

# Servicios SOAP

## estándares

### WSDL

#### WSDL, Web Services Description Language:

- Descriptor de un web service.
- Especifica los detalles de la interacción entre el cliente y el servidor, en un formato genérico XML, desacoplado de las plataformas y tecnologías.
- En principio, un cliente sólo necesita conocer el WSDL de un web service para saber cómo invocarlo.

La versión de WSDL utilizada por los frameworks es la 1.1 o 1.2. La versión 2.0 es bastante diferente, y soportada sólo por algunos frameworks.

A continuación se presentan los conceptos que forman un WSDL, desde un punto de vista de diseño de la interacción cliente – servidor. Pero antes, se realiza un ejercicio práctico para familiarizarse con los términos.

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

#### Resumen del ejercicio:

- El ejercicio introduce la publicación y consumo de web services, para ejemplificar el estándar WSDL y el protocolo SOAP sobre HTTP.
- Se utilizan en forma guiada y simplificada los wizard que provee Eclipse para el framework Axis, antes de presentarlo como tal.
- Utilizar un proyecto Web que contiene una clase que se quiere publicar como Web Service, y que está desplegado sobre un Tomcat.
- Publicar la clase, utilizando el wizard de Eclipse "Web Service".
- Verificar la publicación, observando el WSDL a través de su URL.
- Utilizar un proyecto Java para consumir el servicio, utilizando el wizard de "Web Service Client".
- Utilizar la herramienta Tcpmon para observar los mensajes SOAP.

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

- Importar el proyecto **cp-basic-server**, del tipo dynamic web project creado con Eclipse. Contiene una clase UtilService, con dos métodos básicos:

```
public class UtilService {  
  
    public String concat(String s1, String s2) {  
        return s1 + s2;  
    }  
  
    public int sum(int n1, int n2) {  
        return n1 + n2;  
    }  
}
```

En el ejercicio, se publica la clase UtilService como web service.

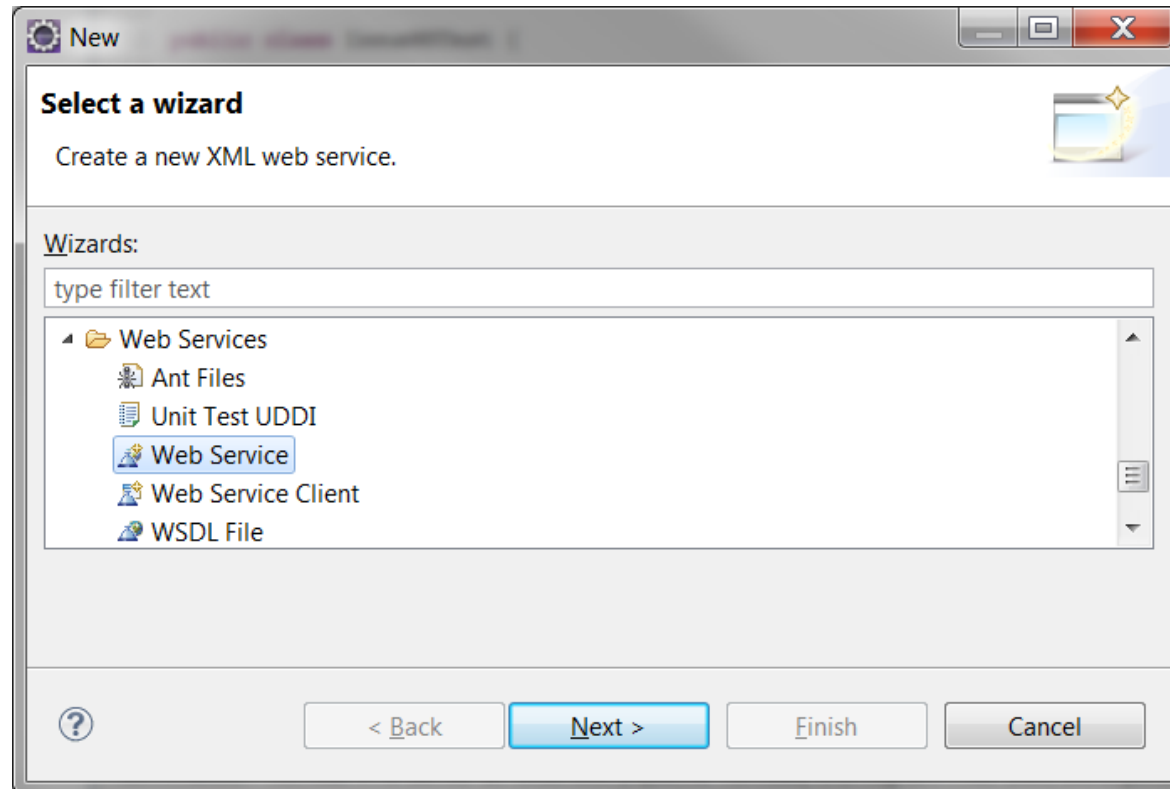
- Agregar a **Tomcat** el proyecto cp-basic-server.
- Iniciar el servidor Tomcat y asegurar que parte correctamente. Este paso es imprescindible antes de publicar el servicio, ya que el Wizard de Eclipse que se utiliza en este ejercicio levanta el servidor, y si no parte bien, **no publica el servicio**.

