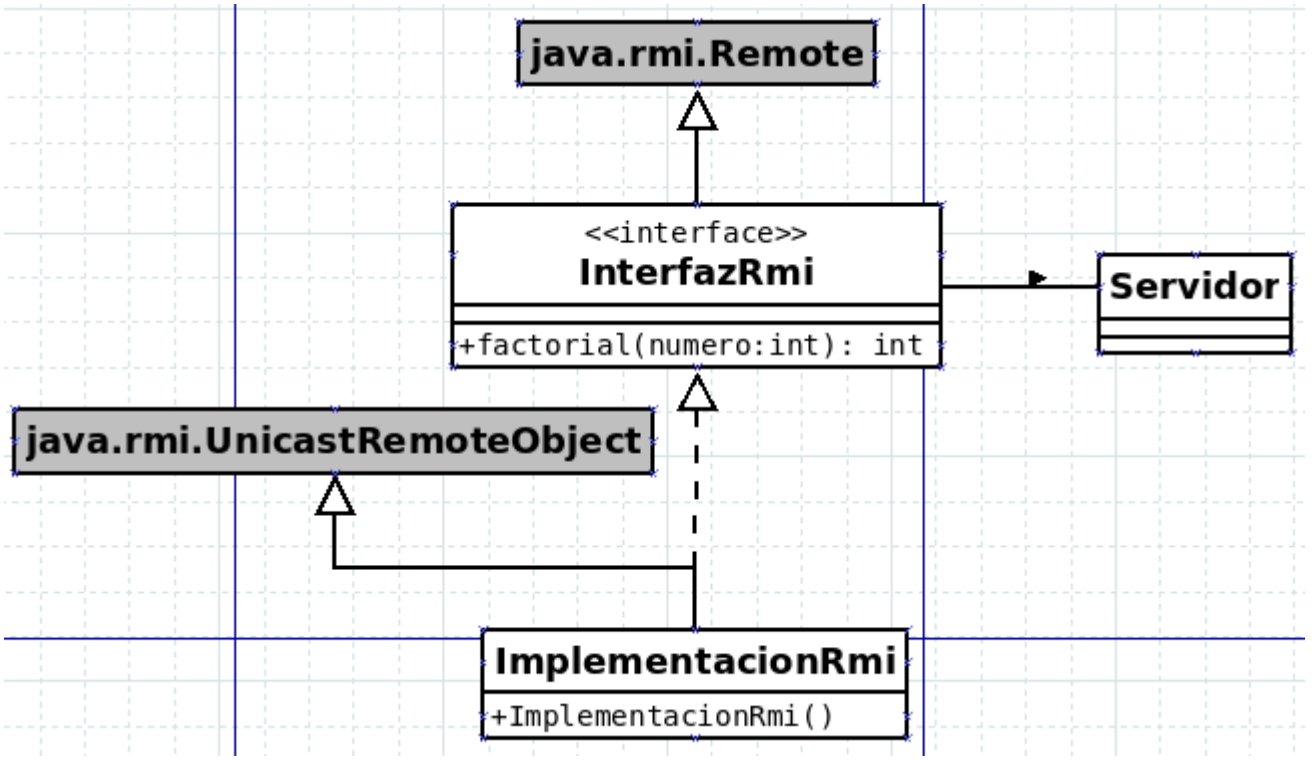
Diagrama de arquitectura de RMI básico



ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Diagrama de arquitectura de RMI básico



ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Objetos Remotos

Remote: se envia una referencia al objeto en otro computador(algo así como un paso por referencia). Los cambios que se hacen se reflejan en la instancia final.

