CAPÍTULO 4: GENERACIÓN DE SERVICIOS EN RED

Programación de Servicios y Procesos

ÍNDICE

- Servicios
- Programación de aplicaciones cliente y servidor
- Protocolos estándar de nivel de aplicación
- Técnicas avanzadas de programación de aplicaciones distribuidas

Servicios

- Todo sistema está formado por dos partes fundamentales:
  - Estructura: Componentes hardware y software que lo forman.
  - Función: Aquello para lo que está pensado el sistema, es decir, para lo qué sirve y/o se usa.

Alberto Sánchez Campos
Jesús Montes Sánchez

Concepto de servicio

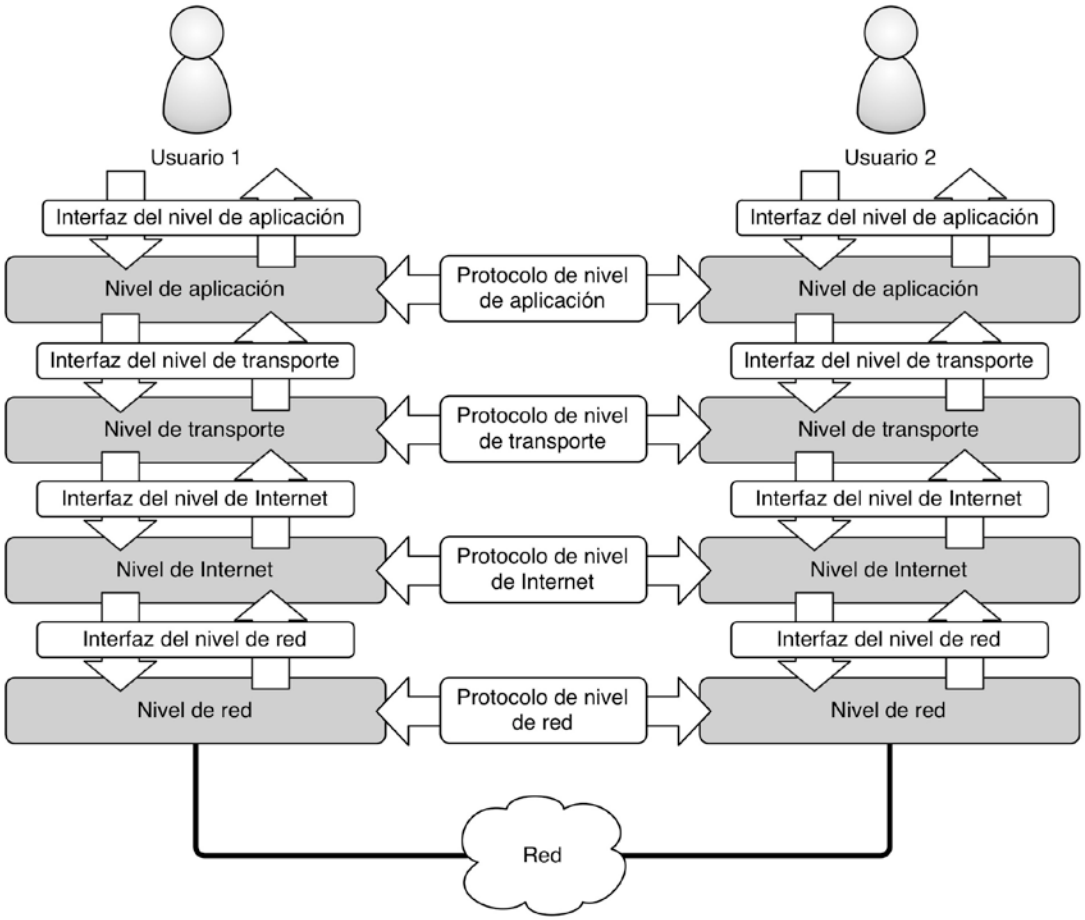
- El **servicio** de un sistema es el conjunto de mecanismos concretos que hacen posible la su función.
- Los usuarios interactúan con el sistema y hacen uso de sus servicios a través de la **interfaz del servicio**.

