

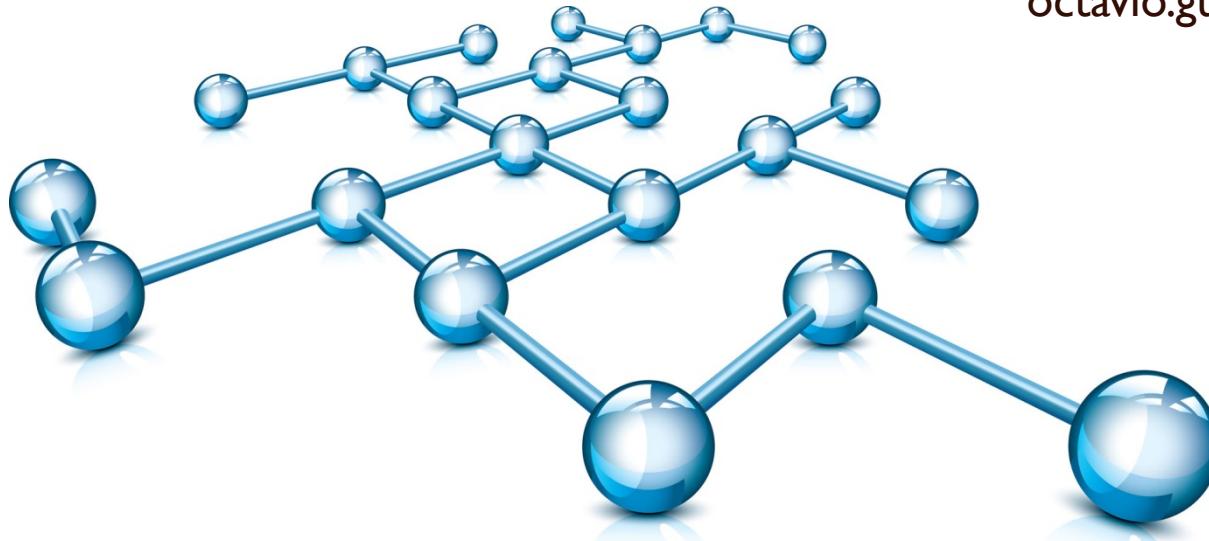


# Invocación remota

Profesor:

Dr. J. Octavio Gutiérrez García

[octavio.gutierrez@itam.mx](mailto:octavio.gutierrez@itam.mx)



# Invocación remota



Aplicaciones, Servicios

Invocación remota, comunicación indirecta

Primitivas subyacentes de comunicación entre procesos:  
Sockets, paso de mensajes, empaquetado y  
representación de datos.

UDP y TCP

# Invocación remota

- Llamada a procedimientos remotos  
RPC - Remote Procedure Call



- Invocación de métodos remotos  
RMI – Remote Method Invocation



- Protocolos request-reply  
HTTP - HyperText Transfer Protocol



# Llamada a procedimientos remotos



- Por Birrel y Nelson (1985)
- Objetivo de RPC: **acercar la semántica** de las llamadas a **procedimiento** convencional a un entorno distribuido (transparencia)



- RPC constituye el **núcleo** de muchos SDs
- Han **evolucionado** hacia orientación a objetos:  
Invocación de métodos remotos (**RMI**)

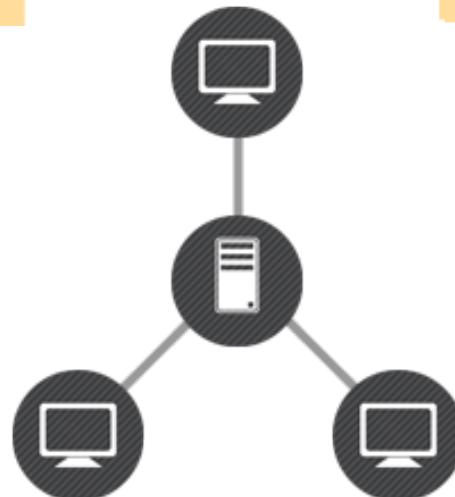
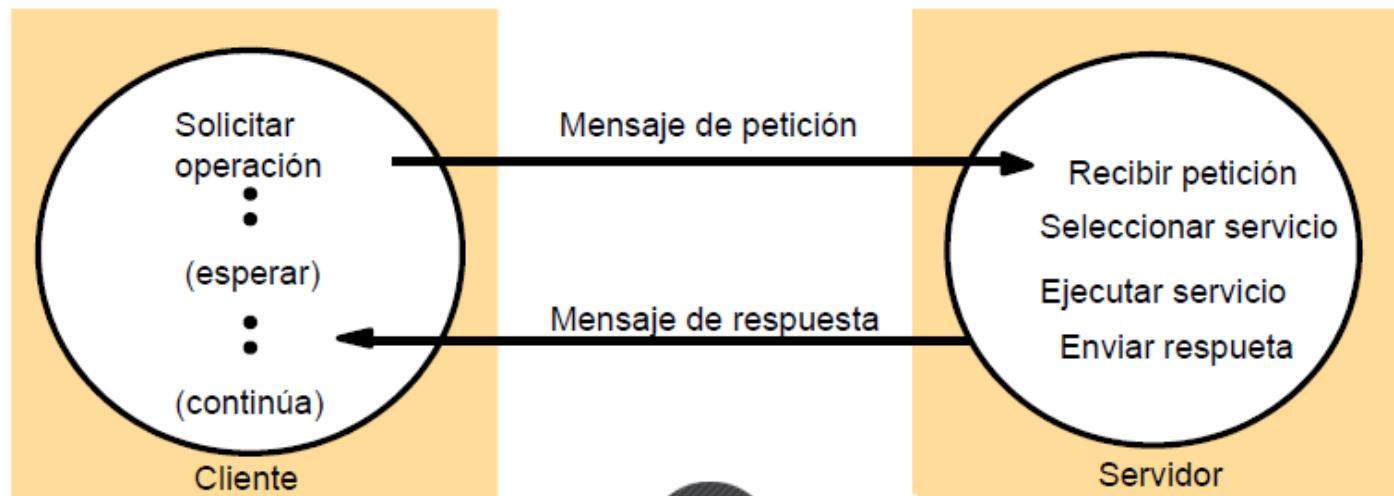
# Llamada a procedimientos remotos



