

CAPÍTULO 4: GENERACIÓN DE SERVICIOS EN RED

Curso 20-21

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.



Concepto de servicio

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

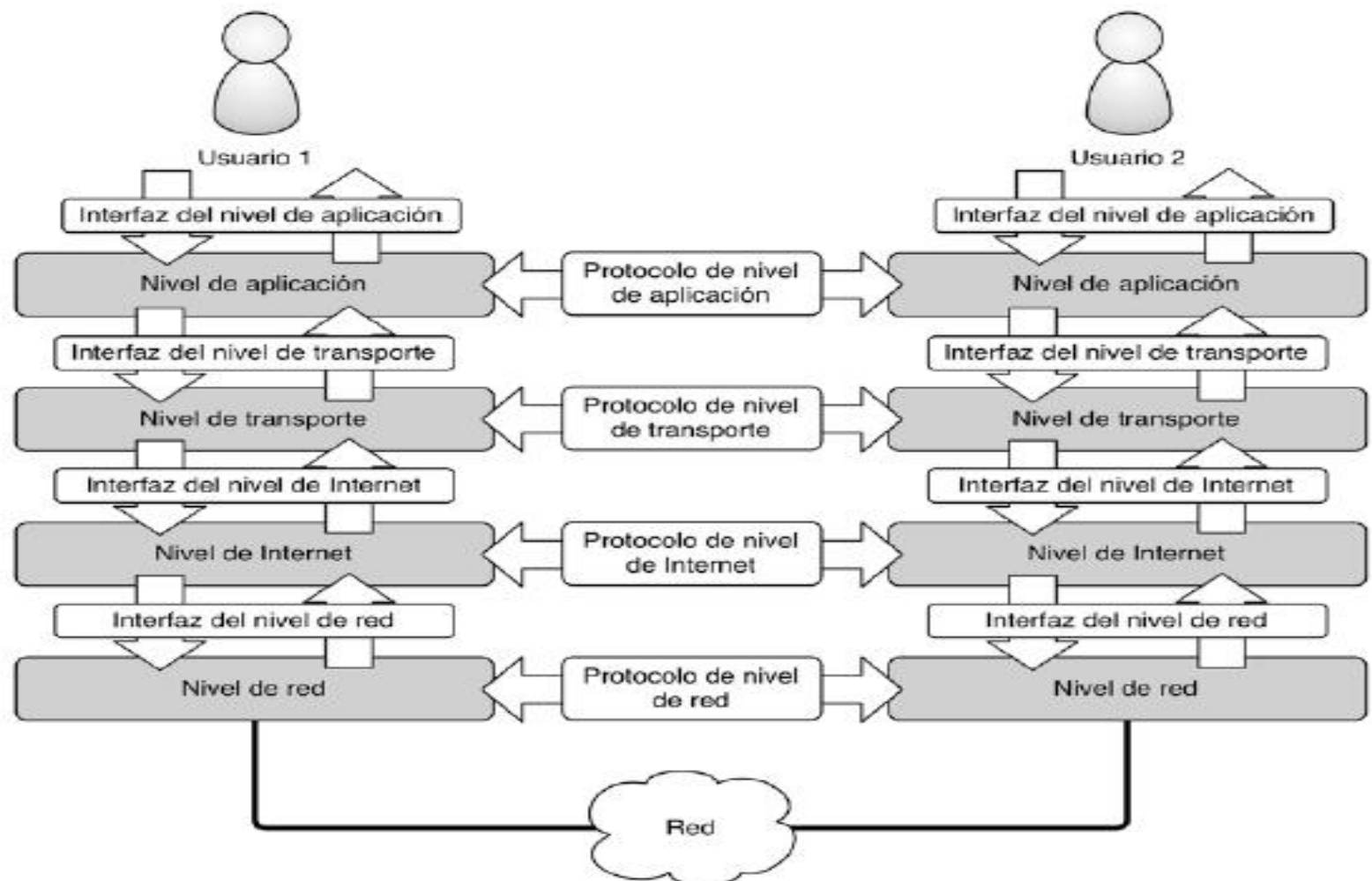


Servicios en red

- ▶ 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 específico 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.