Servicios en red

- La pila de protocolos IP es un conjunto de sistemas independientes, montados unos sobre otros para realizar una tarea compleja.
- Cada nivel de la jerarquía (red, Internet, transporte y aplicación) proporciona un servicio especifico y ofrece una interfaz de servicio al nivel superior.
- Cada nivel de la pila dispone de su propio protocolo de comunicaciones, que gobierna la interacción a ese nivel con los demás elementos del sistema distribuido.

Alberto Sánchez Campos
Jesús Montes Sánchez

Servicios en red



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Servicios de nivel de aplicación

- El nivel más alto de la pila IP lo componen las aplicaciones que forman el sistema distribuido.
- Un **protocolo de nivel de aplicación** es el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida.

Programación de aplicaciones cliente y servidor

- A la hora de programar una aplicación siguiendo el modelo cliente/servidor, se deben definir de forma precisa los siguientes aspectos:
  - Funciones del servidor.
  - Tecnología de comunicaciones.
  - Protocolo de nivel de aplicación.

Funciones del servidor

- Definir de forma clara qué hace el *servidor*:
  - ¿Cuál es la función básica de nuestro servidor?
  - El servicio que proporciona nuestro servidor, ¿es rápido o lento?
  - El servicio que proporciona nuestro servidor, ¿puede resolverse con una simple petición y respuesta o requiere del intercambio de múltiples mensajes entre este y el cliente?
  - ¿Debe ser capaz nuestro servidor de atender a varios clientes simultáneamente?
  - Etc.

Tecnología de comunicaciones

- Se debe escoger la tecnología de comunicaciones que es necesario utilizar.
- Los dos mecanismos básicos de comunicación que se han estudiado son los *sockets stream* y los *sockets datagram*.
- Cada tipo de *socket* presenta una serie de ventajas e inconvenientes.
- Es necesario escoger el tipo de *socket* adecuado, teniendo en cuenta las características del servicio que proporciona nuestro servidor.
  - Los *sockets stream* son más fiables y orientados a conexión, por lo que se deben usar en aplicaciones complejas en las que clientes y servidores intercambian muchos mensajes.
  - Los *sockets datagram* son menos fiables, pero más eficientes, por lo que es preferible usarlos cuando las aplicaciones son sencillas, y no es problema que se pierda algún mensaje.

Definición del protocolo de nivel de aplicación

- Un **protocolo de nivel de aplicación** es el conjunto de reglas que gobiernan la interacción entre los diferentes elementos de una aplicación distribuida.
- Se debe definir explícitamente el formato de los mensajes que se intercambian entre cliente y servidor, así como las posibles secuencias en las que estos pueden ser enviados y recibidos.
- La definición del protocolo de nivel de aplicación debe contener todos los posibles tipos de mensajes (tanto peticiones como respuestas) que pueden ser enviados y recibidos, indicando cuándo se puede enviar cada uno y cuándo no.

Alberto Sánchez Campos
Jesús Montes Sánchez

Protocolos sin estado y con estado

- Protocolos sin estado (*stateless*): son aquellos en los que la secuencia concreta en la que se reciben los mensajes no es importante, ya que no afecta al resultado de estos. El servidor se limita a recibir las peticiones del cliente y a responderlas una por una, de forma independiente.
- Protocolos con estado (*stateful*): son aquellos en los que la secuencia concreta en la que se reciben los mensajes es importante, ya que afecta al resultado final de las peticiones. En estos casos, el servidor debe almacenar información intermedia durante la sesión, denominada el *estado de la sesión*. Conocer este estado es imprescindible para poder resolver las peticiones de manera correcta.