Serializable: se envia una copia exacta del objeto (algo así como un paso por valor). Los cambios que se hacen en otro computador no se reflejan en el objeto original (por que es una copia)

UnicastRemoteObject

Una vez que los métodos han sido implementados (comportamiento remoto con Remote o Serializable), 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

Ejemplo RMI

InterfazRmi:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface InterfazRmi extends Remote {
    public String saludar(String nombre) throws RemoteException;
}
```

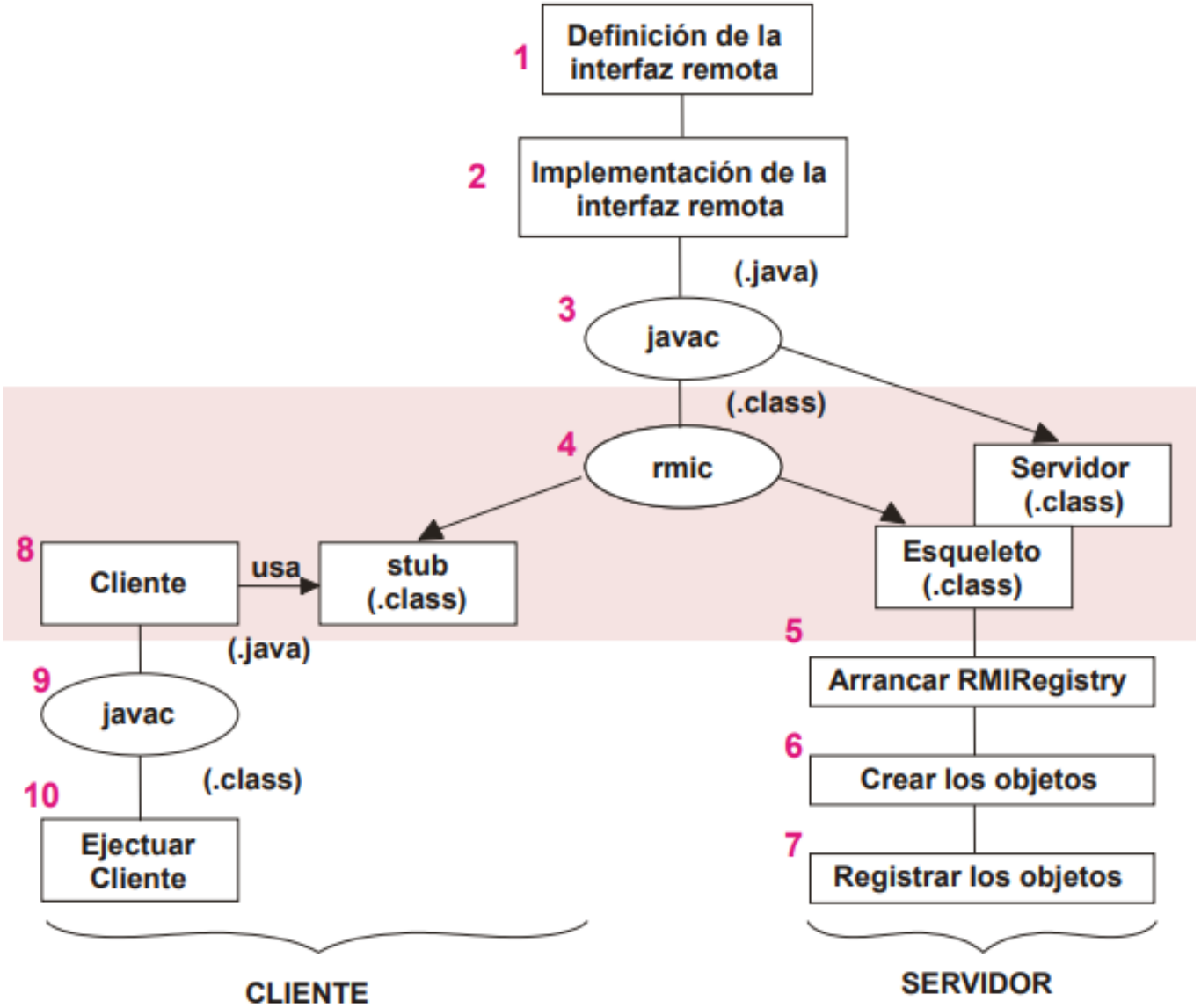
Ejemplo RMI

ImplementacionRmi:

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ImplementacionRmi extends UnicastRemoteObject
    implements InterfazRmi{

   public ImplementacionRmi() throws RemoteException {
      super();
   }

   public String saludar(String nombre) throws RemoteException {
      return "hola "+ nombre;
   }
}
```

