6. Arquitecturas Orientadas a Servicios y Servicios Web

6.1 El Problema: Comunicación entre Programas Distantes

Imagina que tienes una clase Java `SaySomething` en tu portátil y quieres usar un método `helloBuddy` de una clase `Hello` que está en un servidor en la otra punta del mundo .

6.1.1 La Evolución Fallida

Se intentaron varias soluciones (RPC, CORBA, DCOM), pero fracasaron en la web abierta por dos razones:

1. **Bloqueo:** Los **Firewalls** bloqueaban sus comunicaciones complejas.
2. **Incompatibilidad:** Requerían que ambas máquinas usaran la misma tecnología (ej. Java con Java).

> **Info**
>
> El **firewall** es el guardia de seguridad de la red de una empresa o un ordenador.
>
> - **Funcionamiento:** Controla las "puertas" (puertos) de entrada y salida. Imagina un edificio con 65.000 puertas. El Firewall las cierra todas por seguridad, excepto las imprescindibles.
> - **El problema histórico:** Antiguamente, las tecnologías distribuidas usaban puertos raros y aleatorios. Los Firewalls las bloqueaban sistemáticamente, impidiendo la comunicación entre empresas.
> - **La solución Web:** Los Firewalls casi siempre dejan abierta la **puerta 80 (HTTP/Web)**, porque todo el mundo necesita navegar por internet. SOA aprovecha esto para "colar" sus datos por esa puerta abierta.

6.1.2 La Solución: Servicios Web

La idea genial fue: *"¿Y si empaquetamos nuestros datos en **XML** (que todos entienden) y los enviamos usando **HTTP** (que atraviesa todos los Firewalls)?"*.

De aquí nace la **Arquitectura Orientada a Servicios (SOA)**.

> ⓘ **Info**
>
> **XML:** Imagina que escribes una carta a un amigo, pero quieres que un ordenador entienda perfectamente qué parte es la fecha, cuál es el saludo y cuál el contenido.
>
> - **Definición:** Es un estándar para estructurar datos en formato de texto. A diferencia de HTML (que le dice al navegador *cómo mostrar* algo, ej: "pon esto en negrita"), **XML le dice al ordenador *qué es* ese dato**.
> - **Funcionamiento:** Usa "etiquetas" (tags) que abren y cierran.
>   - Ejemplo: `<precio>50</precio>`. Aquí el ordenador sabe que "50" es un precio.
> - **Por qué es vital en SOA:** Porque es **independiente del lenguaje**. Java, Python o C++ pueden leer texto plano. Si enviamos datos en formato XML, todos se entienden .
>
> **HTTP** es el protocolo (el idioma) de la World Wide Web.
>
> - **Funcionamiento:** Funciona con un modelo de **Petición-Respuesta**.
>   1. Tu navegador (Cliente) envía una petición (Request) a un servidor: *"Dame la página https://www.google.com/search?q=google.com"*.
>   2. El servidor responde (Response): *"Aquí tienes el código de la página"*.
> - **Características:** Es un protocolo de texto y **sin estado** (stateless), lo que significa que cada petición es independiente de la anterior.
> - **Verbos:** Usa acciones como `GET` (pedir info) o `POST` (enviar info). En Servicios Web, usaremos mucho `POST` para enviar mensajes de datos al servidor.

6.2 Arquitectura Orientada a Servicios (SOA)

SOA no es un software, es una forma de organizar tus sistemas. Se basa en construir aplicaciones conectando piezas independientes llamadas **Servicios**.