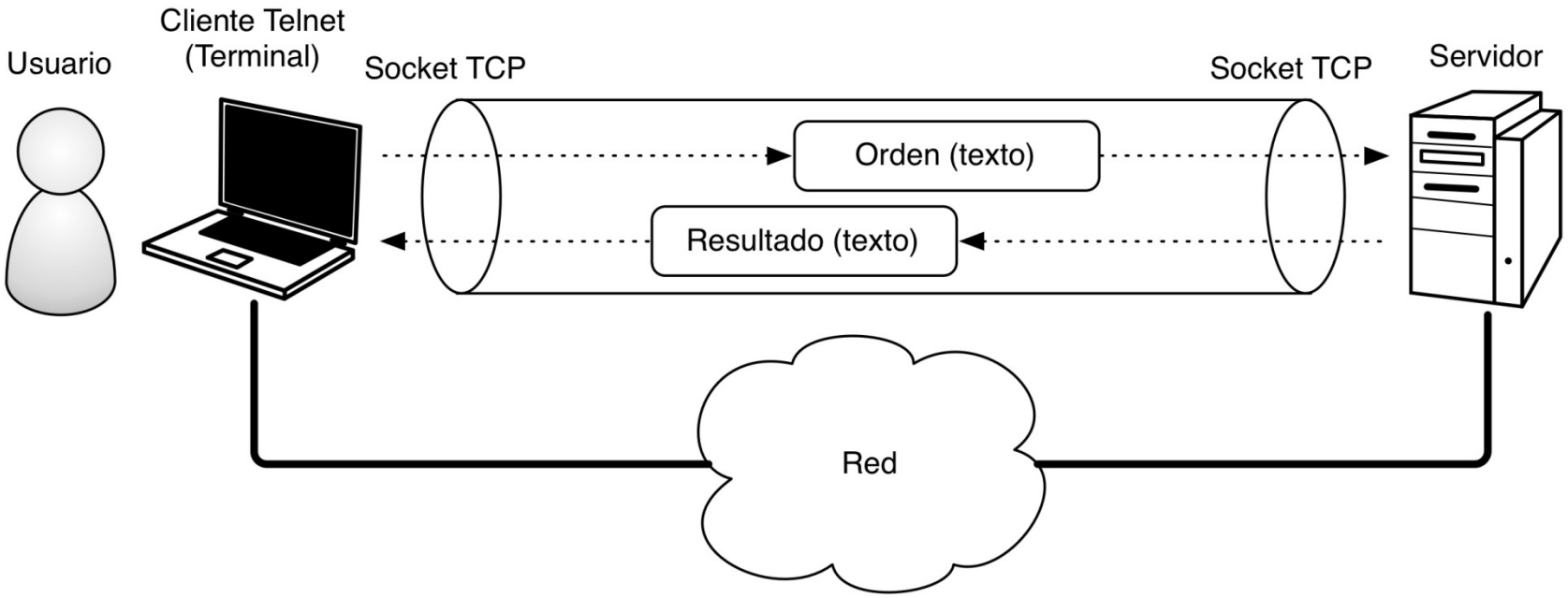
Protocolos estándar de nivel de aplicación

- En muchos casos el cliente y el servidor de sistemas distribuidos modernos son aplicaciones independientes, desarrolladas por diferentes personas/compañías y muchas veces ni siquiera implementadas usando el mismo lenguaje de programación.
- La definición exhaustiva de un protocolo de nivel de aplicación es fundamental en estos casos para garantizar que clientes y servidores pueden comunicarse de manera efectiva.
- Para la mayoría de aplicaciones distribuidas comunes se han definido, a lo largo de los anos, protocolos de nivel de aplicación estándar, que facilitan la tarea de desarrollar servidores y clientes para ellas.

Telnet

- Protocolo de nivel de aplicación diseñado para proporcionar comunicación bidireccional basada en texto plano (caracteres ASCII) entre dos elementos de una red.
- Telnet se ha usado para conectarse remotamente a maquinas, creando una sesión de línea de mandatos como las *Shell* de los sistemas operativos UNIX o el “símbolo de sistema” de Windows (C:\>).
- Es un protocolo con estado (*stateful*).
- Usa por defecto el puerto 23.

Telnet



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

SSH (Secure Shell)

- Un protocolo de nivel de aplicación muy similar a Telnet.
- Se usa par establecer sesiones de línea de mandatos con servidores remotos.
- Incorpora sistemas para hacer que la comunicación sea segura, por lo que es un alternativa muy recomendable a Telnet.
- Es un protocolo con estado (*stateful*).
- Usa por defecto el puerto 22.