MTI
Maestría en Tecnologías de la Información

Ejemplo RMI

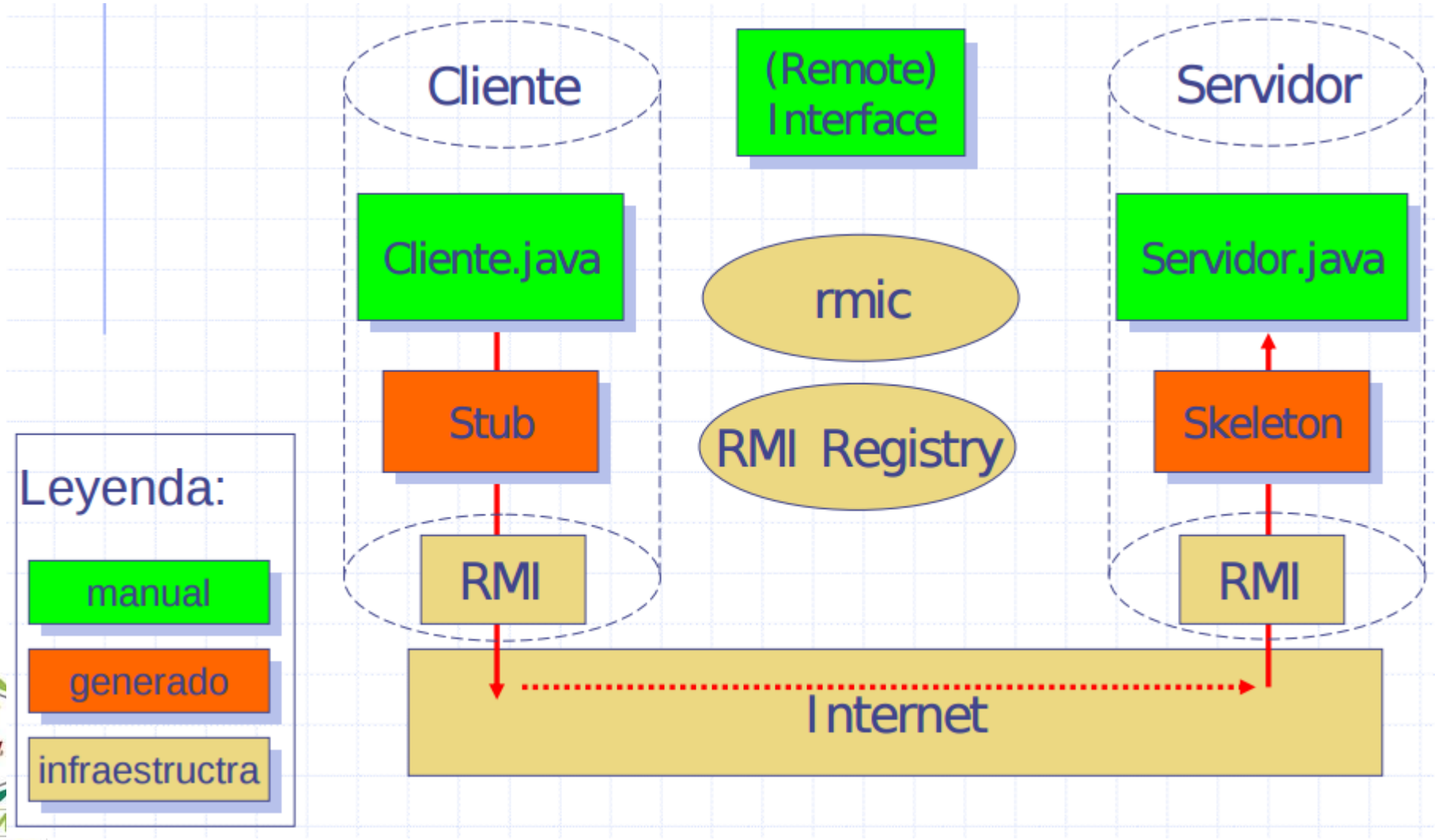

1
Definición de la interfaz remota
2
Implementación de la interfaz remota
(.java)
3
javac
(.class)
4
rmic
Servidor (.class)
Esqueleto (.class)
8
Cliente
usa
stub (.class)
(.java)
9
javac
(.class)
10
Ejectuar Cliente
5
Arrancar RMIRegistry
6
Crear los objetos
7
Registrar los objetos
CLIENTE
SERVIDOR