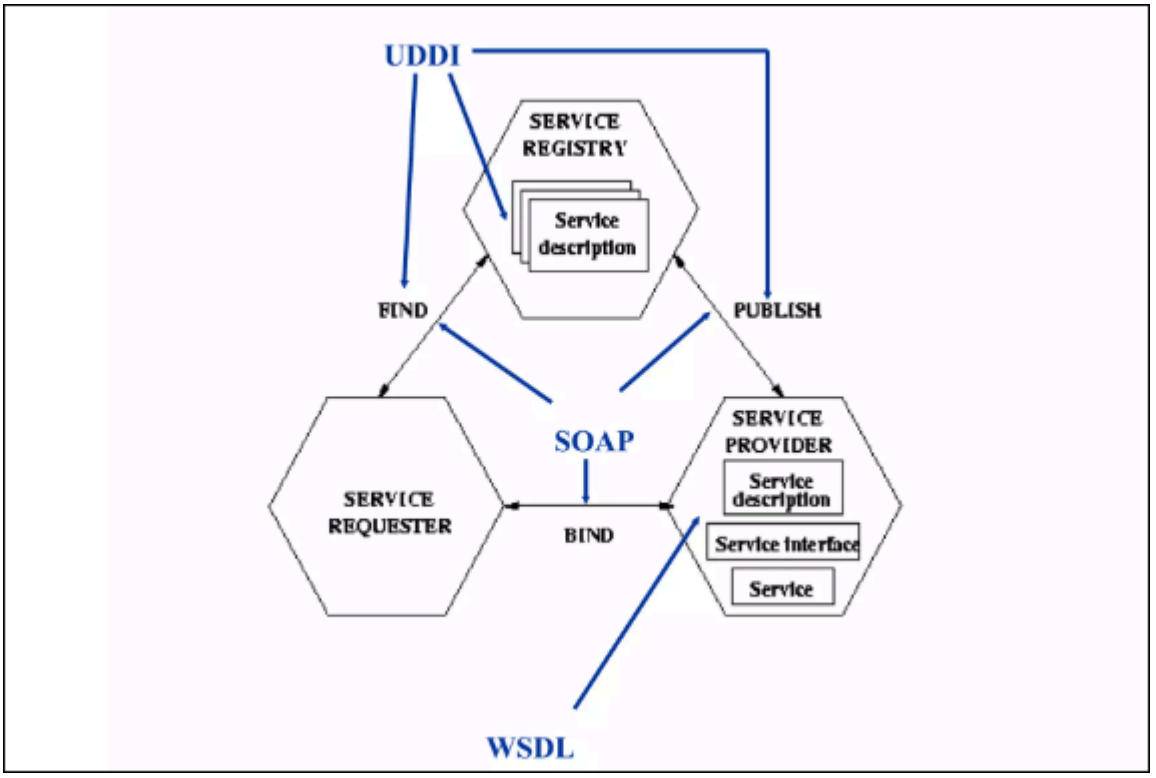
Para que esto funcione, necesitamos tres actores :

1. **Proveedor (Service Provider):** Crea el servicio y lo mantiene.
2. **Registro (Service Registry):** Un directorio (como unas Páginas Amarillas) donde se listan los servicios disponibles.
3. **Consumidor (Service Requester):** El cliente que necesita usar el servicio.

El Flujo de Trabajo:

1. **PUBLISH (Publicar):** El Proveedor sube la descripción de su servicio al Registro.
2. **FIND (Buscar):** El Consumidor busca en el Registro: "¿Quién tiene un servicio de calculadora?".

3. **BIND (Ligar/Conectar):** El Consumidor obtiene la dirección y conecta directamente con el Proveedor para usarlo.



6.3 Implementación de SOA mediante Servicios Web

Aunque SOA es el concepto, los **Servicios Web** son la implementación técnica más común. Utilizan estándares basados en **XML** para garantizar que cualquier lenguaje (Java, Python, C#) pueda hablar con cualquier otro .

El "Stack" tecnológico estándar del W3C incluye :

1. **Transporte:** HTTP (generalmente).
2. **Mensajería (Formato): SOAP** (Simple Object Access Protocol).
3. **Descripción (Contrato): WSDL** (Web Service Description Language).
4. **Descubrimiento: UDDI** (Universal Description, Discovery and Integration) — *Nota: Este último ha caído en desuso, pero es parte de la teoría clásica*.

6.4 El Manual de Instrucciones: WSDL

Si vas a conectar tu código con un servicio remoto, necesitas saber exactamente cómo hablarle. Para eso existe el **WSDL** (Web Service Description Language).

Es un documento **XML** que actúa como un "contrato". Describe dos cosas fundamentales :

Descripción Abstracta (El "Qué")

Define la funcionalidad sin importar el lenguaje de programación.

- **Types (Tipos):** Define los datos usando **XML Schema**. Aquí definimos qué es un "Usuario" o una "Factura".
  - *Traducción:* Si en Java tienes un `int`, en WSDL se define como `xsd:int`.
- **Messages (Mensajes):** Define qué datos entran y cuáles salen.
- **Port Type (Interfaz):** Agrupa las operaciones (funciones) disponibles .