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

La clase UtilService se publica con el Wizard de Eclipse. Se pide lo siguiente:

- Sobre cp-basic-server, se selecciona la opción New → Web Services → Web Service.

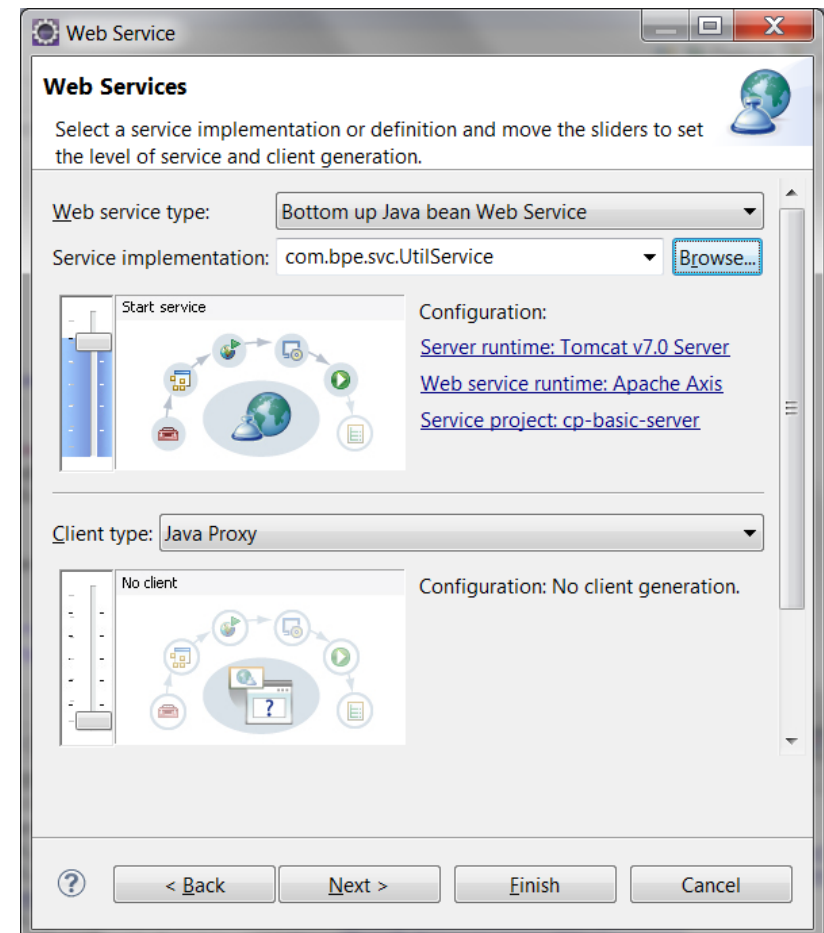


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- En la ventana siguiente, se elige la opción **Bottom Up**, porque se quiere publicar una clase Java como Web Service. Se selecciona la clase UtilService, incluyendo su paquete.