Servicios en red



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.



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 años, protocolos de nivel de aplicación estándar, que facilitan la tarea de desarrollar servidores y clientes para ellas.

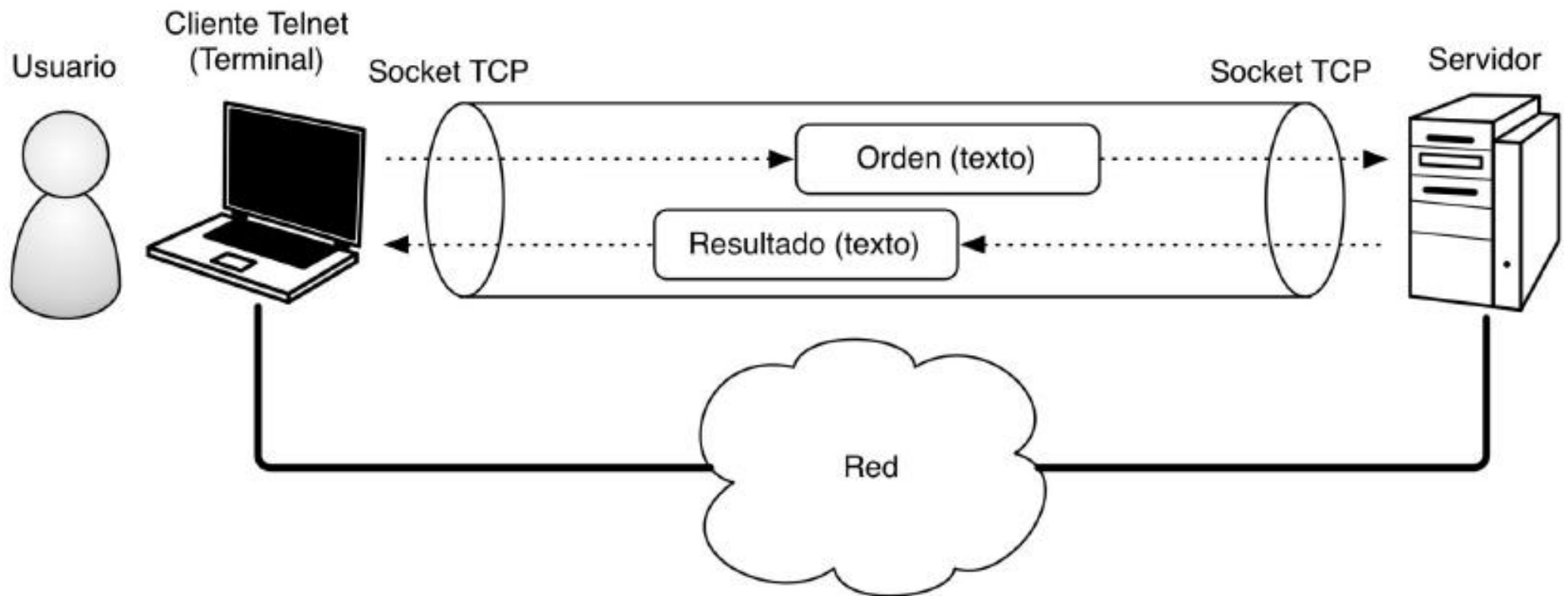


Telnet

- ▶ 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



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.



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.

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 transferencia de documentos de *hipertexto*.
 - Documentos de texto expresados en un código especial.
 - Contiene mecanismos que permiten formatear el texto (tipografía, imágenes, etc.).
- ▶ ○ Contiene *hiperenlaces* (*hyperlinks*) 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 transferencia 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.



Otros protocolos de nivel de aplicación importantes

- ▶ DHCP (*Dynamic Host Configuration Protocol*): Para configurar máquinas dentro de una red.
- ▶ DNS (*Domain 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.



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.