Alberto Sánchez Campos
Jesús Montes Sánchez

FTP (File Transfer Protocol)

- Protocolo de nivel de aplicación diseñado para la transferencia de archivos a través de una red de comunicaciones.
- FTP establece un par de conexiones simultáneas entre cliente y servidor, que usa de forma distinta.
  - Conexión de control: Se usa para enviar órdenes al servidor y obtener información.
  - Conexión de datos: Se utiliza para transferir los archivos.
- FTP permite tanto la descarga de archivos (del servidor al cliente) como la subida de estos (del cliente al servidor).
- Es un protocolo con estado (*stateful*).
- Usa por defecto el puerto 21.

Alberto Sánchez Campos
Jesús Montes Sánchez

HTTP (Hypertext Transfer Protocol)

- **HTTP** es, probablemente, el protocolo de nivel de aplicación más importante de la actualidad.
- La mayoría del tráfico que se realiza en la *World Wide Web*, la parte más visible de la red Internet, utiliza este protocolo para controlar la transferencia de información.

HTTP (Hypertext Transfer Protocol)

- HTTP está diseñado para facilitar la trasferencia de documentos de *hipertexto*.
  - Documentos de texto expresados en un código especial.
  - Contiene mecanismos que permiten formatear el texto (tipografía, imágenes, etc.).
  - Contiene *hiperenlaces* (*hiperlinks*) a otros documentos.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 80.

HTTP (Hypertext Transfer Protocol)

- Un servidor HTTP organiza la información que contiene (fundamentalmente documentos de *hipertexto*, es decir, páginas web) usando *recursos*.
- Los clientes pueden realizar diferentes peticiones al servidor:
  - GET: Obtener un recurso.
  - POST: Modificar un recurso.
  - PUT: Crear un recurso.
  - DELETE: Eliminar un recurso.
  - etc.

HTTP (Hypertext Transfer Protocol)

- Las respuestas de un servidor HTTP contienen una cabecera y un cuerpo de mensaje.
  - Cabecera: Información sobre el servidor y estado de la operación solicitada.
  - Cuerpo: Contenido de la respuesta, normalmente un documento de *hipertexto*.
- La cabecera de una respuesta siempre contiene un código (número) de estado:
  - Información (códigos que empiezan por 1)
  - Estado de éxito (códigos que empiezan por 2)
  - Redirección (códigos que empiezan por 3)
  - Error del cliente (códigos que empiezan por 4)
  - Error del servidor (códigos que empiezan por 5)

POP3 (Post Office Protocol ver.3)

- Lo usan las aplicaciones de correo electrónico para acceder a los mensajes alojados en servidores.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 110.

SMTP (Simple Mail Transfer Protocol)

- Es el protocolo estándar para la tranferencia de correo electrónico.
- Lo usan todos los servidores de e-mail de Internet.
- Se usa también para enviar mensajes desde cliente de correo locales, como Thunderbird o Outlook.
- Es un protocolo sin estado (*stateless*).
- Usa por defecto el puerto 587.

Alberto Sánchez Campos
Jesús Montes Sánchez

Otros protocolos de nivel de aplicación importantes

- DHCP (*Dynamic Host Configuration Protocol*): Para configurar máquinas dentro de una red.
- DNS (*Nomain Name Service*): Para localizar direcciones IP a partir de nombre simbólicos.
- NTP (*Network Time Protocol*): Para sincronizar relojes entre máquinas.
- TLS (*Transport Layer Security*) y SSL *(Secure Socket Layer*): Para proporcionar comunicaciones seguras.

Alberto Sánchez Campos
Jesús Montes Sánchez

Técnicas avanzadas de programación distribuida

- Los *sockets stream* y *sockets datagram* son la base de la mayoría de mecanismos de comunicación entre aplicaciones.
- No obstante, las interfaces de programación que normalmente se usan para operar con resultan poco practicas, ya que requieren el formateo completo de los mensajes y el control detallado de las comunicaciones.

Técnicas avanzadas de programación distribuida

- A la hora de programar aplicaciones mas sofisticadas se hace uso de técnicas mas avanzadas.
- Estas técnicas internamente hacen uso de *sockets*, pero los abstraen utilizando mecanismos de mas alto nivel, que incorporan funcionalidades adicionales.
- Ejemplos de estas técnicas son:
  - Invocación de métodos remotos.
  - Servicios Web.