Ofrece una interfaz “**sencilla**” para construir aplicaciones distribuidas

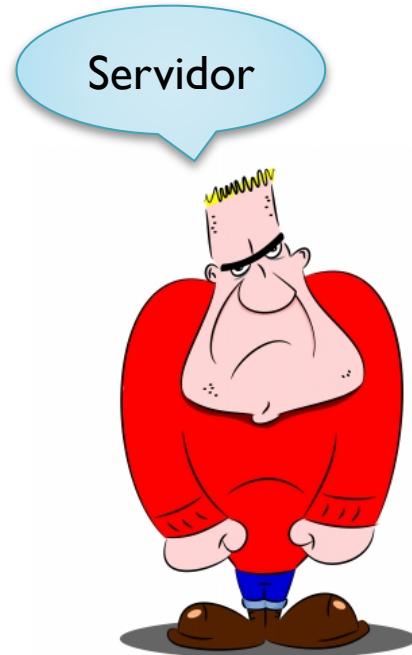
# Llamada a procedimientos remotos

## Comunicación cliente-servidor



# Llamada a procedimientos remotos

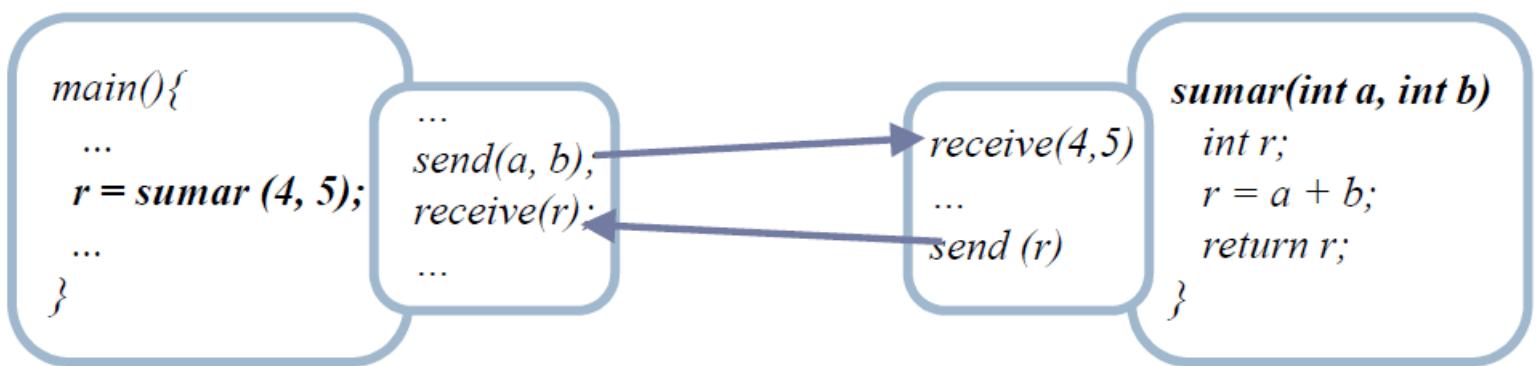
- Una RPC tiene dos **participantes**:



Un **cliente activo** que envía una  
RPC al servidor



Un **servidor pasivo** que calcula un  
resultado y lo devuelve al cliente

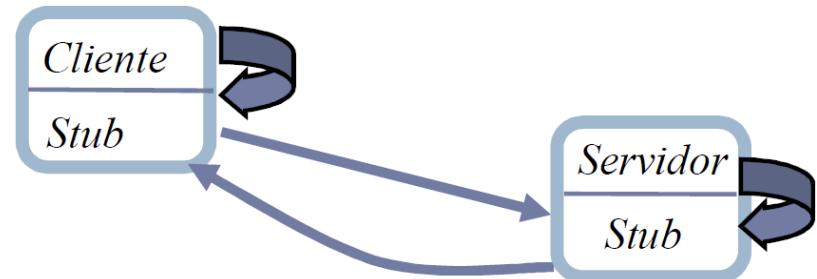


- 1) El proceso que realiza la llamada **empaqueta** los argumentos en un mensaje
- 2) Envía mensaje a otro proceso
- 3) Espera el resultado
- 4) El proceso que ejecuta el procedimiento **extrae** los argumentos del mensaje
- 5) Realiza la **llamada** de forma **local**
- 6) Envía resultado de vuelta.