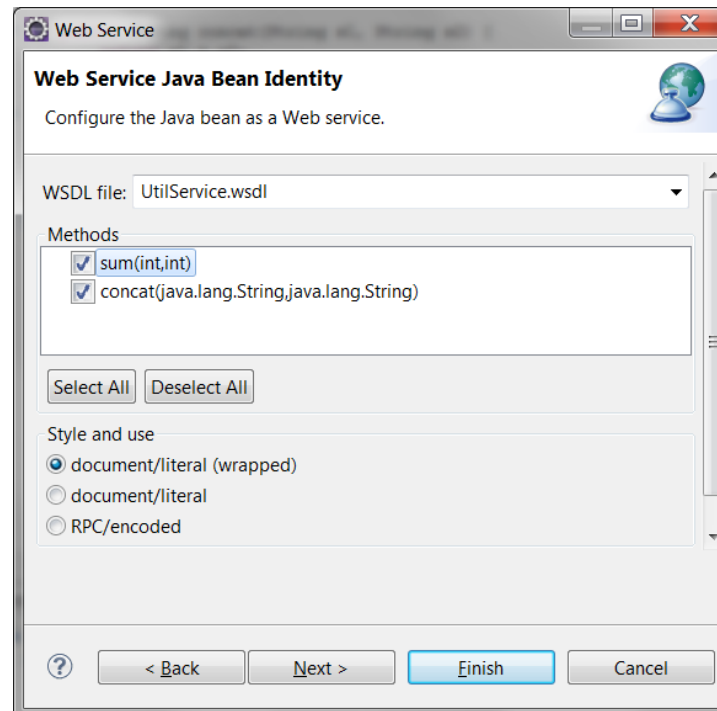


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- La clase UtilService es analizada por el wizard, y se muestran sus métodos. Se selecciona la opción **document/literal (wrapped)**, opción recomendada.

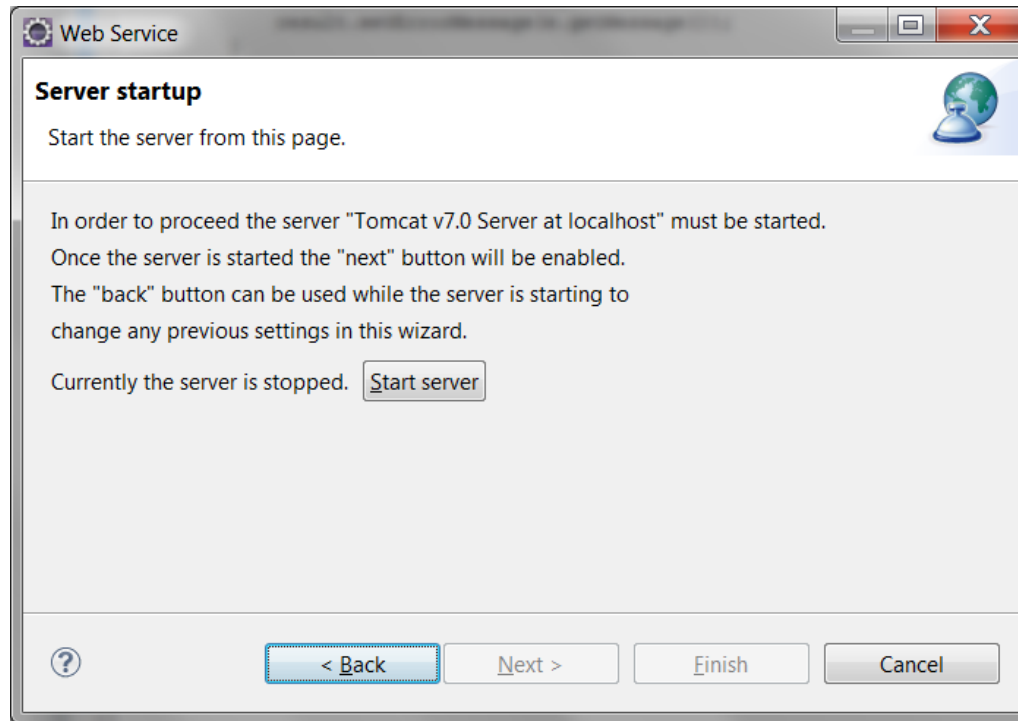


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- Hasta aquí, el servicio está parcialmente configurado. Se debe iniciar el servidor para completar la configuración y finalizar la publicación. Luego de iniciado, pulsar Finish.





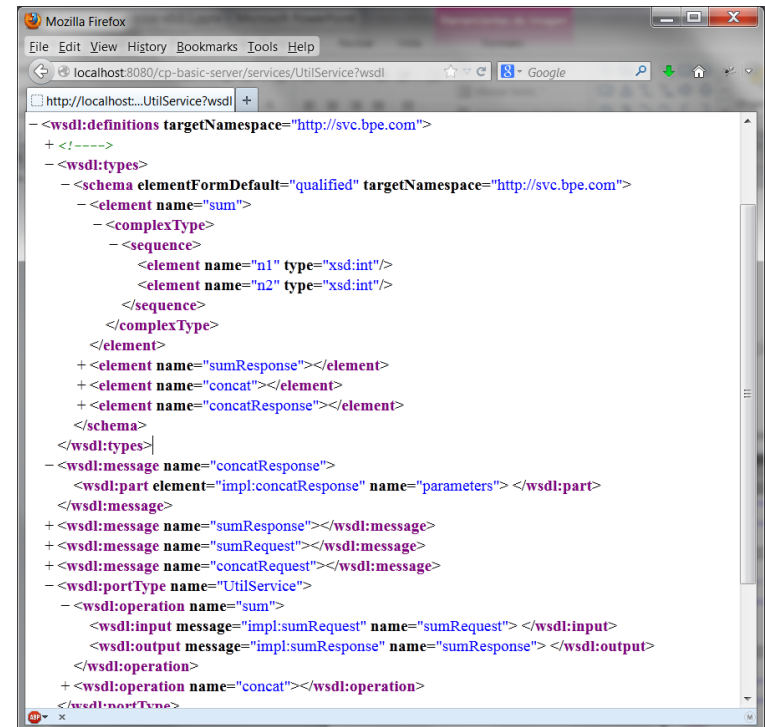
## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Después de la publicación, abrir un browser con la siguiente URL:

`http://localhost:8080/cp-basic-server/services/UtilService?wsdl`

- Esto muestra el WSDL del servicio UtilService publicado.
- El WSDL es el "contrato" del servicio.
- Se observan las partes del WSDL:
  - types: schema con los tipos asociados a los parámetros y respuesta de las operaciones (métodos sum y concat)
  - message: se observan los mensajes request y response de cada operación.
  - portType: se asocia a la clase.
  - binding: se explica más adelante.
  - service: se explica más adelante.



```

<?xml version='1.0' encoding='utf-8'>
<wsdl:definitions targetNamespace="http://svc.bpe.com">
  <!--
  -->
  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://svc.bpe.com">
      <element name="sum">
        <complexType>
          <sequence>
            <element name="n1" type="xsd:int"/>
            <element name="n2" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
      <element name="sumResponse"></element>
      <element name="concat"></element>
      <element name="concatResponse"></element>
    </schema>
  </wsdl:types>
  <wsdl:message name="concatResponse">
    <wsdl:part element="impl:concatResponse" name="parameters"></wsdl:part>
  </wsdl:message>
  <wsdl:message name="sumResponse"></wsdl:message>
  <wsdl:message name="sumRequest"></wsdl:message>
  <wsdl:message name="concatRequest"></wsdl:message>
  <wsdl:portType name="UtilService">
    <wsdl:operation name="sum">
      <wsdl:input message="impl:sumRequest" name="sumRequest"></wsdl:input>
      <wsdl:output message="impl:sumResponse" name="sumResponse"></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="concat"></wsdl:operation>
  </wsdl:portType>
  </wsdl:definitions>

```

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

A continuación, se revisa cómo consumir el Web Service publicado.

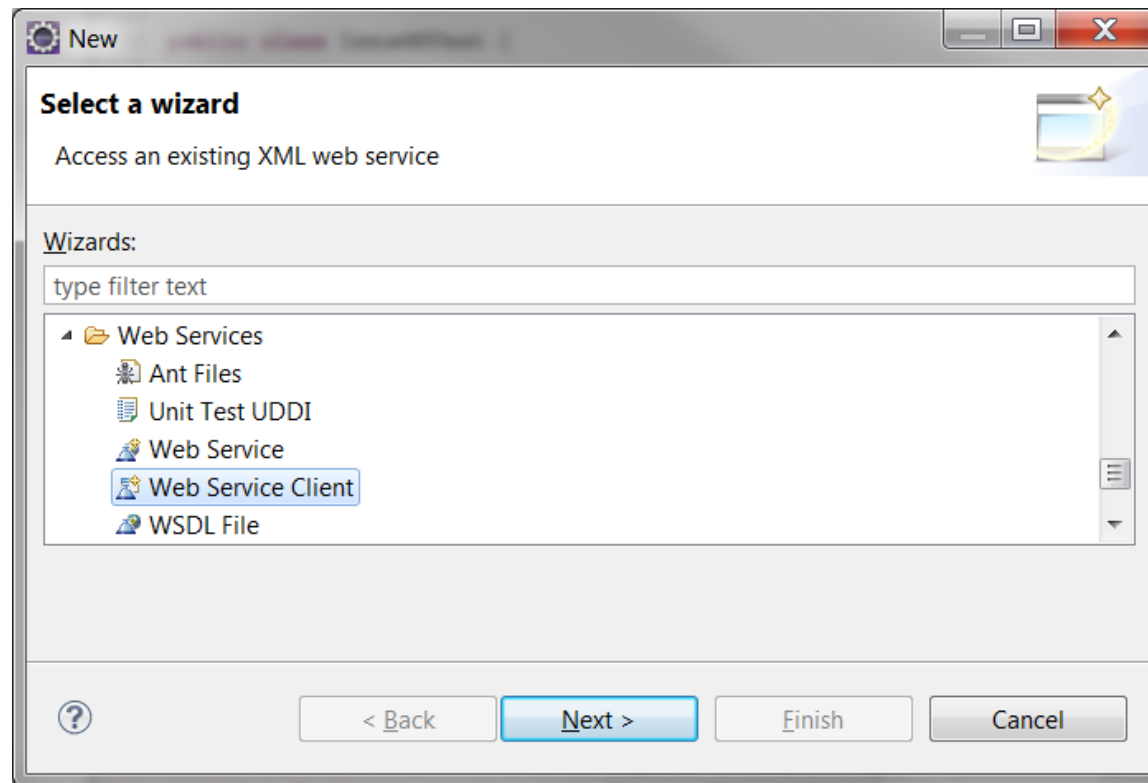
- Importar el proyecto llamado **cp-basic-client**, que es de tipo "Java Project" creado con Eclipse. El proyecto está **separado** de cp-basic-server para ilustrar que cliente y servidor están desacoplados.
- Asegurar que el servidor Tomcat que contiene el proyecto cp-basic-server continúa levantado. Esto es necesario ya que desde el cliente se necesita acceder al WSDL, el cual se lee directamente desde la URL del servicio.