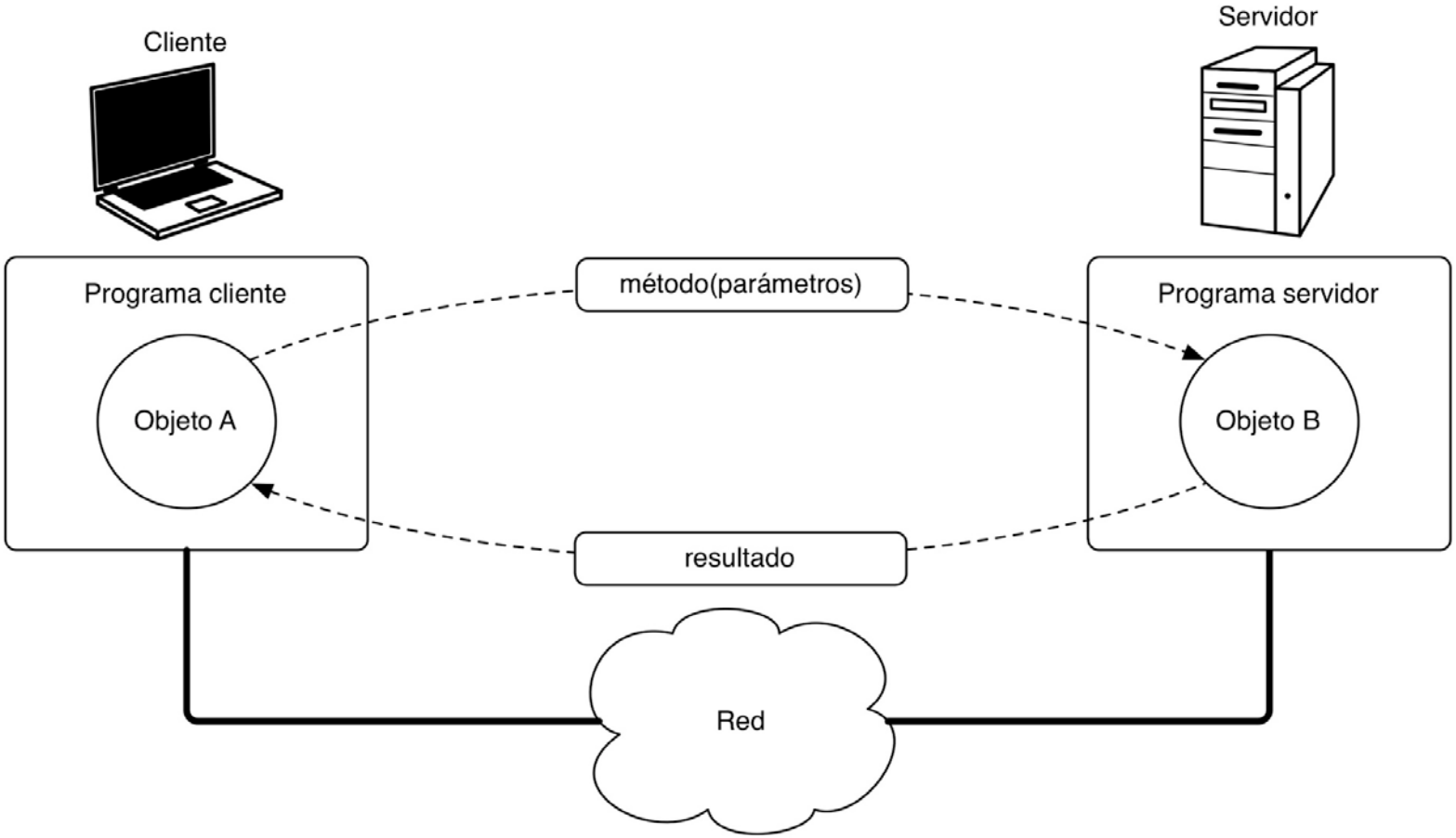
Alberto Sánchez Campos
Jesús Montes Sánchez

Invocación de métodos remotos

- En un programa orientado a objetos la invocación de un método se puede ver como un proceso de comunicación:
  - El objeto A envía un mensaje al B a modo de petición (invocación del método).
  - El objeto B procesa la petición (ejecución del método) contesta con un mensaje de respuesta (valor de retorno del método).
- La **invocación de métodos remotos** funciona de la misma forma, pero cada objeto (A y B) se encuentran e máquinas distintas.