# Llamada a procedimientos remotos

- Suplentes (*stubs*)

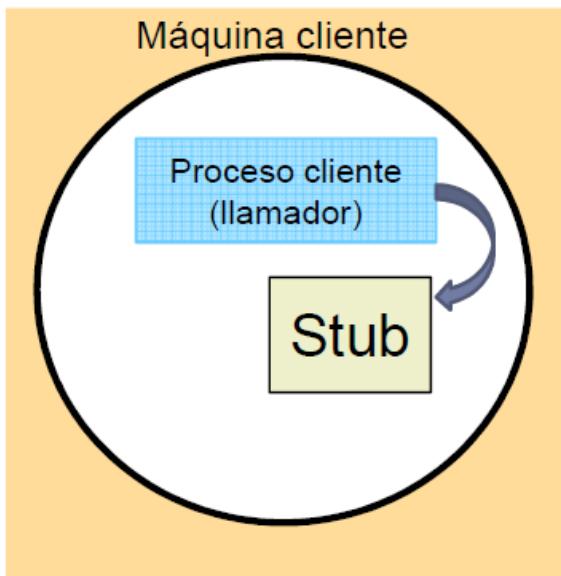


- Responsables de **convertir** los parámetros de la aplicación cliente/servidor durante una llamada a procedimiento remoto



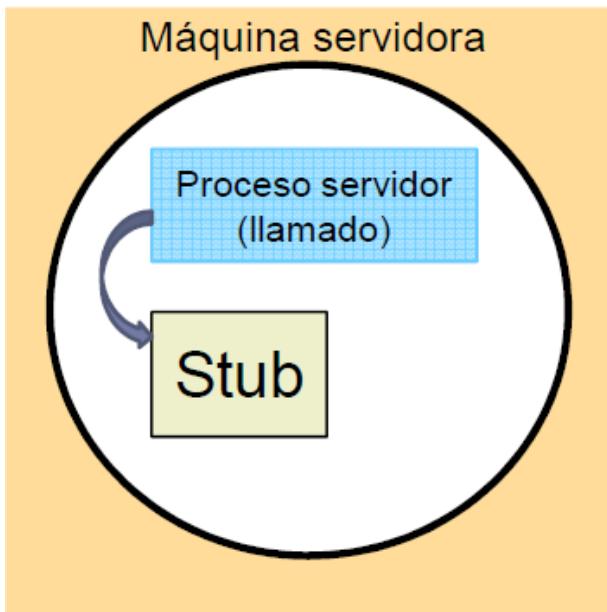
Se generan **automáticamente** por el software de RPC

# Protocolo básico



- **Proceso cliente**
  - **Conectar** al servidor
  - **Invocar** una llamada a procedimiento remoto
  
- **Stub** del cliente:
  - **Empaquetar** los parámetros y construir los mensajes
  - **Enviar** los mensajes al servidor
  - **Bloquearse** hasta esperar la respuesta
  
- Obtener la **respuesta**

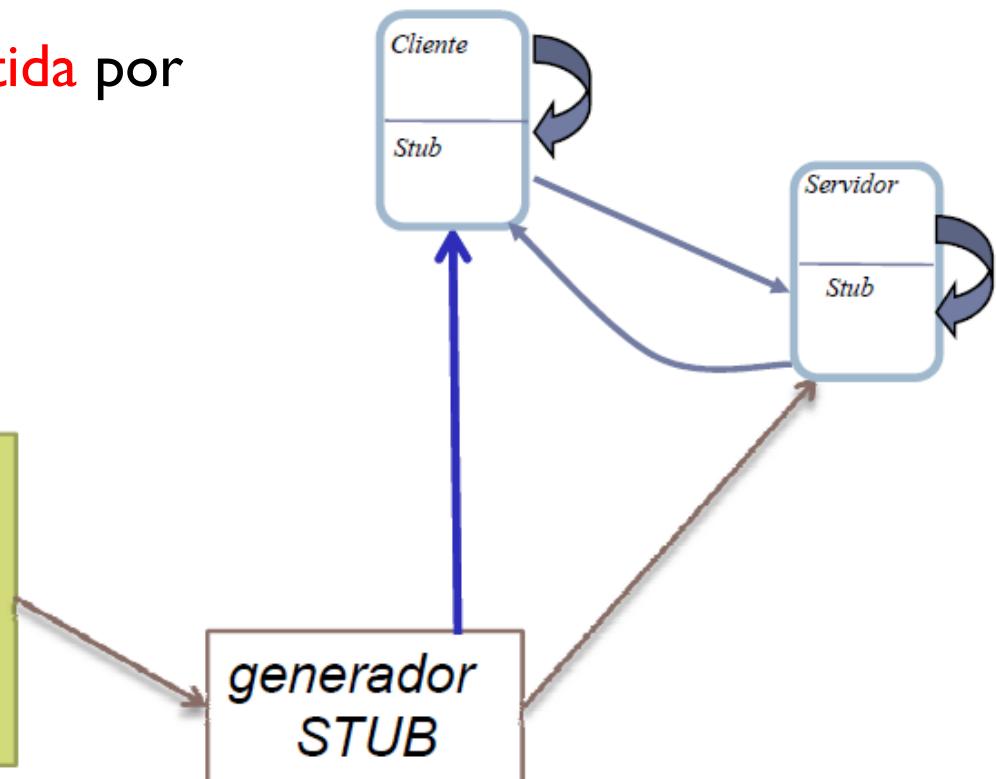
# Protocolo básico



- **Proceso servidor**
  - Registrar las RPC
  - Implementar los procedimientos
- *Stub* del servidor:
  - **Recibir** petición del cliente
  - **Desempaquetar** los parámetros
  - **Invocar** el procedimiento de manera local
  - **Bloquearse** hasta esperar la respuesta
- **Enviar respuesta** al cliente