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();  
    }  
}
```

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();  
    }  
}  
}
```

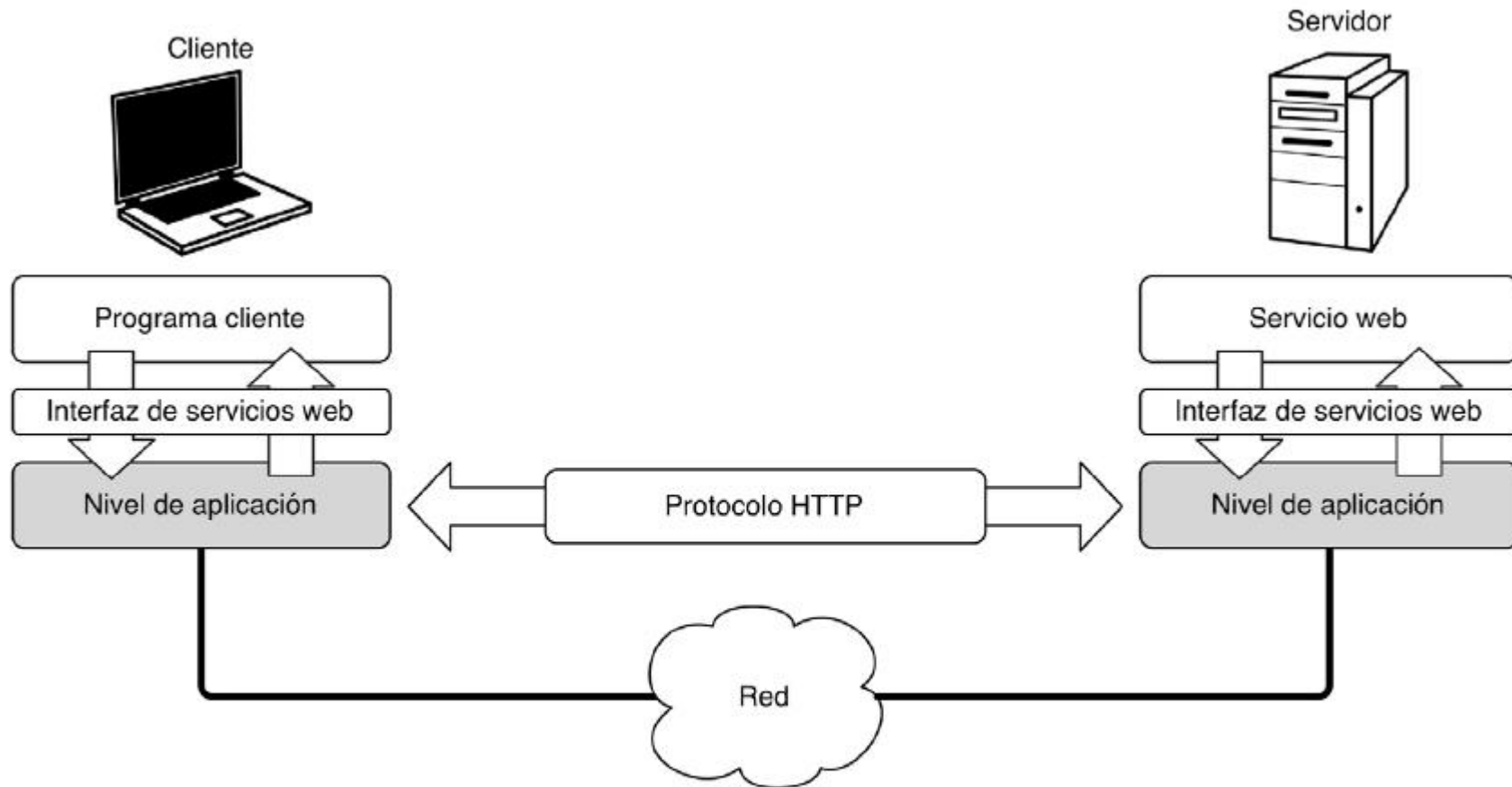


Servicios Web

- ▶ 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



Servicios Web

▶ Servicios Web SOAP:

- Representan los mensajes usando el lenguaje 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.

Guía Breve de Servicios Web

?php include("inc/menu.inc.php")?

¿Qué son los Servicios Web?

Existen múltiples definiciones sobre lo que son los Servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

¿Para qué sirven?