Alberto Sánchez Campos
Jesús Montes Sánchez

Invocación de métodos remotos



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Invocación de métodos remotos

- Objeto servidor: El objeto cuyo método es invocado. También se le llama **objeto remoto**.
- Objeto cliente: El objeto que realiza la petición.
- Método invocado: El mensaje de petición, incluyendo los parámetros del método.
- Valor de retorno: El mensaje de respuesta.

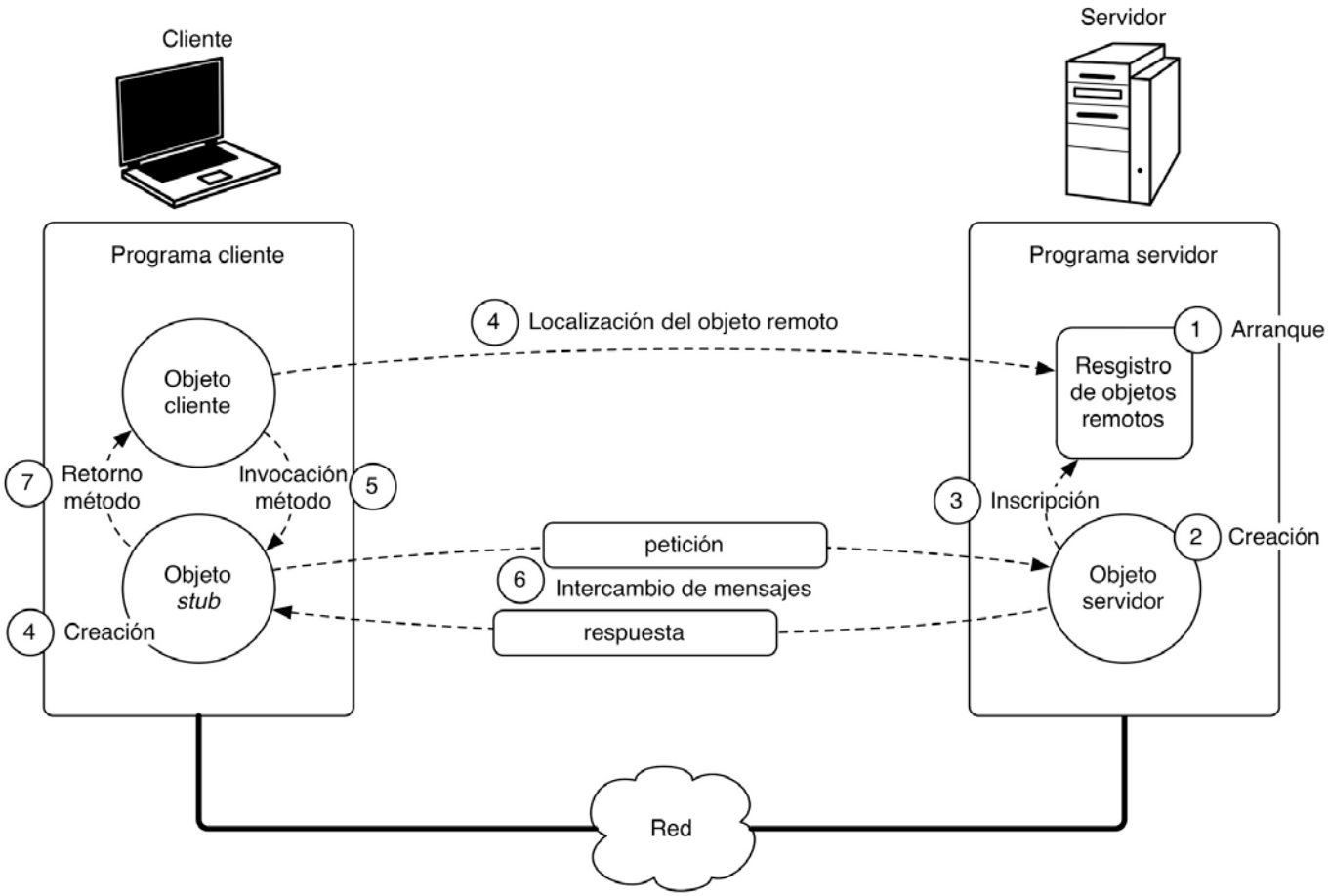
Invocación de métodos remotos

- Para que puedan llevarse a cabo invocaciones de métodos remotos, debe existir una infraestructura que capture las llamadas y genere el tráfico en la red. Esta infraestructura tiene dos componentes:
  - Objetos *stub*: Sustituyen a los objetos remotos en el programa cliente. Convierten las invocaciones de métodos en mensajes y los transfieren por la red.
  - Registro de objetos remotos: Es un servicio de nivel de aplicación que sirve para poder localizar objetos remotos.

Invocación de métodos remotos

1. (servidor) Arranque del registro de objetos remotos
2. (servidor) Creación del objeto servidor.
3. (servidor) Inscripción del objeto servidor en el registro.
4. (cliente) Localización del objeto remoto y creación del objeto *stub*.
5. (cliente) Invocación del método del objeto *stub*.
6. Intercambio de mensajes entre servidor y cliente.
7. (cliente) Obtención del valor de retorno del método.

Invocación de métodos remotos



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Programación de aplicaciones basadas en métodos remotos

- Como en toda aplicación cliente/servidor, se deben realizar lis siguientes pasos:
  1. Definición de las funciones del servidor.
  2. Selección de la tecnología de comunicaciones. En este caso, invocación de métodos remotos.
  3. Definición del protocolo de nivel de aplicación. Para la invocación de métodos remotos en Java esto implica la programación de la **interfaz remota**.

1. Una vez hecho esto, se implementan las clases del objeto servidor y el objeto cliente.

