Invocación de Métodos Remotos: Prácticas de Laboratorio

XIII Jornadas de Enseñanza Universitaria de la Informática
Teruel, 16-18 de julio de 2007

Coromoto León Hernández, Gara Miranda Valladares
(cleon | gmiranda)@ull.es

Dpto. Estadística, I.O. y Computación.
Universidad de La Laguna
Tenerife

Contenido

- Contexto
  - Programación de Sistemas Distribuidos
- Paradigma de Objetos Distribuidos
  - Llamadas a procedimientos remotos
  - Invocación de Métodos Remotos
- Ejercicios de Laboratorio
  - Ejemplo Simple: Suma Vectorial
  - Ejemplo Complejo: Técnicas Algorítmicas
- Conclusiones

Contexto

- El *Centro Superior de Informática* de la Universidad de La Laguna se fundó en el año 1990.
- En el 2003 cambia de nombre:

  Escuela Técnica Superior de Ingeniería Informática
- Los estudios que se imparten en el mismo son:
  - Ingeniería Técnica en Informática de Gestión
  - Ingeniería Técnica en Informática de Sistemas
  - Ingeniería Superior en Informática
- La asignatura asociada a este trabajo es:
  - Programación de Sistemas Distribuidos
  - Optativa de la Ingeniería Técnica e Informática de Sistemas
  - Créditos: 6,0

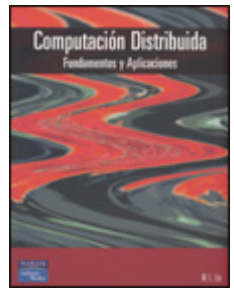
Programación de Sistemas Distribuidos

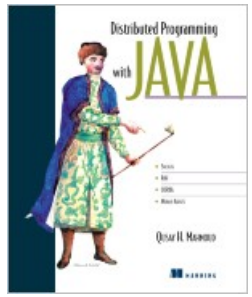
- Contenido
  - Comunicación entre Procesos
  - Paradigma Cliente/Servidor
  - Interfaces de Programación de Sockets
    - orientados a conexión
    - no orientados a conexión
  - Objetos Distribuidos
    - Invocación de Métodos Remotos
    - CORBA
- Herramienta: Java

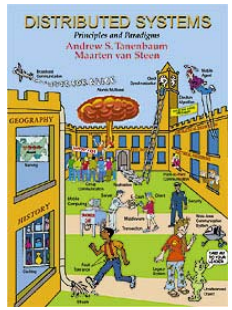
Práctica	**Semanas**
Creación de Hilos (*Threads*)	1
Sincronización de Hilos (*Threads*)	1
Direcciones IP y Nombres de dominio	1
Serialización de objetos	1
Comunicación no orientada a conexión	2
Comunicación orientada a conexión	2
Comunicación en grupos	1
Invocación de Métodos Remotos	2
CORBA	2
Servicio de resolución mediante Técnicas algorítmicas	1
Total	**14**

Bibliografía

- G. Coulouris, G., Dollimore, J., Kindberg, T. "*Sistemas Distribuidos: Conceptos y Diseño*", Addison-Wesley, 3ra. Edición, 2001.
- Liu, M. L. "*Computación Distribuida. Fundamentos y Aplicaciones".* Addison-Wesley, 1ra. Edición, 2004.
- Mahmoud, Qusay H. "*Distributed Programming with Java*". Manning, 1999.
- Tanenbaum, A, van Steen, M. "*Distributed Systems: Principles and Paradigms*", Prentice Hall, 2002.
- Horstmann, C. S., Cornell, G. "*Java 2. Volumen II – Características Avanzadas",* Prentice-Hall 2002