Ejemplo RMI

Servidor:

```
import java.rmi.Naming;

public class Servidor {

    public Servidor(){
        try{
            System.out.println("Estamos  en  el servidor");
            InterfazRmi servidor = new ImplementacionRmi();
            Naming.rebind("rmi://localhost/oyente", servidor);
        }catch(Exception ex){        }
    }

    public static void main (String [] args){
        new Servidor();
    }
}
```

Arquitectura Rmi



Stubs y Skeletons

Una vez definidas la interfase e implementación de las respectivas funciones es necesario definir otro elemento para ejecutar e invocar funciones remotas: Stubs y Skeletons

.

Estos Stubs y Skeletons permiten que al momento de ser invocada la función remota esta pueda ser simulada localmente

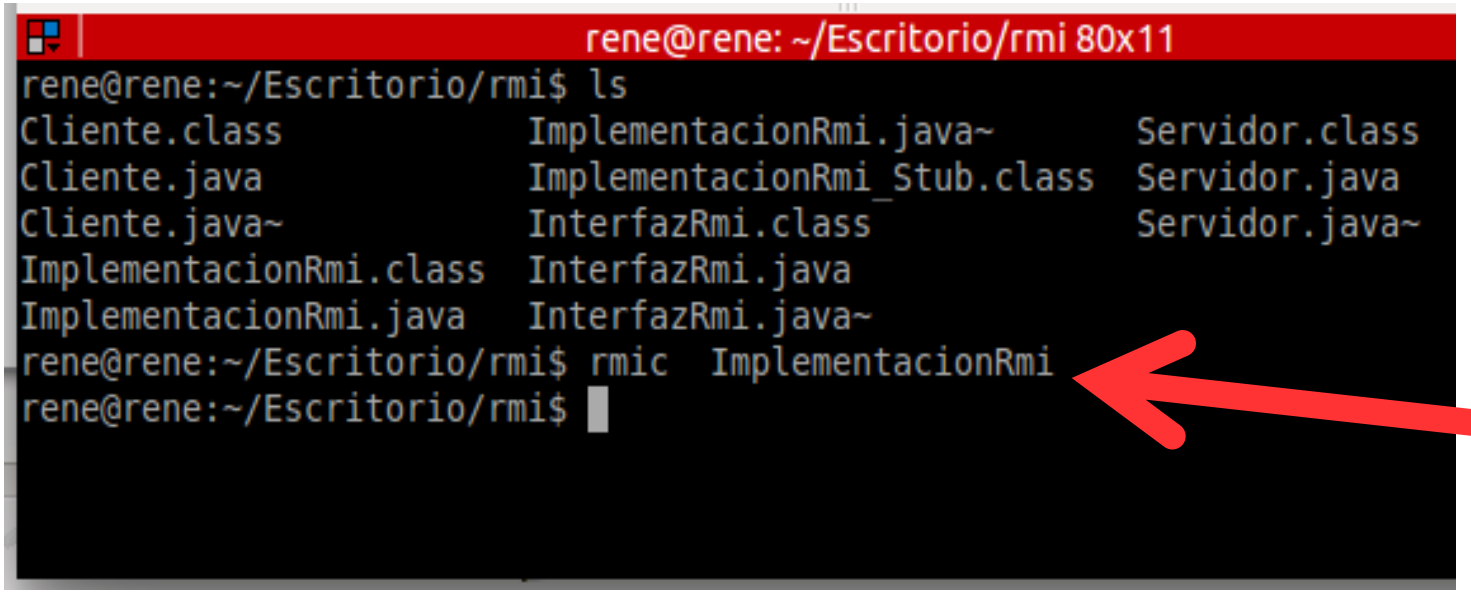
Stubs y Skeletons

- El **Stub** funciona como un simulador para todas las funciones que están siendo invocadas y pertenecen a la implementación de servidor,

- El **Skeleton** funciona como simulador para recibir parámetros de la implementación de cliente

Stubs y Skeletons

Para generar estos "**Stubs**" y "**Skeletons**" java ofrece una herramienta llamada **rmic** , basta ejecutar rmic sobre la implementación de las funciones correspondientes