- Programación con **interfaces**
- Un Lenguaje de Definición de Interfaz (**IDL**) para especificar el formato de los procedimientos remotos
- La **interfaz es compartida** por
  - Cliente y
  - Servidor



<opciones de comunicación:  
versión interfaz  
>  
<interfaz remota:  
suma (in int a, in int b, out int c) ;  
>



# Llamada a procedimientos remotos

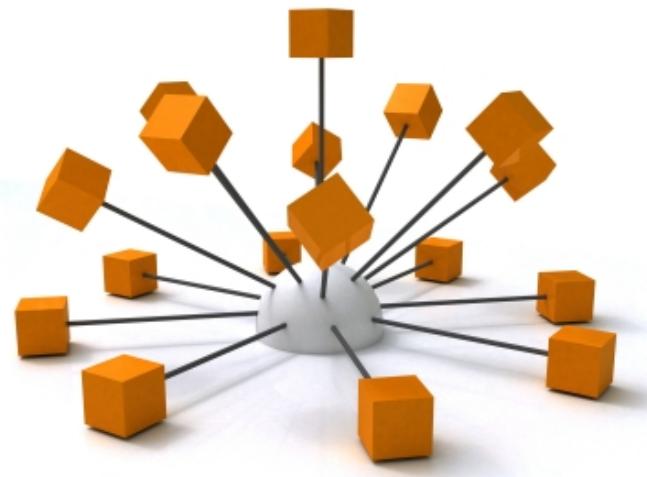
- Una **interfaz** especifica:
  - **Nombre de servicio** que utilizan los clientes y servidores
  - Nombres de los **procedimientos**
  - **Parámetros** de los procedimientos
    - Entrada
    - Salida
  - **Tipos de datos** de los argumentos



# Llamada a procedimientos remotos

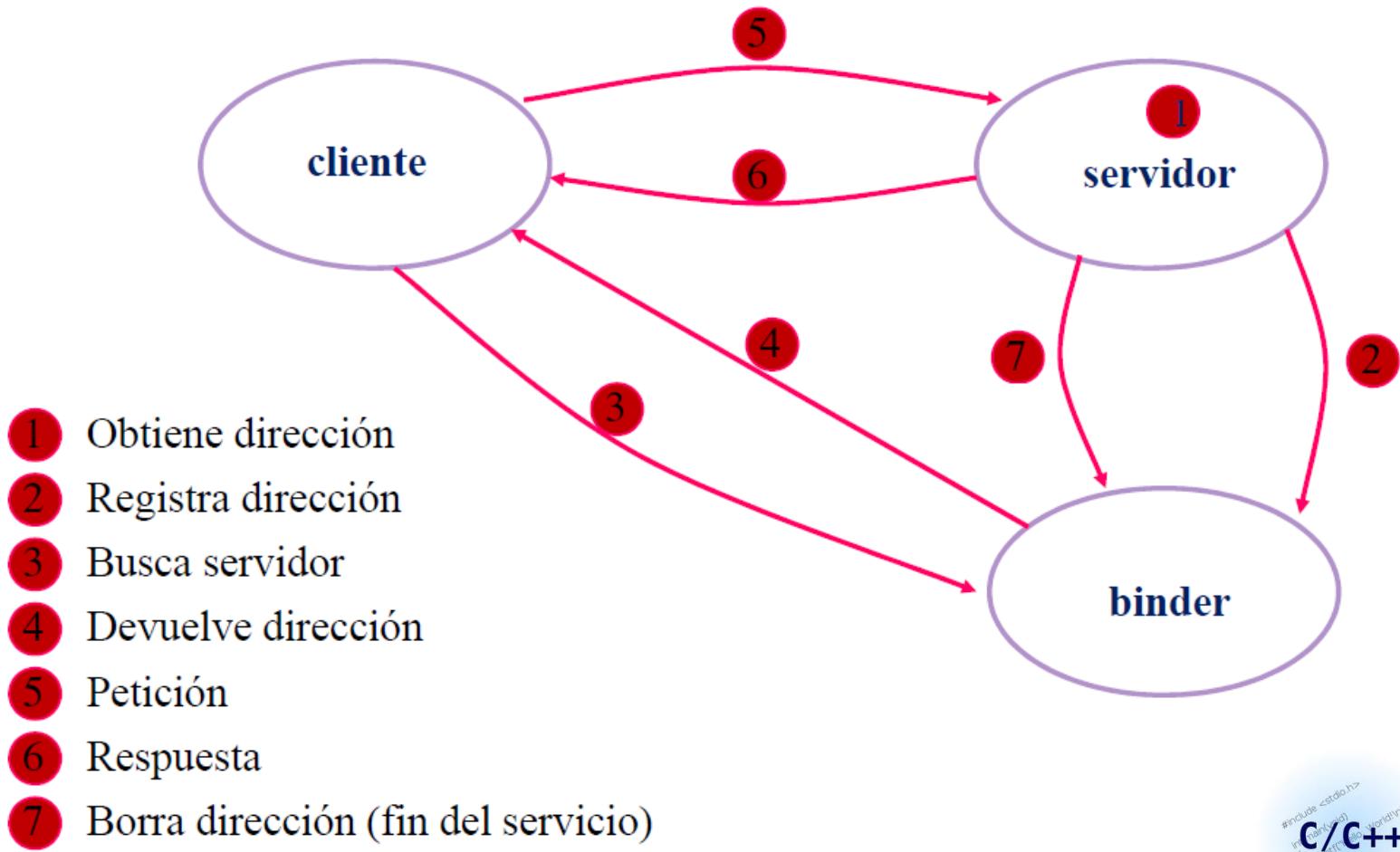
## Enlace (*Binding*)