Interfaz remota

- Interfaz Java que contiene los métodos que se ejecutarán de forma remota.
- Ejemplo:

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

public interface RMICalcInterface extends Remote {

        public int suma(int a, int b) throws RemoteException;
        public int resta(int a, int b) throws RemoteException;
        public int multip(int a, int b) throws RemoteException;
        public int div(int a, int b) throws RemoteException;
}
```

Ejemplo de clase de objetos remotos (1 de 2)

```
import java.rmi.*;

public class RMICalcServer implements RMICalcInterface {

   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 multip(int a, int b) throws RemoteException {
      return (a * b);
   }
   public int div(int a, int b) throws RemoteException {
      return (a / b);
   }
```

Ejemplo de clase de objetos remotos (2 de 2)

```
public static void main(String[] args) {

    Registry reg = null;
    try {
       reg = LocateRegistry.createRegistry(5555);
    } catch (Exception e) {
       System.out.println("ERROR: No se ha podido crear el registro");
       e.printStackTrace();
    }

    RMICalcServer serverObject = new RMICalcServer();

    try {
       reg.rebind("Calculadora”,
                  (RMICalcInterface) UnicastRemoteObject.exportObject(serverObject, 0));
    } catch (Exception e) {
       System.out.println("ERROR: No se ha podido inscribir el objeto servidor.");
       e.printStackTrace();
    }
  }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Ejemplo de clase de cliente de objetos remotos (1 de 2)

```
import java.rmi.*;

public class RMICalcClient {

   public static void main(String[] args) {

      RMICalcInterface calc = null;

      try {

         Registry registry = LocateRegistry.getRegistry("localhost", 5555);
         calc = (RMICalcInterface) registry.lookup("Calculadora");

      } catch (Exception e) {
         e.printStackTrace();
      }
```

Ejemplo de clase de cliente de objetos remotos (2 de 2)