ESPAMMFL

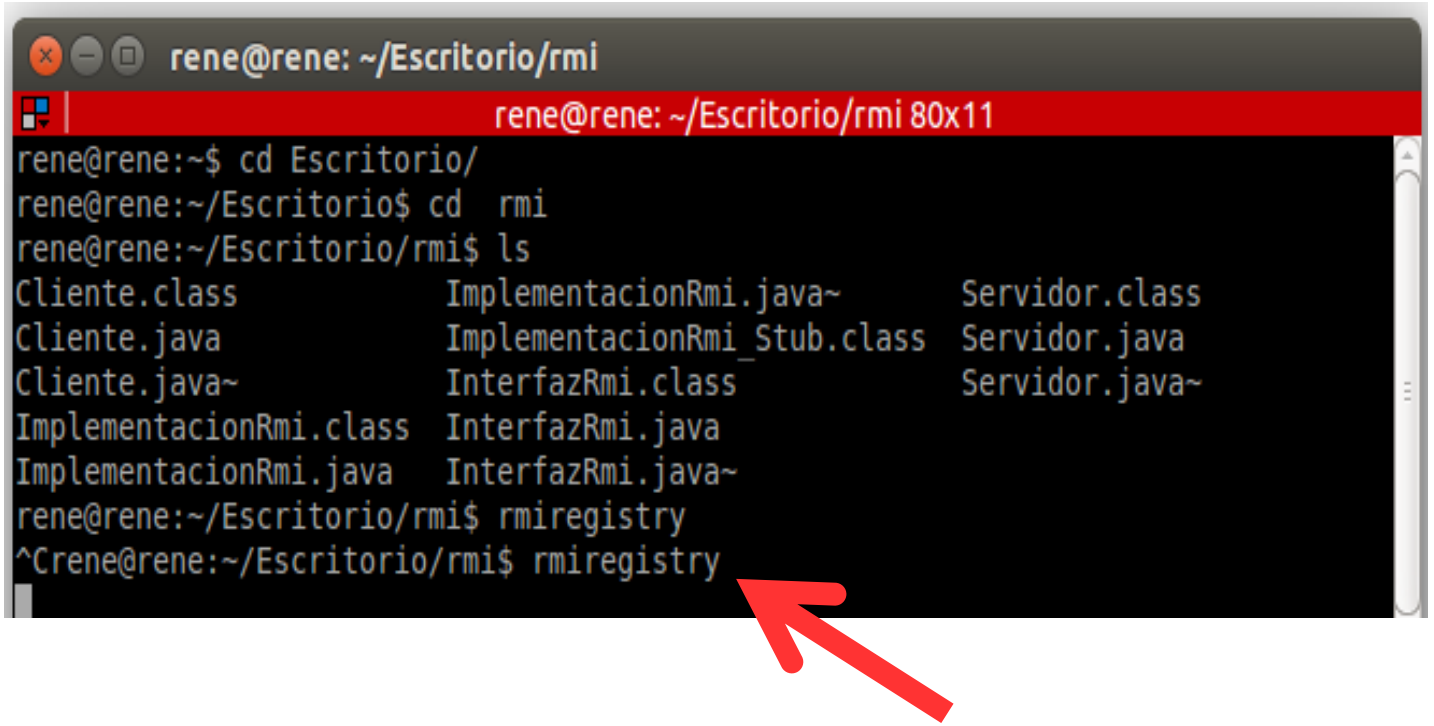
MTI
Maestría en Tecnologías de la Información

Rmi registry

La comunicación requiere definir lo que es denominado **RMI Registry**, el **RMI Registry** se establece en un puerto TCP/IP (por default **1099**) y mantiene un mapa de las funciones remotas que serán utilizadas, esto es, cuando arriva la requisición para una función remota en el puerto conocido

Rmi registry

RMI Registry será encargado de redireccionar a la implementación apropiada.