- Asociación entre el cliente y el servidor
- Implica localizar al servidor que ofrece un determinado servicio
- El servidor debe registrar su dirección en un servicio de nombres (binder)





# Esquema de registro y enlace



```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

# Llamada a procedimientos remotos

- Aplicación Hello

hello.c

Servidor



```
#include <stdio.h>
#include <windows.h>

void HelloProc(char * pszString)
{
    printf("%s\n", pszString);
}

void Shutdown(void)
{
    RPC_STATUS status;

    status = RpcMgmtStopServerListening(NULL);

    if (status)
    {
        exit(status);
    }

    status = RpcServerUnregisterIf(NULL, NULL, FALSE);

    if (status)
    {
        exit(status);
    }
}
```

Saluda

Deja de escuchar

Se da de baja

C/C++

# Llamada a procedimientos remotos

- Interface **hello.idl**

```
[  
    uuid(3f9fd0c7-d4a1-4bb8-b46b-c763b28a06e5),  
    version(1.0)  
]  
interface hello  
{  
    void HelloProc([in, string] unsigned char * pszString);  
    void Shutdown(void);  
}
```

# Llamada a procedimientos remotos

- Los **STUBS** se crean con compiladores especializados, por ejemplo: MIDL (**M**icrosoft **I**nterface **Description **L**anguage)**

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

**C/C++**

# Llamada a procedimientos remotos

## hello\_c.s (stub del cliente)

```
void HelloProc(/* [string][in] */ unsigned char *pszString)
{
    NdrClientCall2(
        ( PMIDL_STUB_DESC )&hello_StubDesc,
        (PFORMAT_STRING) &hello__MIDL_ProcFormatString.Format[0],
        ( unsigned char * )&pszString);
}
```

### void Shutdown( void)

```
{
    NdrClientCall2(
        ( PMIDL_STUB_DESC )&hello_StubDesc,
        (PFORMAT_STRING) &hello__MIDL_ProcFormatString.Format[30],
        ( unsigned char * )0);
}
```



Responsables de convertir los parámetros de la aplicación cliente/servidor durante una llamada a procedimiento remoto

# Llamada a procedimientos remotos

- Aplicación del cliente

Crea el enlace:



```
RpcBindingFromStringBinding(  
    pszStringBinding,  
    &hello_IfHandle  
)
```

# Llamada a procedimientos remotos

- Aplicación del cliente
  - Llama a los procedimientos remotos:



## RpcTryExcept

```
{  
    1 HelloProc(pszString);  
  
    2 Shutdown();  
}
```

# Llamada a procedimientos remotos

Aplicación del servidor

Registrar el servidor con el servicio  
de nombres (Binder)

```
RpcServerRegisterIf(  
    hello_v1_0_s_ifspec,  
    ...);
```



# Llamada a procedimientos remotos

- Aplicación del servidor

Escucha peticiones

```
RpcServerListen(  
    cMinCalls,  
    RPC_C_LISTEN_MAX_CALLS_DEFAULT,  
    fDontWait);
```



# Llamada a procedimientos remotos

## Aplicación del servidor

Detiene el servidor

Dejar de **escuchar** peticiones

RpcMgmtStopServerListening(NULL);

Dar de **baja** el servicio

RpcServerUnregisterIf(NULL, NULL, FALSE);



#include <stdio.h>  
int main(void)  
{  
 printf("HelloWorld\\n");  
 return 0;  
}

**C/C++**



# Fallos en las RPC

- El cliente podría **no** ser capaz de **localizar** al **servidor**
- **Pérdidas** de mensajes
  - Se pierde el mensaje de **petición** del cliente al servidor
  - Se pierde el mensaje de **respuesta** del servidor al cliente
- El **servidor falla** después de recibir una petición
- El **cliente falla** después de enviar una petición

# Invocación remota

- Llamada a procedimientos remotos  
RPC - Remote Procedure Call



- Invocación de métodos remotos  
RMI – Remote Method Invocation



- Protocolos request-reply  
HTTP - HyperText Transfer Protocol





# Invocación de métodos remotos

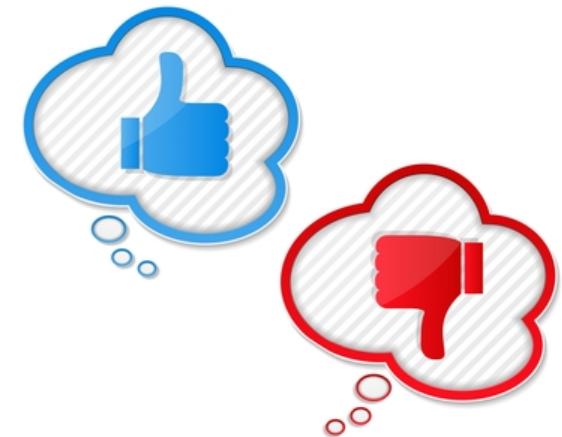
- Modelo **equivalente** a las **llamadas a procedimientos remotos**
- Primera aproximación al uso de un modelo **orientado a objetos** sobre aplicaciones distribuidas
- **Objetos distribuidos** dentro de una red. Los objetos proporcionan **métodos**, los cuales dan acceso a los **servicios**



# RMI vs Sockets

- **Ventajas**

- Los programas RMI son más sencillos de diseñar
- Servidor RMI concurrente



- **Inconvenientes:**

- Sockets tienen menos sobrecarga
- “RMI sólo para plataformas Java”

# RMI y RPC



- En común
  - Interfaces
  - Basados en protocolos petición-respuesta
  - Transparentes
- Diferencias
  - Programación orientada a objetos
  - Referencias a los objetos como parámetros

# Java RMI

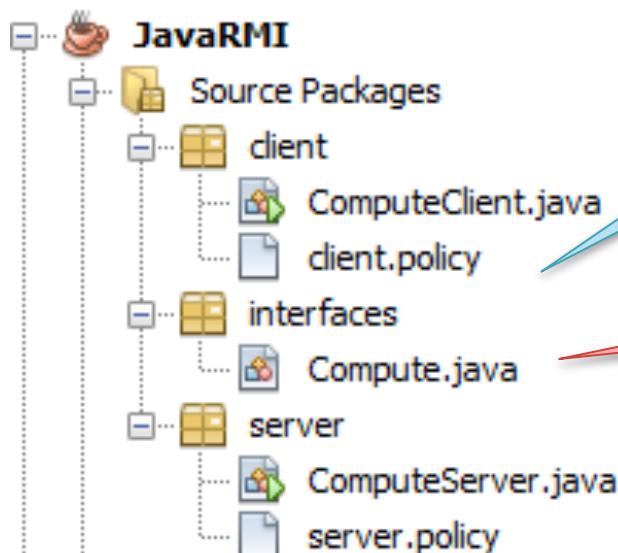


## Ofrece

- Mecanismos para crear **servidores** y **objetos** cuyos métodos se pueden **invocar remotamente**.
- Mecanismos que permiten a los **clientes** **localizar** los **objetos** remotos.
- Servicio de directorios
  - **rmiregistry**, servicio de directorios de Java
  - Se **ejecuta en** la máquina **servidor** objeto



# Hola mundo: Java RMI



Autoriza el acceso  
al puerto de RMI  
1099

Interfaz que  
describe el servicio  
“Compute”

Implementa la  
interfaz “Compute”

# Java RMI

## Paso 0. Crear permisos para Cliente y Servidor