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí

para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

¿Cómo funcionan?

El siguiente gráfico muestra cómo interactúa un conjunto de Servicios Web:

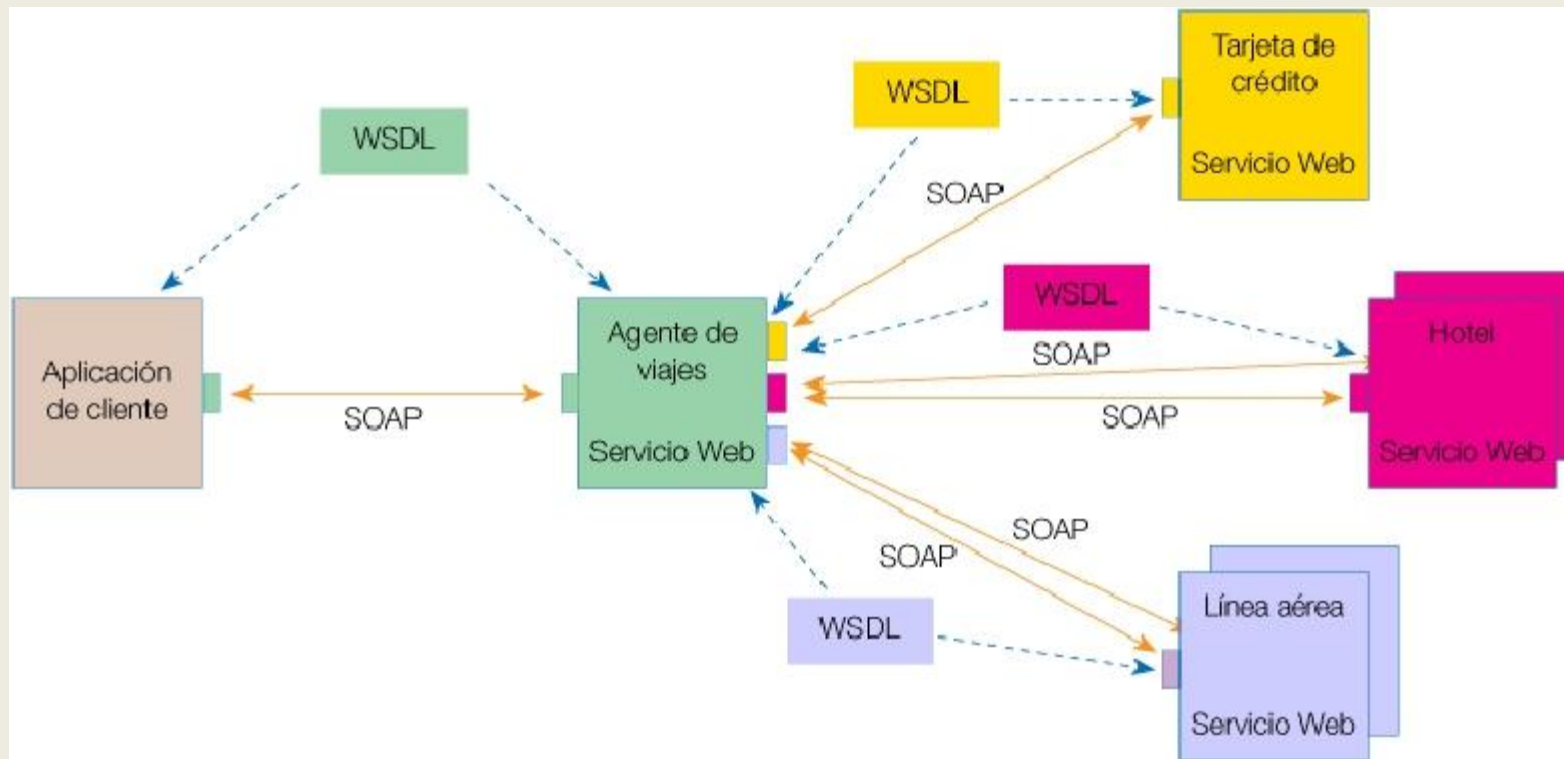


Figura 1 - Los servicios Web en Funcionamiento

Según el ejemplo del gráfico, un usuario (que juega el papel de cliente dentro de los Servicios Web), a través de una aplicación, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus **servicios** a través de Internet. La agencia de viajes ofrecerá a su cliente (usuario) la

información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros Servicios Web) en relación con el hotel y la compañía aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros Servicios Web que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el servicio Web que gestionará el pago.