Descripción Concreta (El "Cómo")

- **Binding:** Dice qué protocolo usar. Casi siempre dirá: "Usa **SOAP** sobre **HTTP**".
- **Service:** Dice la dirección web (URL) donde está el servidor escuchando.

> **Proceso Mágico:** Gracias a que el WSDL es XML estandarizado, herramientas automáticas (como JAX-WS) pueden leerlo y generar código Java (clases) automáticamente por ti .

Info

WSDL Es el **Menú y el Formulario de Pedido**. No es la comida, ni es el camarero. Es un documento (un papel) que te dice:

- **Qué ofrecen:** "Tenemos Hamburguesas y Ensaladas".
- **Cómo pedirlo:** "Para la hamburguesa DEBES decirnos: tipo de carne (texto) y si quieres queso (booleano)".
- **Dónde:** "Estamos en la Calle Falsa 123".

En resumen: El WSDL es el **contrato**. Antes de escribir una sola línea de código, el cliente lee el WSDL para saber qué métodos existen y qué parámetros necesitan.

6.5 El Mensaje: SOAP (Simple Object Access Protocol)

Ya tenemos el manual (WSDL) y el canal (HTTP). Ahora necesitamos el sobre para enviar la carta. Eso es **SOAP**.

SOAP es un protocolo basado estrictamente en **XML** para intercambiar información estructurada.

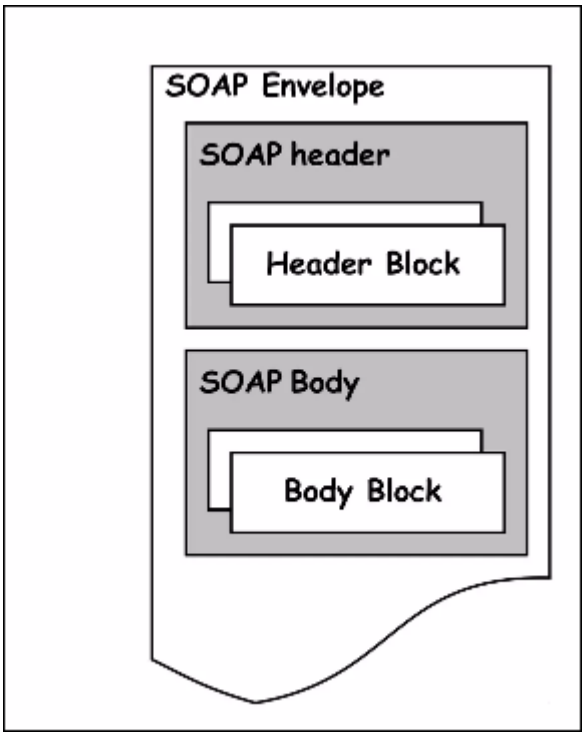
Estructura de un mensaje SOAP

Visualiza una carta real. Un mensaje SOAP tiene :

1. **Envelope (Sobre):** Es la etiqueta XML raíz que envuelve todo. Sin esto, no es SOAP.
2. **Header (Cabecera) - Opcional:** Información para la infraestructura, no para la aplicación.

- Aquí van cosas como firmas digitales de seguridad o IDs de transacción.
- Es procesado por los "intermediarios" (servidores por los que pasa el mensaje antes de llegar al destino).

3. **Body (Cuerpo) - Obligatorio:** Aquí va la información real que quiere el usuario (ej: los números a sumar).



Info

SOAP es el **Sobre certificado** o la **Caja de envío**. Una vez que sabes qué pedir (gracias al WSDL), no puedes gritarlo ni escribirlo en una servilleta. Tienes que meter tu pedido en una caja estándar que el restaurante acepta.

- **Envelope:** La caja.
- **Header:** La etiqueta de envío (para quién es, firma de seguridad).
- **Body:** La hamburguesa (los datos reales).

En resumen: SOAP es el **formato del paquete**. Es puro XML estructurado para asegurar que el mensaje llegue y se entienda.

No hay aliasing, En **SOAP**, la comunicación es por **copia y texto (Pass-by-Value)**.

1. Tus datos en Java se convierten en texto (XML).
2. Ese texto viaja por internet.
3. El servidor lee el texto y crea **nuevos** objetos Java.