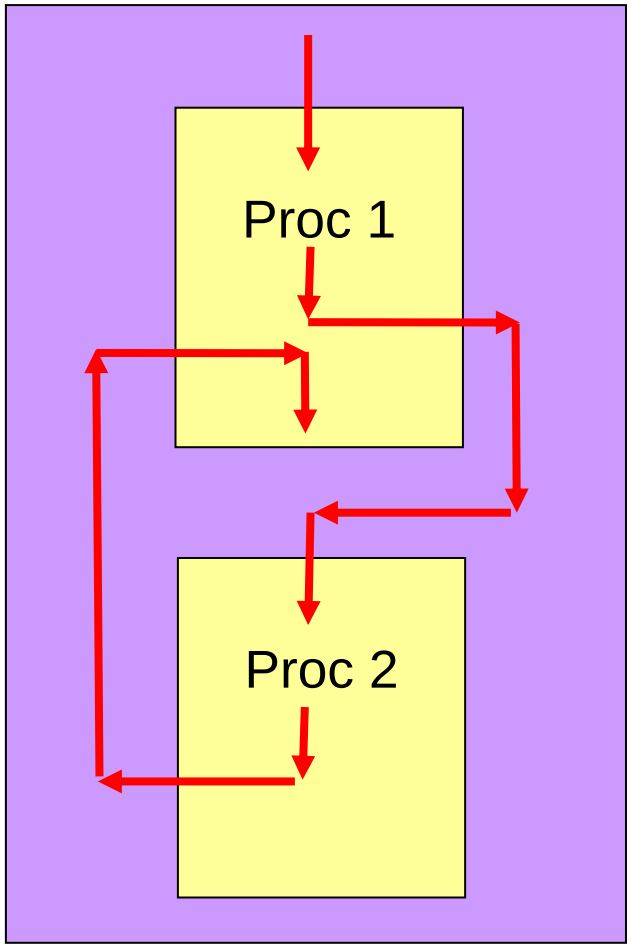






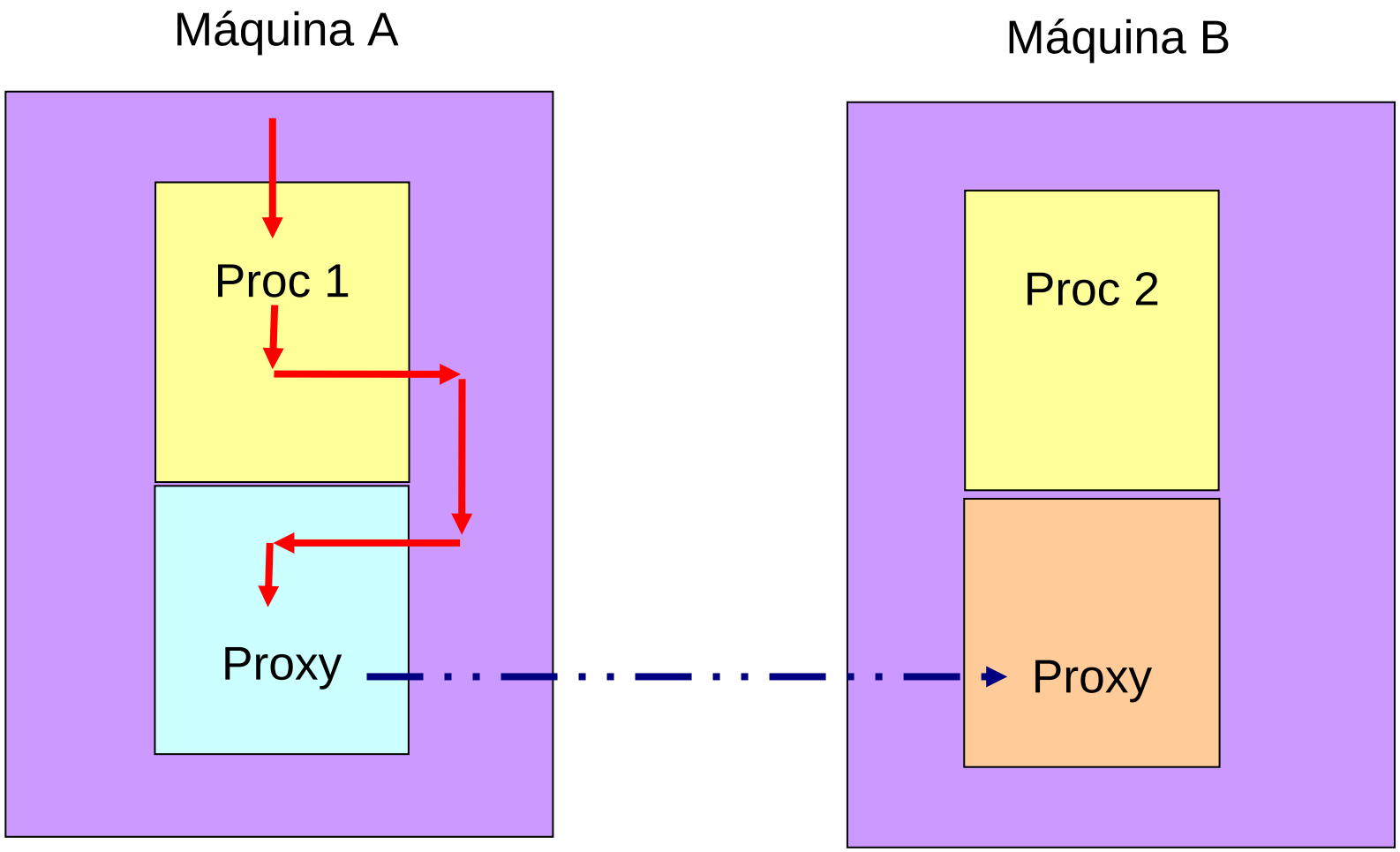
Llamadas a procedimientos locales

Máquina A



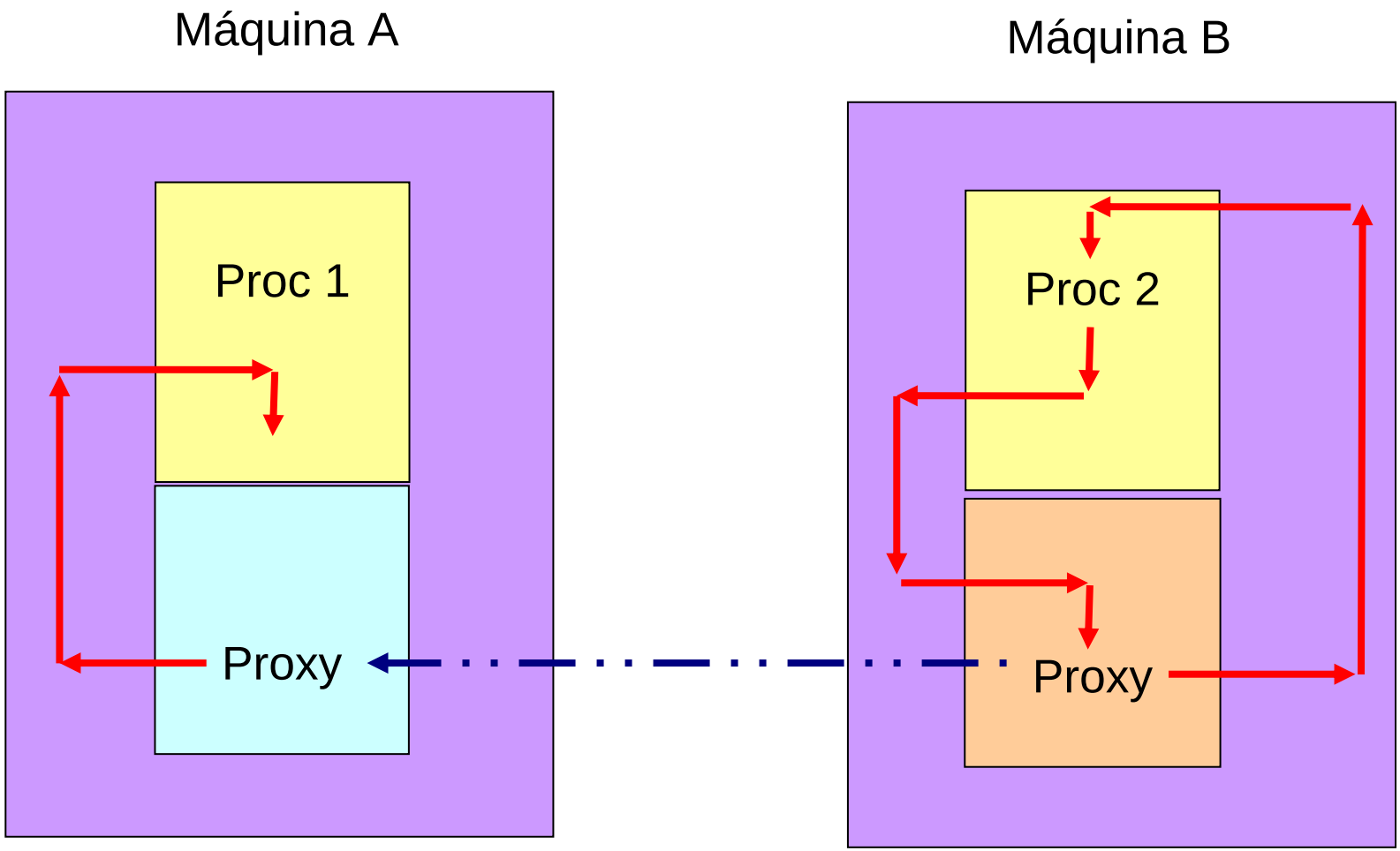
Flujo de ejecución de una llamada a procedimiento local

Llamadas a procedimientos remotos



Flujo de ejecución de una llamada a procedimiento remoto

Llamadas a procedimientos remotos



Flujo de ejecución de una llamada a procedimiento remoto

Invocación a Métodos Remotos

- La invocación a métodos remotos (Remote Method Invocation – RMI) es una implementación del modelo de llamadas a procedimiento remoto orientadas a objeto.
- Permite trabajar con el paradigma de objetos distribuidos pero sólo con el lenguaje Java.