```
System.setProperty("java.security.policy","file:/C:/.../server.policy");
```

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

```
client.policy //ubicado en el mismo paquete  
server.policy //ubicado en el mismo paquete
```

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



# Java RMI

## Paso 1. Definir la interfaz

```
package interfaces;
```

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

```
public interface Compute extends Remote {
```

```
    // Calculate the square of a number  
    public double square ( int number ) throws RemoteException;
```

```
    // Calculate the power of a number
```

```
    public double power ( int num1,int num2) throws RemoteException;
```

```
}
```



JavaRMI

# Java RMI

## Paso 2. Implementar la interfaz (en el Servidor)

```
public class ComputeServer implements Compute {
```

```
    public ComputeServer() throws RemoteException {  
        super();  
    }
```

```
    @Override  
    public double square(int number) throws RemoteException {  
        return (number*number);  
    }
```

```
    @Override  
    public double power(int num1, int num2) throws RemoteException {  
        return (java.lang.Math.pow(num1, num2));  
    }
```

...



# Java RMI

## Paso 3. Arrancar el servidor de nombres (en el servidor)

Vía código al instanciar al servidor:

```
LocateRegistry.createRegistry(1099);
```

O antes de instanciar al servidor en una consola

```
start rmiregistry
```



# Java RMI

## Paso 4. Generar y registrar el stub (del Servidor)

```
String name = "Compute";
```

```
ComputeServer engine = new ComputeServer();  
Compute stub = (Compute) UnicastRemoteObject.exportObject(engine, 0);
```

```
Registry registry = LocateRegistry.getRegistry();  
registry.rebind(name, stub);
```



# Java RMI

## Paso 5. Localización del servicio (por el cliente)

```
String name = "Compute";
```

```
Registry registry = LocateRegistry.getRegistry("localhost"); // server's ip address
```

```
Compute comp = (Compute) registry.lookup(name);
```



# Java RMI

## Paso 6. Invocar métodos remotos (por el cliente)