En todo este proceso intervienen una serie de tecnologías que hacen posible esta circulación de información. Por un lado, estaría SOAP (Protocolo Simple de Acceso a Objetos). Se trata de un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etc. SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un **envelope** (sobre), cuya estructura está formada por los siguientes elementos: **header** (cabecera) y **body** (cuerpo).

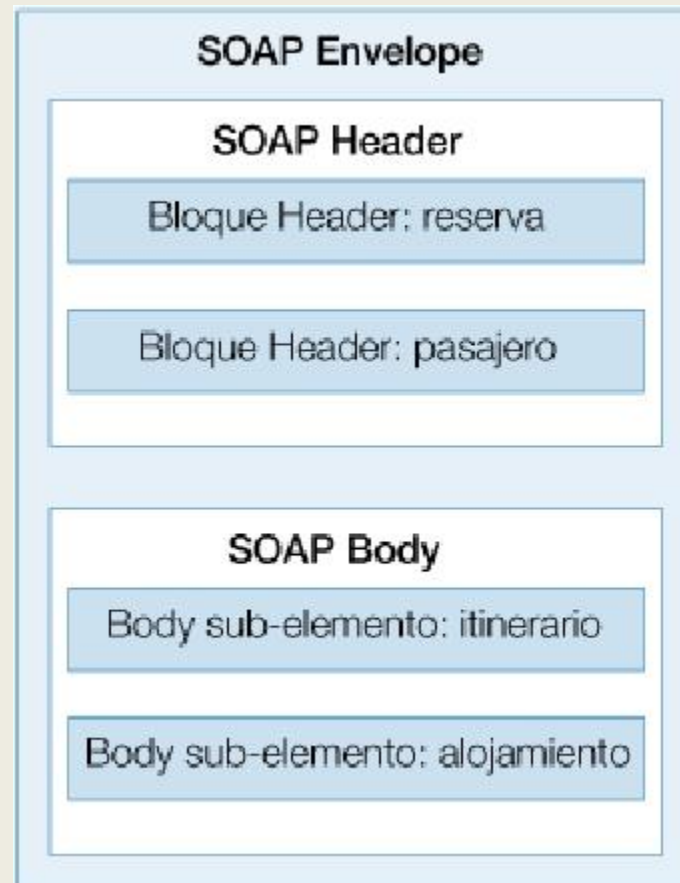


Figura 2 - Estructura de los mensajes

Para optimizar el rendimiento de las aplicaciones basadas en

Servicios Web, se han desarrollado tecnologías complementarias a SOAP, que agilizan el envío de los mensajes (MTOM) y los recursos que se transmiten en esos mensajes (SOAP-RRSHB). Por otro lado, WSDL (Lenguaje de Descripción de Servicios Web), permite que un servicio y un cliente establezcan un acuerdo en lo que se refiere a los detalles de transporte de mensajes y su contenido, a través de un documento procesable por dispositivos. WSDL representa una especie de contrato entre el proveedor y el que solicita. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes.

Durante la evolución de las necesidades de las aplicaciones basadas en Servicios Web de las grandes organizaciones, se han desarrollado mecanismos que permiten enriquecer las descripciones de las operaciones que realizan sus servicios mediante anotaciones semánticas y con directivas que definen el comportamiento. Esto permitiría encontrar los Servicios Web que mejor se adapten a los objetivos deseados. Además, ante la complejidad de los procesos de las grandes aplicaciones empresariales, existe una tecnología que permite una definición de estos procesos mediante la composición de varios Servicios

Web individuales, lo que se conoce como coreografía.