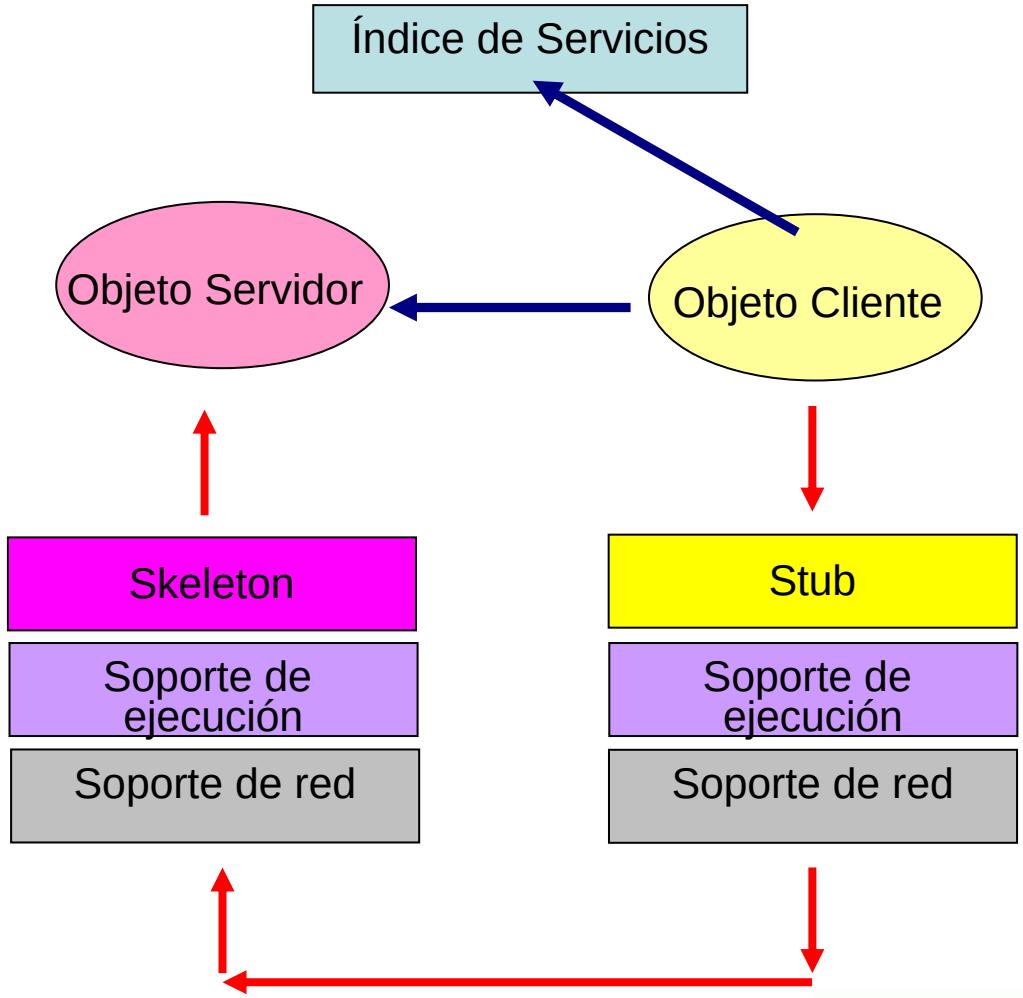
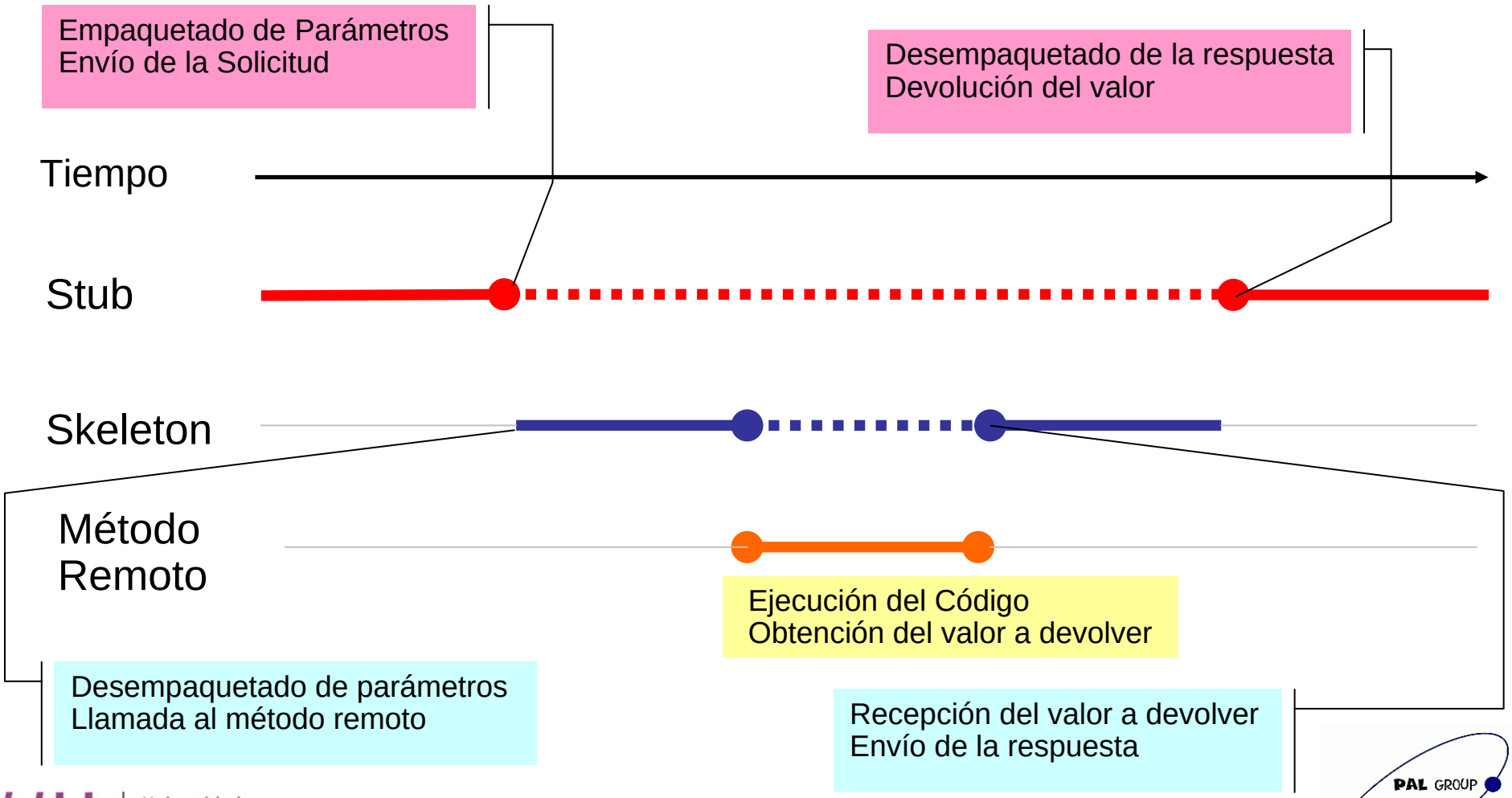


Diagrama de Eventos

- Interacción entre el proxy del Cliente (Stub) y el proxy del Servidor (Skeleton)