Conceptos Clave del Header

Como el mensaje viaja por internet saltando de servidor en servidor, el Header tiene atributos especiales:

- `role`: Define **quién** debe leer esa parte de la cabecera (¿el siguiente servidor o el destino final?).
- `mustUnderstand`: Es un atributo booleano (`true`/`false`). Si está en `true`, obliga al receptor a entender y procesar esa cabecera. Si el receptor no sabe qué hacer con ella, **debe** rechazar el mensaje por seguridad .

6.6 Flujo

Supongamos que quieres sumar `5 + 5`.

1. **Tu Código (Java):** Escribes `servicio.sumar(5, 5)`. Para ti es un método normal.
2. **Stub (El Traductor):** Este es un código falso generado automáticamente. Tú crees que estás llamando al servidor, pero el Stub intercepta la llamada.
   - *Acción:* El Stub toma el `5` y el `5` y empieza a escribir XML.
3. **SOAP Engine:** Mete ese XML en un sobre SOAP.
   - *Resultado:*

```xml
<soap:Envelope>
   <soap:Body>
       <sumar>
           <arg0>5</arg0>
           <arg1>5</arg1>
       </sumar>
   </soap:Body>
</soap:Envelope>
```

4. **Transporte:** Ese tocho de texto viaja por HTTP (Internet).
5. **Servidor (Skeleton/Tie):** Recibe el texto XML. Hace el proceso inverso (**Unmarshalling**).
   - Lee `<arg0>5</arg0>` → Lo convierte a `int a = 5`.
   - Lee `<arg1>5</arg1>` → Lo convierte a `int b = 5`.
6. **Ejecución:** El servidor ejecuta `return 5 + 5`.
7. **Respuesta:** El servidor convierte el `10` resultante en XML (`<return>10</return>`) y lo envía de vuelta.

6.7 Ejemplo

El Servidor (Service Provider)

1. La Interfaz (Calculadora.java) Cambio clave: `javax.jws` → `jakarta.jws`

JAVA

```java
package com.ejemplo;

import jakarta.jws.WebService;
import jakarta.jws.WebMethod;

@WebService
public interface Calculadora {
    @WebMethod
    int sumar(int a, int b);
}
```

2. **La Implementación (CalculadoraImpl.java)**

JAVA

```java
package com.ejemplo;

import jakarta.jws.WebService;

@WebService(endpointInterface = "com.ejemplo.Calculadora")
public class CalculadoraImpl implements Calculadora {

    @Override
    public int sumar(int a, int b) {
        System.out.println(">> Servidor: He recibido una petición para sumar
" + a + " + " + b);
        return a + b;
    }
}
```

3. **El Publicador (Publicador.java)**

JAVA

```java
package com.ejemplo;

import jakarta.xml.ws.Endpoint;

public class Publicador {
    public static void main(String[] args) {
        // Publicamos el servicio en localhost
        String url = "http://localhost:8080/miCalculadora";

        System.out.println("Iniciando servidor...");
        Endpoint.publish(url, new CalculadoraImpl());

        System.out.println("Servicio publicado exitosamente.");
        System.out.println("WSDL disponible en: " + url + "?wsdl");
    }
}
```

El Cliente (Service Consumer)

JAVA

```java
package com.ejemplo;

import jakarta.xml.ws.Service;
import javax.xml.namespace.QName; // QName sigue siendo parte del JDK estándar (javax)
import java.net.URL;

public class Cliente {
    public static void main(String[] args) {
        try {
            // 1. URL donde está el contrato (WSDL)
            URL wsdlURL = new URL("http://localhost:8080/miCalculadora?wsdl");

            // 2. Qualified Name (QName) del servicio
            // Estos nombres (Namespace y ServiceName) están definidos dentro del XML del WSDL
            QName qname = new QName("http://ejemplo.com/", "CalculadoraImplService");

            // 3. Crear la fábrica de servicios (Usando JAKARTA)
            Service service = Service.create(wsdlURL, qname);

            // 4. Obtener el "Stub" (el objeto proxy que implementa nuestra interfaz)
            Calculadora calculadoraProxy = service.getPort(Calculadora.class);

            // 5. Invocar el método remoto
            System.out.println("Cliente: Enviando petición sumar(10, 20)...");

            // --- AQUÍ OCURRE EL MARSHALLING (Java -> XML SOAP) ---
            int resultado = calculadoraProxy.sumar(10, 20);
            // --- AQUÍ OCURRE EL UNMARSHALLING (XML SOAP -> Java) ---

            System.out.println("Cliente: El resultado recibido es " + resultado);

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

```
            }
        }
    }
```