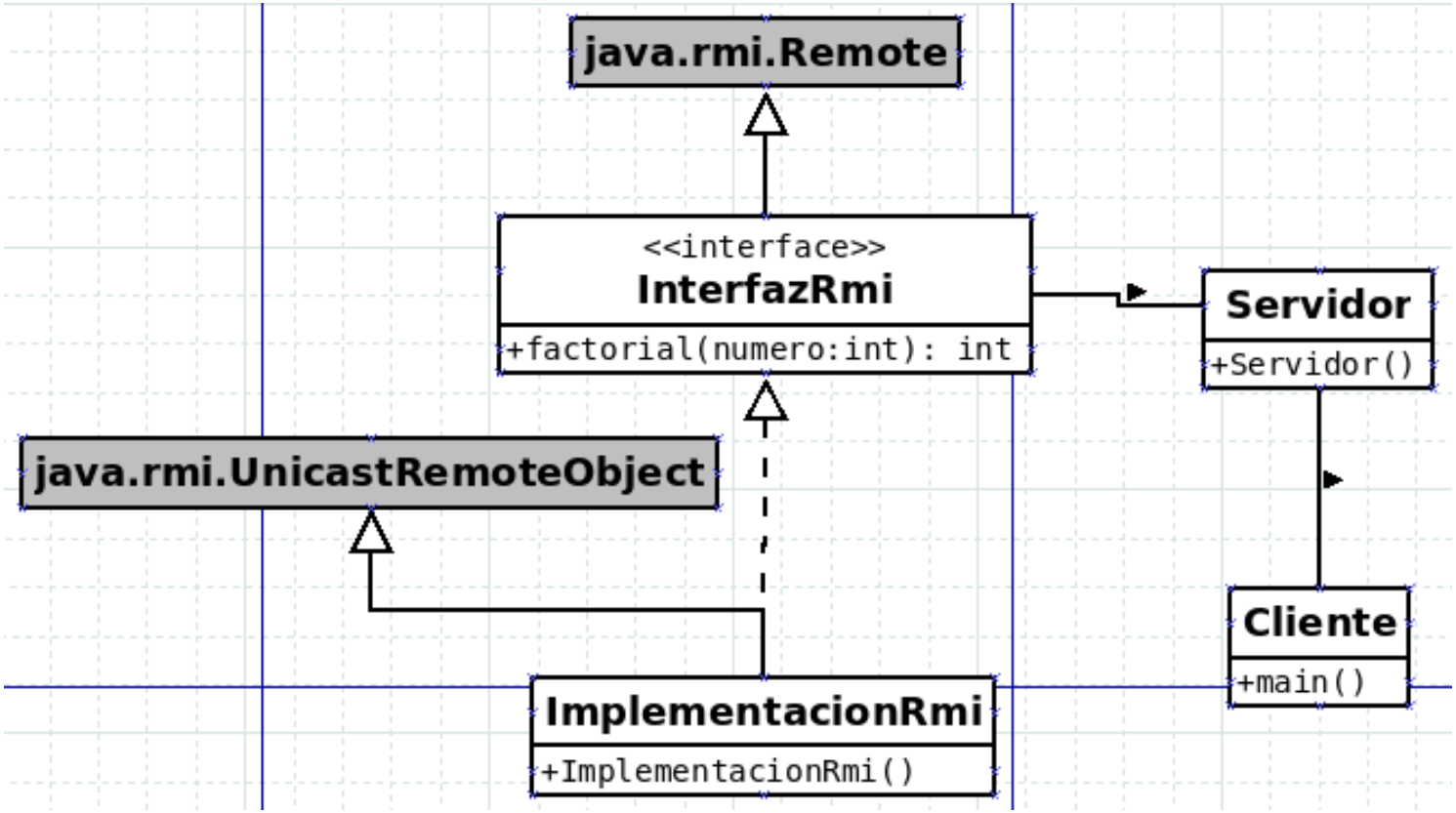


ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Ejemplo RMI

Cliente:



ESPAMMFL

MTI
Maestría en Tecnologías de la Información

Ejemplo RMI

Cliente:

```
import java.rmi.Naming;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Cliente{
        public static void main(String args[]){
                try{
                        InterfazRmi interfaz = (InterfazRmi)Naming.lookup
                                ("rmi://localhost/saludo");
                        System.out.println("como te llamas");
                        Scanner escanner = new Scanner(new InputStreamReader
                                (System.in));
                        System.out.println(interfaz.saludar(escanner.next()));
                }catch(Exception ex){ }
        }
}
```