Prácticas de Laboratorio

- “Hola Mundo”
- Operaciones Vectoriales
- Calculadora

Simples

- Inventario
- Técnicas Algorítmicas

Complejos

Definición del Interfaz Remoto: *Arith.java*

```
import java.rmi.*;

public interface Arith extends Remote {

 int[] add(int a[], int b[])
                    throws RemoteException;
}
```

Implementación del Interfaz Remoto: *ArithImpl.java*

```
import java.rmi.*;
import java.rmi.server.*;

public class ArithImpl extends UnicastRemoteObject implements Arith {
    private String name;

    public ArithImpl(String s) throws RemoteException {
      super();
      name = s;
    }

    public int[] add(int a[], int b[]) throws RemoteException {
      int c[] = new int[10];
      for (int i=0; i<10; i++)
        c[i] = a[i] + b[i];
      return c;
    }
}
```

Implementación del Servidor: *ArithServer.java*

```
import java.rmi.*;
import java.rmi.server.*;

public class ArithServer {
  public static void main(String argv[]) {
    try {
      System.out.println(“...servidor...");
      ArithImpl obj = new ArithImpl("ArithServer");
      Naming.rebind("ArithServer", obj);
    } catch (Exception e) {
        System.out.println("ArithImpl err: " +
                           e.getMessage());
    }
  }
}
```

Implementación del Cliente: *ArithClient.java*

```
import java.rmi.*;

public class ArithClient {
  public static void main(String argv[]) {
    System.setProperty("java.security.policy", "client.policy");
    System.setSecurityManager(new RMISecurityManager());

    int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int b[] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    int result[] = new int[10];
    try {
      String name = "rmi://localhost/ArithServer";
      Arith obj = (Arith)Naming.lookup(name);
      result = obj.add(a, b);
    } catch (Exception e) {
      System.out.println("ArithApp exception:"+ e.getMessage());
    }
   System.out.println("La suma es: ");
   for (int j=0; j<10; j++) System.out.print(result[j]+"    ");
   System.out.println();
} }
```

Pasos para probar el ejemplo

1) Compilar los ficheros fuente de la clase interfaz, implementación de la interfaz, servidor y cliente:

```
javac Arith*.java
```

2) Mediante el compilador de RMI ***rmic*** procesar la implementación de la interfaz para generar los ficheros *proxy* del objeto remoto:

```
rmic -v1.2 ArithImpl
```

Aparece en el directorio el fichero ***ArithImpl_Stub.class***.

3) Inicializar el registro de nombres RMI con:

```
rmiregistry &
```

4) Lanzar el servidor:

```
java ArithServer &
```

5) Ejecutar el cliente:

```
java ArithClient
```

Se debe encontrar en el directorio el fichero ***Arith.class*** y ***ArithImpl_Stub.class.***