<http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion02-apuntes.html>

<http://www.ooscarr.com/nerd/elblog/2008/12/crear-un-simple-servicio-web-en.php>

[http://programacion.net/articulo/introduccion a los servicios web en java 190](http://programacion.net/articulo/introduccion_a_los_servicios_web_en_java_190)

Introducción a Servicios REST

por **Cecilio Álvarez Caules** sobre 14 junio, 2013 en **Servicios Web**

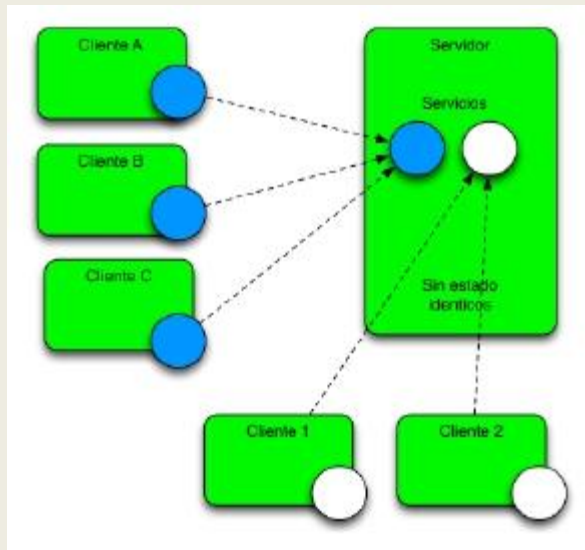
Cada día necesitamos mas usar servicios web REST . Estos servicios se diferencian de una forma importante de los servicios web SOAP con los que hemos trabajado . REST (Representational State Transfer) es un estilo de arquitectura para desarrollar servicios. Los servicios web que siguen este estilo deben cumplir con las siguientes premisas.

Cliente/Servidor : Como servicios web son cliente servidor y definen un interface de comunicación entre ambos separando completamente las responsabilidades entre ambas partes.

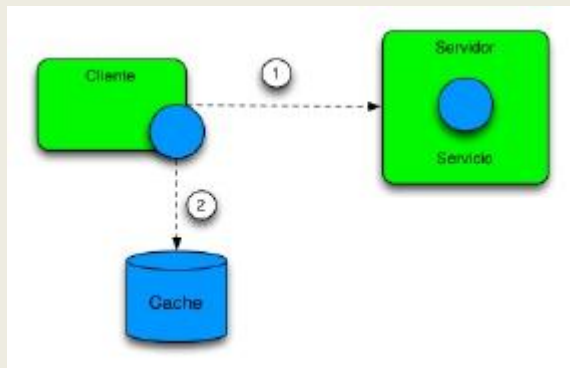


Sin estado : Son servicios web que no mantienen estado asociado al cliente

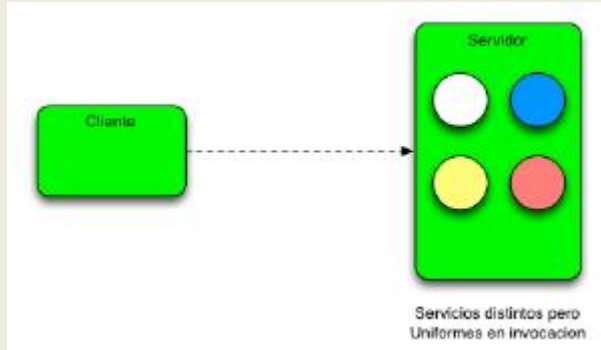
.Cada petición que se realiza a ellos es completamente independiente de la siguiente . Todas las llamadas al mismo servicio serán idénticas.



Cache : El contenido de los servicios web REST ha se puede cachear de tal forma que una vez realizada la primera petición al servicio el resto puedan apoyarse en la cache si fuera necesario.



Servicios Uniformes : Todos los servicios REST compartirán una forma de invocación y métodos uniformes utilizando los métodos GET, POST, PUT, DELETE



Arquitectura en Capas: Todos los servicios REST están orientados hacia la escalabilidad y un cliente REST no será capaz de distinguir entre si está realizando una petición directamente al servidor, o se lo está devolviendo un sistema de caches intermedio o por ejemplo existe un balanceador que se encarga de redirigirlo a otro servidor.