```
System.out.println("3^2 = "+comp.square(3));
```

```
System.out.println("3^3 = "+comp.power(3, 3));
```



# Java RMI



- Práctica de Laboratorio de RMI
  - Incluir **credenciales** en las llamadas a los métodos de “Compute”
    - Credential(“Juan”, “Aguascalientes”, 1987, “123456”)



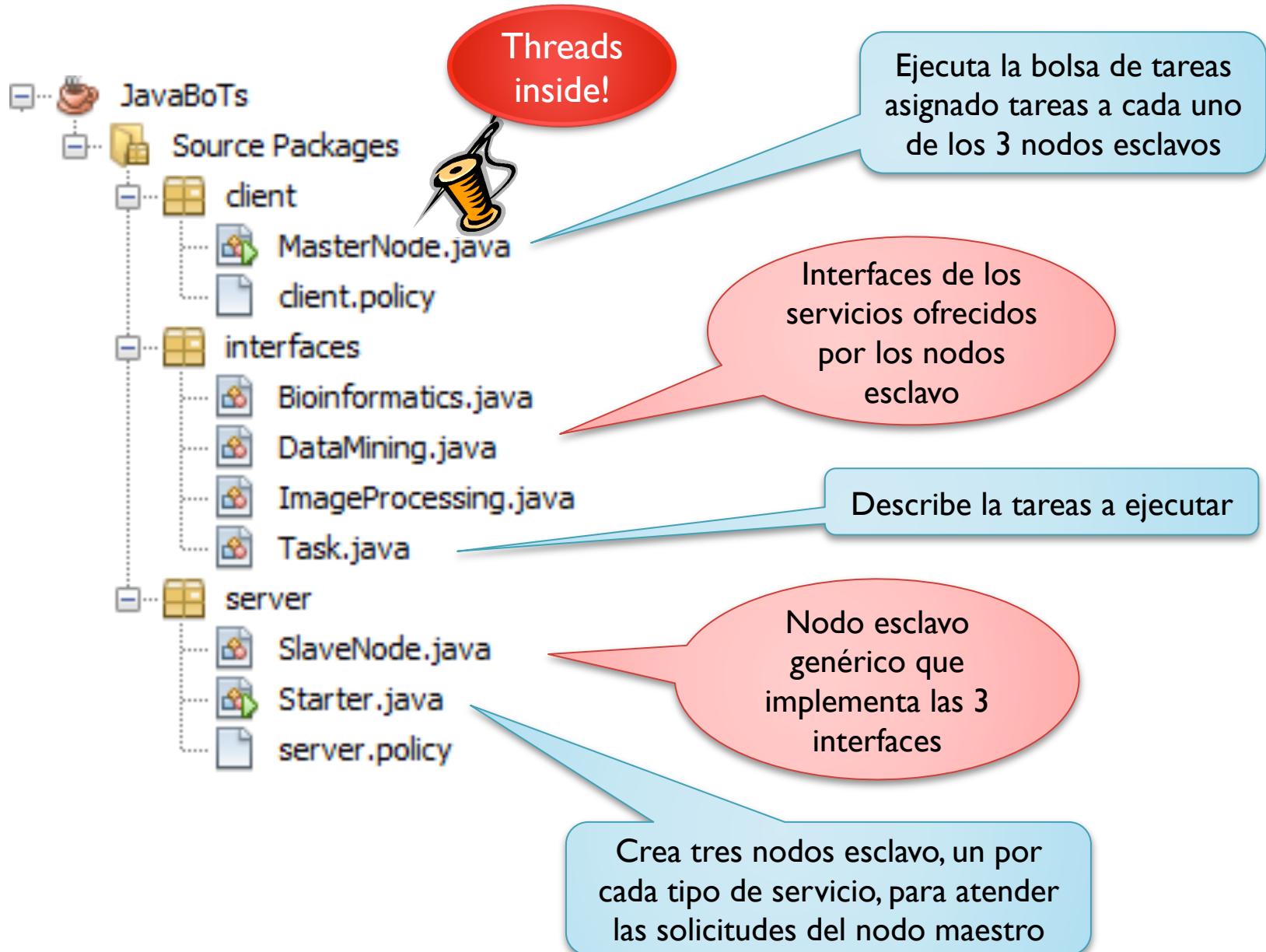
# Java RMI



- Práctica de Laboratorio de RMI
  - Bolsa de Tareas



# Java RMI: Bolsa de Tareas



# Java RMI: Bolsa de Tareas



## Nodo “Maestro”

- Contiene tres bolsas de tareas
  - Procesamiento de imágenes (10 tareas)
  - Minería de datos (20 tareas)
  - Bioinformática (15 tareas)
- Las tareas son **independientes entre sí**, es decir, no hay restricciones de orden ni dependencias entre ellas.
- Las **tareas** son **heterogéneas**
  - Requieren diferentes **servicios**
    - Tareas de procesamiento de imágenes – Servicio: “Imagenes”
    - Tareas de minería de datos – Servicio: “Mineria”
    - Tareas de bioinformática – Servicio: “Bioinformatica”
  - Tardan diferente **tiempo** en ejecutarse
    - 5, 10, 15, 20 y 30 segundos



STEINSTEIN  
COMPUTER PARK

# Java RMI: Bolsa de Tareas



- Definición de las bolsas de tareas
- [T1, Imagenes, 5] [T2, Imagenes, 10] [T3, Imagenes, 15] [T4, Imagenes, 20]  
[T5, Imagenes, 30] [T6, Imagenes, 5] [T7, Imagenes, 10] [T8, Imagenes, 15]  
[T9, Imagenes, 20] [T10, Imagenes, 30]
- [T11, Mineria, 5] [T12, Mineria, 10] [T13, Mineria, 15] [T14, Mineria, 20]  
[T15, Mineria, 30] [T16, Mineria, 5] [T17, Mineria, 10] [T18, Mineria, 15]  
[T19, Mineria, 20] [T20, Mineria, 30] [T21, Mineria, 5] [T22, Mineria, 10]  
[T23, Mineria, 15] [T24, Mineria, 20] [T25, Mineria, 30] [T26, Mineria, 5]  
[T27, Mineria, 10] [T28, Mineria, 15] [T29, Mineria, 20] [T30, Mineria, 30]
- [T31, Bioinformatica, 5] [T32, Bioinformatica, 10] [T33, Bioinformatica, 15]  
[T34, Bioinformatica, 20] [T35, Bioinformatica, 30] [T36, Bioinformatica, 5]  
[T37, Bioinformatica, 10] [T38, Bioinformatica, 15] [T39, Bioinformatica, 20]  
[T40, Bioinformatica, 30] [T41, Bioinformatica, 5] [T42, Bioinformatica, 10]  
[T43, Bioinformatica, 15] [T44, Bioinformatica, 20] [T45, Bioinformatica, 30]

# Java RMI: Bolsa de Tareas



- Nodo “**Maestro**”
  - Manda ejecutar todas las tareas a nodos esclavos tan rápido como sea posible
  - Una vez que todas las tareas han sido ejecutadas, el nodo maestro imprime el tiempo que duro la ejecución.



# Java RMI: Bolsa de Tareas



- **3 Nodos “Esclavo”**

- Cada **esclavo** simula la ejecución de tareas de un cierto tipo  
[Imagenes, Minería, Bioinformática]
- Recibe órdenes de ejecución de tareas del nodo maestro
  - Ejemplo: [T1, Imagenes, 5]
- Regresa el resultado de la tarea
  - Ejemplo: [T1, Imagenes, 5, **Resultado-Imagen**]
- Las tareas son heterogéneas
  - Requieren diferentes servicios
    - Procesamiento de imágenes (10 tareas) – Servicio: “Imagenes”
    - Minería de datos (20 tareas) – Servicio: “Minería”
    - Bioinformática (15 tareas) – Servicio: “Bioinformática”
  - Tardan diferente tiempo en ejecutarse
    - 5, 10, 15, 20 y 30 segundos



# Java RMI: Bolsa de Tareas



## Paso 1:

- Definir clase **Task**
  - taskId
  - requirementId
  - length
  - output



# Java RMI: Bolsa de Tareas



## Paso 2:

- Definir tres **interfaces** con un método cada una;