Distribución de los ficheros

Máquina A
Servidor

Directorio

- Arith.class
- ArithImpl.class
- ArithImpl_Stub.class
- ArithServer.class

Máquina B
Cliente

Directorio

- Arith.class
- ArithImpl_Stub.class
- ArithClient.class

Diagrama de Secuencia: Suma Vectorial

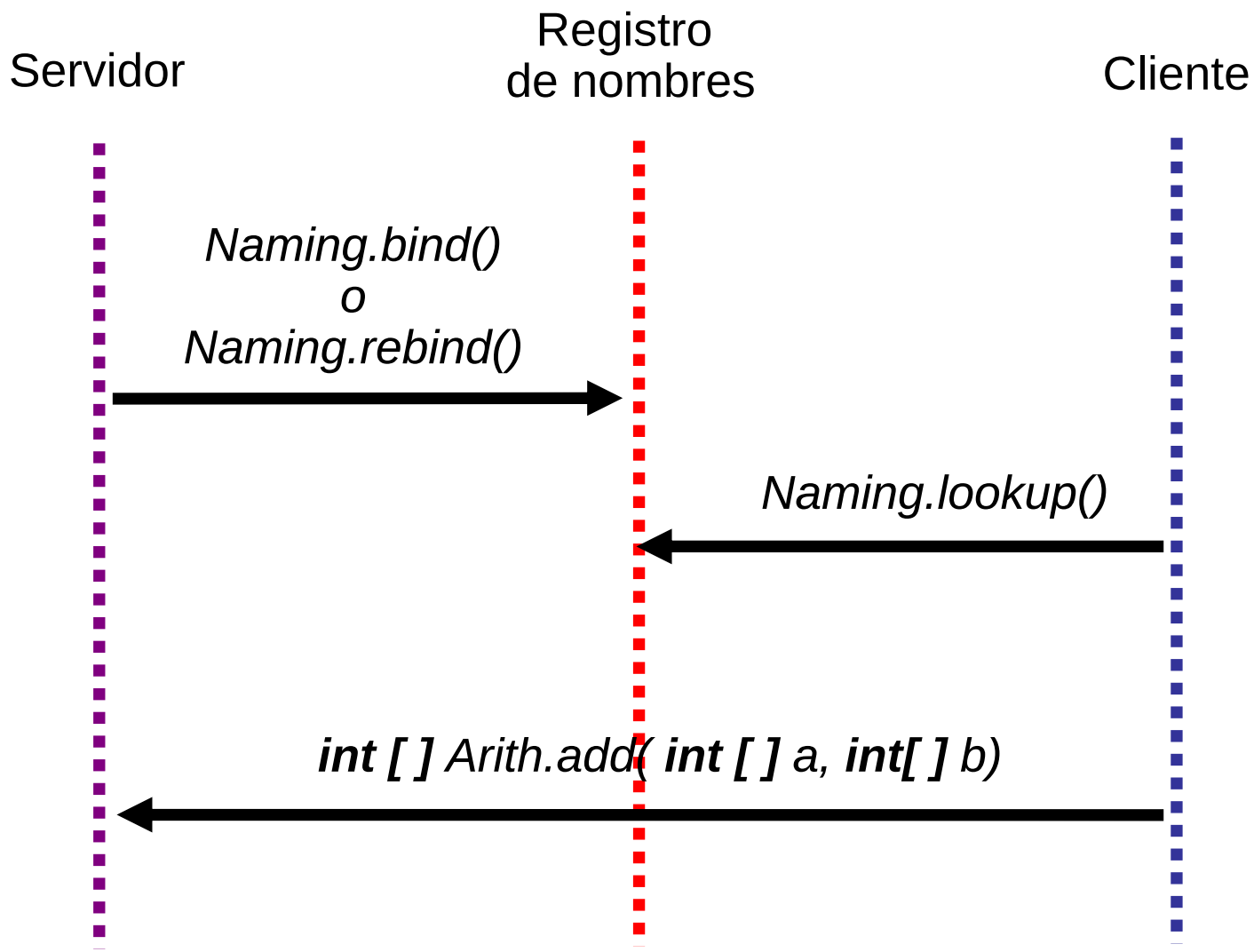


Diagrama UML: Ramificación y Acotación

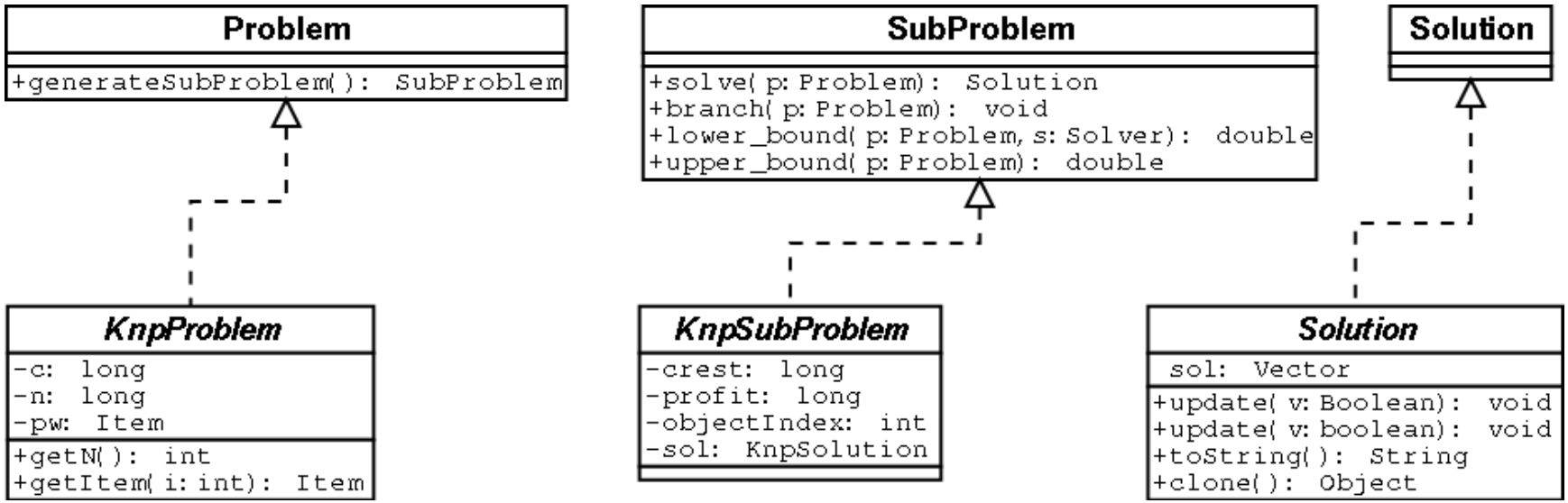
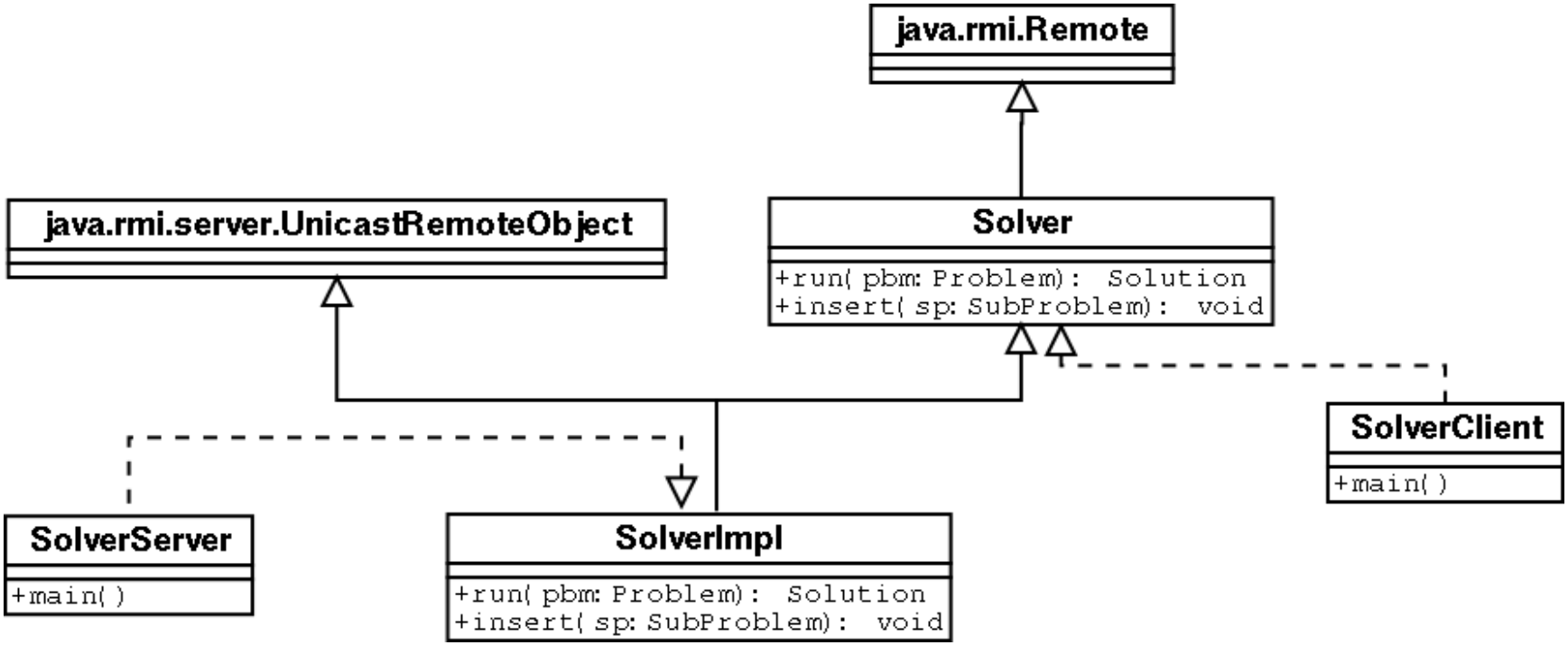