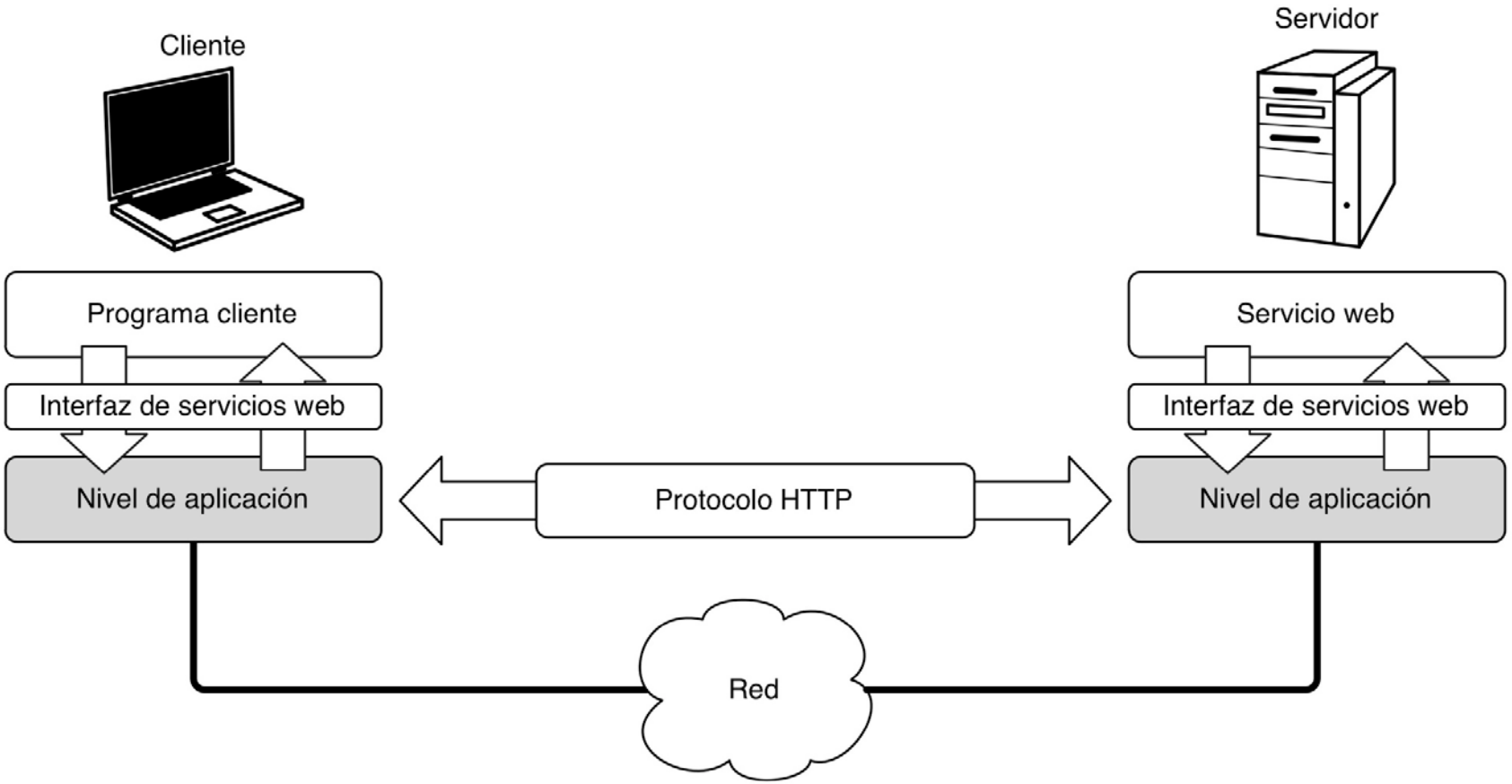
```
        if (calc != null) {

            try {
                System.out.println("2 + 2 = " + calc.suma(2, 2));
                System.out.println("99 - 45 = " + calc.resta(99, 45));
                System.out.println("125 * 3 = " + calc.multip(125, 3));
                System.out.println("1250 / 5 = " + calc.div(1250, 5));

            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Servicios Web

- Un **servicio web** es una aplicación distribuida, basada en el modelo de comunicaciones cliente/servidor, que utiliza el protocolo de nivel de aplicación HTTP para la transferencia de mensajes.
- El uso de este protocolo facilita la interoperabilidad entre clientes y servidores.

Servicios Web



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Servicios Web

- Servicios Web SOAP:
  - Representan los mensajes usando el leguaje SOAP (*Simple Object Access Protocol*), basado en XML.
  - Facilidad de desarrollo e interoperabilidad entre diferentes clientes y servidores.
- Servicios Web REST:
  - Representan los mensajes con REST (*Representational State Transfer*).
  - Menor interoperabilidad que con SOAP, pero mejor rendimiento.