**Bioinformatics**

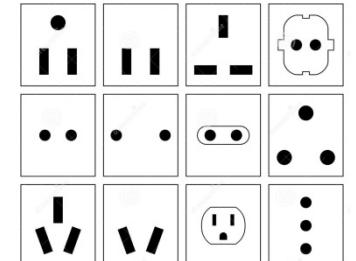
- executeBioTask(task aTask);

**DataMining**

- executeDataTask(task aTask);

**ImageProcessing**

- executeImageTask(task aTask);



# Java RMI: Bolsa de Tareas



## Paso 3:

- Programar **Nodo Maestro**
  - Obtener el registro RMI
  - Crear **bolsas de tareas**
    - Para cada bolsa crear un hilo de ejecución
      - **Hilo ejecutor** de tareas
      - Entrada: (Task[] o Task) y Registry
      - Salida: Tarea ejecutada
    - Algoritmo
      - Buscar servicio para ejecutar tarea
      - Ejecutar una por una cada tarea (invocación a método remoto)
      - Imprimir Salida



# Java RMI: Bolsa de Tareas



## Paso 4:

- Programar **Nodo Esclavo**
  - Entrada: **TipoServicio**
  - Método **despliegue**
    - Crear stub usando la interfaz correspondiente
    - Registrar el servicio
  - Debe **implementar** las tres **interfaces**
    - Bioinformatics   -DataMining   -ImageProcessing
  - Dentro de la implementación de los métodos de la interfaz
    - **Simular ejecución** con Thread.sleep(ms);



# Java RMI: Bolsa de Tareas



## Paso 5:

- Programar **Starter**



- Arranca el **RMI registry**
- Crear instancia de esclavo Bioinformatics
- Crear instancia de esclavo ImageProcessing
- Crear instancia de esclavo DataMining
- **Desplegar los tres servicios**

# Pseudocódigo – Nodo Maestro

- Obtener el registro
- Definir las 3 Bolsas de Tareas
- Para cada **bolsa de tareas BoT** hacer
  - Crear hilo ejecutor(BoT, registro);
  - Iniciar hilo ejecutor



# Pseudocódigo – Hilo en Nodo Maestro



Si (t.getRequirement() = "DataMining") {

- Busca servicio de minería de datos -> registry.lookup ("DataMining")

} si no, Si (t.getRequirement() = "ImageProcessing") {



- Busca servicio de procesamiento de imágenes -> registry.lookup ("ImageProcessing")

} si no {

- Busca servicio de bioinformática -> registry.lookup ("Bioinformatics")

}

Para cada tarea

Invoca a método remoto del servicio correspondiente

# Pseudocódigo - Interfaces

- 1 interfaz **Bioinformatica** {  
    ejecutarTareaBioinformatica(Tarea t)  
}
  
- 2 interfaz **MineriaDatos** {  
    ejecutarTareaMineriaDatos(Tarea t)  
}
  
- 3 interfaz **ProcesamientoImagenes** {  
    ejecutarTareaProcesamientoImagenes(Tarea t)  
}

# Pseudocódigo – Nodo Esclavo

Entrada -> Tipo de Servicio

{ Bioinformatica , MineriaDatos, ProcesamientoImagenes }

Si (Tipo de Servicio = " Bioinformatica ") {

    Registrarse en registro RMI con interfaz Bioinformatica

} si no, Si (Tipo de Servicio = " MineriaDatos ") {

    Registrarse en registro RMI con interfaz MineriaDatos

} si no {

    Registrarse en registro RMI con interfaz ProcesamientoImagenes

}



# Pseudocódigo – Starter



Crear **registro RMI**

Crear **NodoEsclavo1**

NodoEsclavo1.despliegue("Bioinformatica")

Crear **NodoEsclavo2**

NodoEsclavo2.despliegue("MineriaDatos")

Crear **NodoEsclavo3**

NodoEsclavo2.despliegue("ProcesamientoImagenes")

Registra y  
vincula el  
servicio

# Definición de un hilo en Java

```
private static class MyThread extends Thread {  
  
    public MyThread() {  
        // your code here  
    }  
  
    public void run() {  
        // your code here  
    }  
  
}
```



```
MyThread ex1 = new MyThread();  
ex1.start();
```

# Meta

Ejecutar todas las tareas en  
**5 minutos 20 segundos**