Diagrama UML: Técnicas Algorítmicas



Solver.java

```
public interface Solver extends java.rmi.Remote {
    Solution run(Problem pbm)
            throws java.rmi.RemoteException;
    void insert(SubProblem sp)
            throws java.rmi.RemoteException;
}
```

SolverImpl.java

```
public class SolverImpl
                extends UnicastRemoteObject
                implements Solver {
  String name;
  Stack spList;

  public SolverImpl(String name) throws RemoteException {
    super();
    this.name = name;
    spList = new Stack();
  }

  public void insert(SubProblem sp) throws RemoteException {
    spList.push(sp);
  }

                        ...
```

SolverImpl.java

```
public Solution run(Problem pbm) throws RemoteException {
  double bestActual = -1;
  SubProblem actualsp;
  Solution sol = null;
  spList.push(pbm.generateSubProblem());
  while(!spList.empty()){
    double lower, upper;
    actualsp = (SubProblem)spList.pop();
    if ((upper = actualsp.upper_bound(pbm)) > bestActual){
      if ((lower = actualsp.lower_bound(pbm)) > bestActual){
        bestActual = lower;
        sol = actualsp.solve(pbm);
      }
      if (upper != lower) actualsp.branch(pbm, this);
    }
  }
  return sol;
}
```

SolverServer.java

```
import java.rmi.*;
import java.rmi.server.*;

public class SolverServer {
  public static void main(String argv[]) {
    try {
      System.out.println(“...servidor...");
      SolverImpl obj =
                  new SolverImpl(“RyA");
      Naming.rebind(“SolverServer", obj);
    } catch (Exception e) {
        System.out.println(“SolverImpl err: " +
                           e.getMessage());
} } }
```

SolverClient.java

```
import java.rmi.*;
public class SolverClient {
  public static void main(String argv[]) {
   long C = 104;
   int N = 8;
   long w[] = { 25,  35,  45,  5, 25, 3, 2, 2};
   long p[] = {350, 400, 450, 20, 70, 8, 5, 5};
   KnpProblem pbm = new KnpProblem(C, N, w, p);
   KnpSolution sol;
   try {
      String name = “rmi://localhost/SolverServer";
      Solver obj = (Solver)Naming.lookup(name);
      sol = (KnpSolution)obj.run(pbm);
      System.out.println("La solución es: " + sol);
   } catch (Exception e) {
      System.out.println("SolverClient exception:"+e.getMessage());
      e.printStackTrace();
   }
  }
}
```

Distribución de los ficheros

Máquina A
Servidor

Directorio

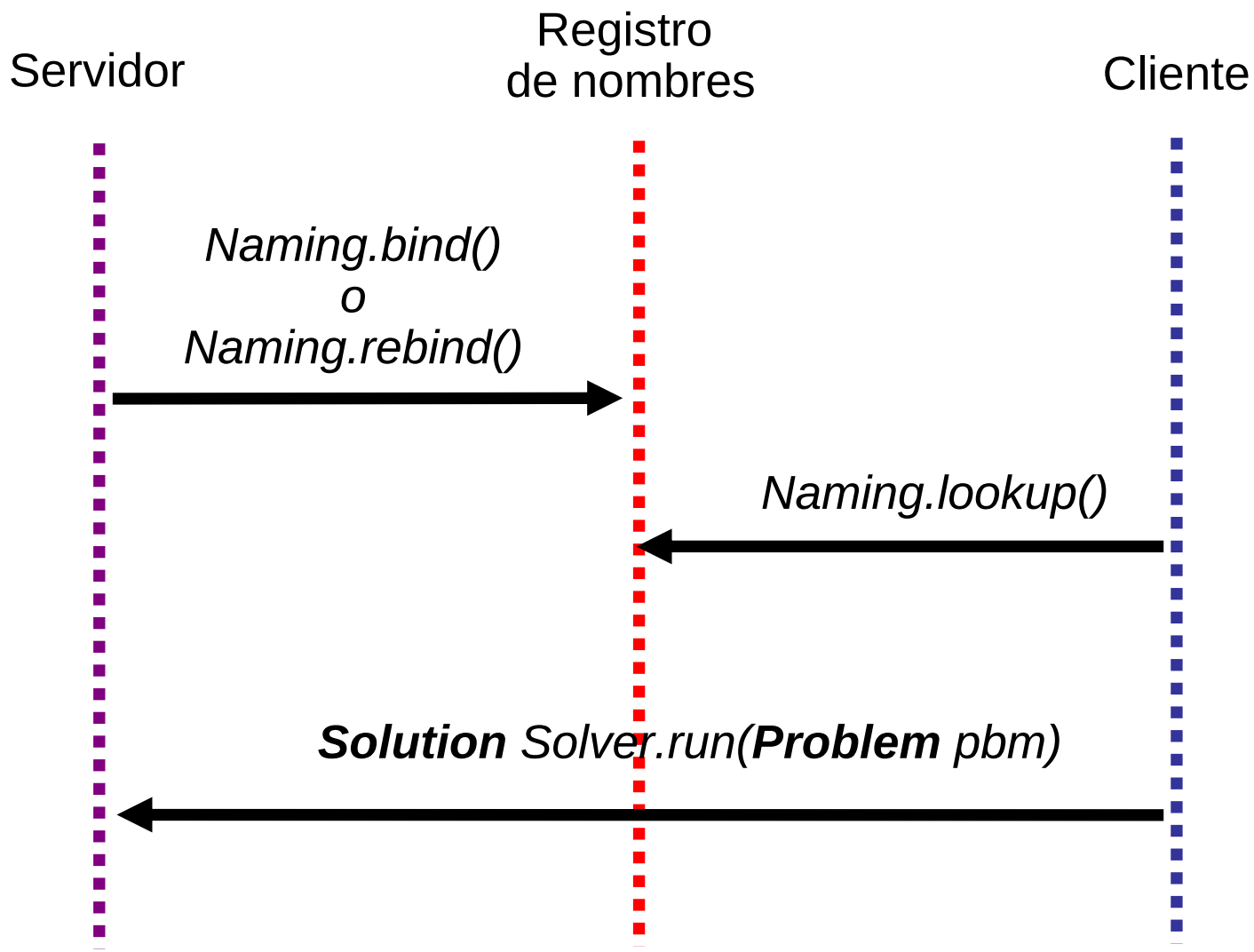
- Solver.class
- SolverImpl.class
- SolverImpl_Stub.class
- SolverServer.class
- Problem.class
- SubProblem.class
- Solution.class

Máquina B
Cliente

Directorio

- Solver.class
- SolverImpl_Stub.class
- SolverClient.class
- KnpProblem.class
- KnpSubProblem.class
- KnpSolution.class

Diagrama de Secuencia: Técnicas Algorítmicas



Ejercicios

- Se proporciona una implementación local para la técnica de Ramificación y Acotación y el problema de la Mochila.
- ¿Cuánto tiempo consume la implementación local?
- Implementar una aplicación Cliente/Servidor usando Java RMI en la cual el servicio que ofrece el objeto remoto sea el mismo que el del programa proporcionado.
- ¿Cuánto tiempo consume la implementación distribuida?
- ¿Qué conclusión puede dar sobre los dos tipos de implementación?

Conclusiones

- Se ha presentado un conjunto de prácticas de laboratorio para cubrir el tema de Invocación a Métodos Remotos en una asignatura con contenidos de Programación de Sistemas Distribuidos.
- La principal novedad es la propuesta de implementación de un servicio que proporciona la posibilidad de resolver problemas utilizando una técnica algorítmica dada, siempre que el usuario especifique el problema utilizando un formato determinado.

Resultados de la Encuesta

Práctica	*Nada*	*Poco*	*Bien*	*Bastante*	*Mucho*	*%interés*
Creación de Threads			25%	67%	8%	100%
Sincronización de Threads		4%	22%	65%	9%	96%
Direcciones IP y Nombres de dominio		8%	35%	22%	35%	92%
Serialización de objetos Java		13%	30%	48%	9%	87%
Las clases Java DatagramPacket y DatagramSocket		4%	21%	42%	33%	96%
Las clases Java Socket y ServerSocket			16%	42%	42%	100%
Comunicación en grupos		13%	25%	29%	33%	87%
Invocación de métodos remotos (Java – RMI)	4%		4%	29%	63%	96%
CORBA (Java-IDL)			13%	29%	58%	100%
Servicio de resolución mediante Técnicas Algorítmicas		8%	33%	34%	25%	92%

Para más Información

- http://pal.pcg.ull.es
  - Enlace de Software
- Contacto
  - gmiranda@ull.es