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Para crear las clases del cliente, se utiliza el wizard de Eclipse:

- En cp-basic-client, seleccionar la opción New → Web Services → Web Service Client.

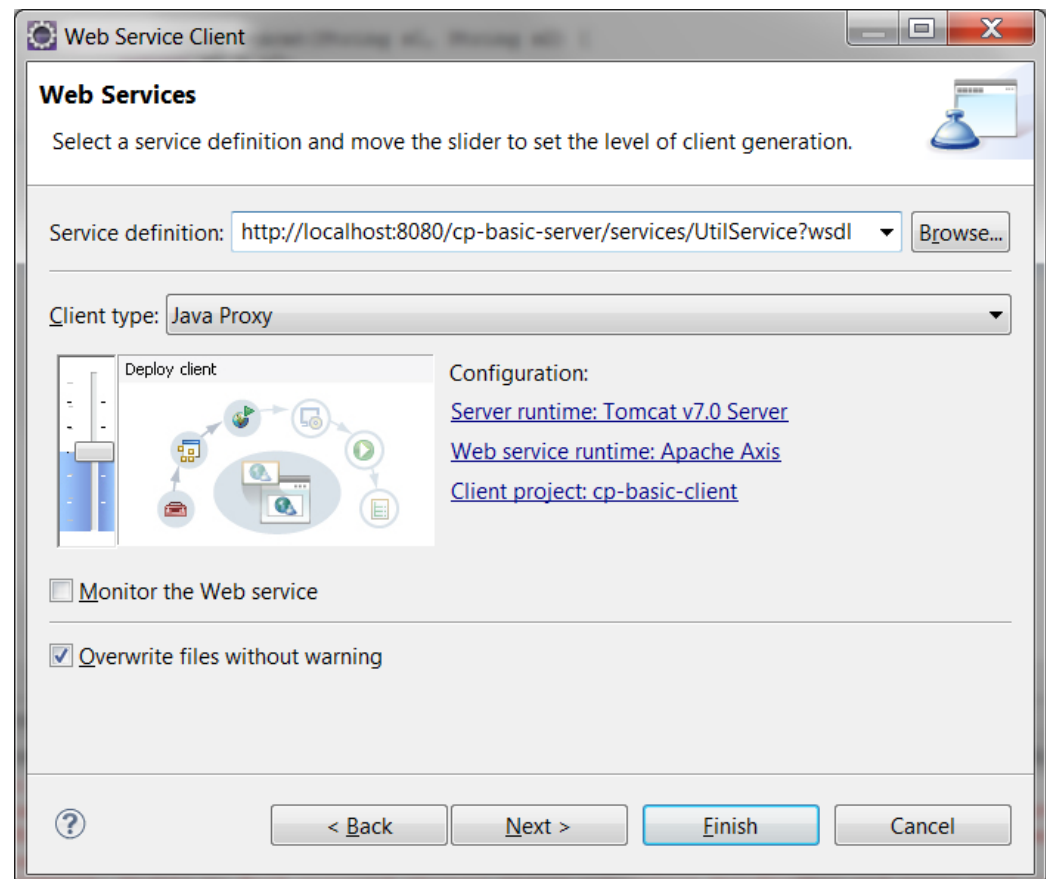


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- En la ventana siguiente, se selecciona la definición del servicio, utilizando la URL del WSDL.

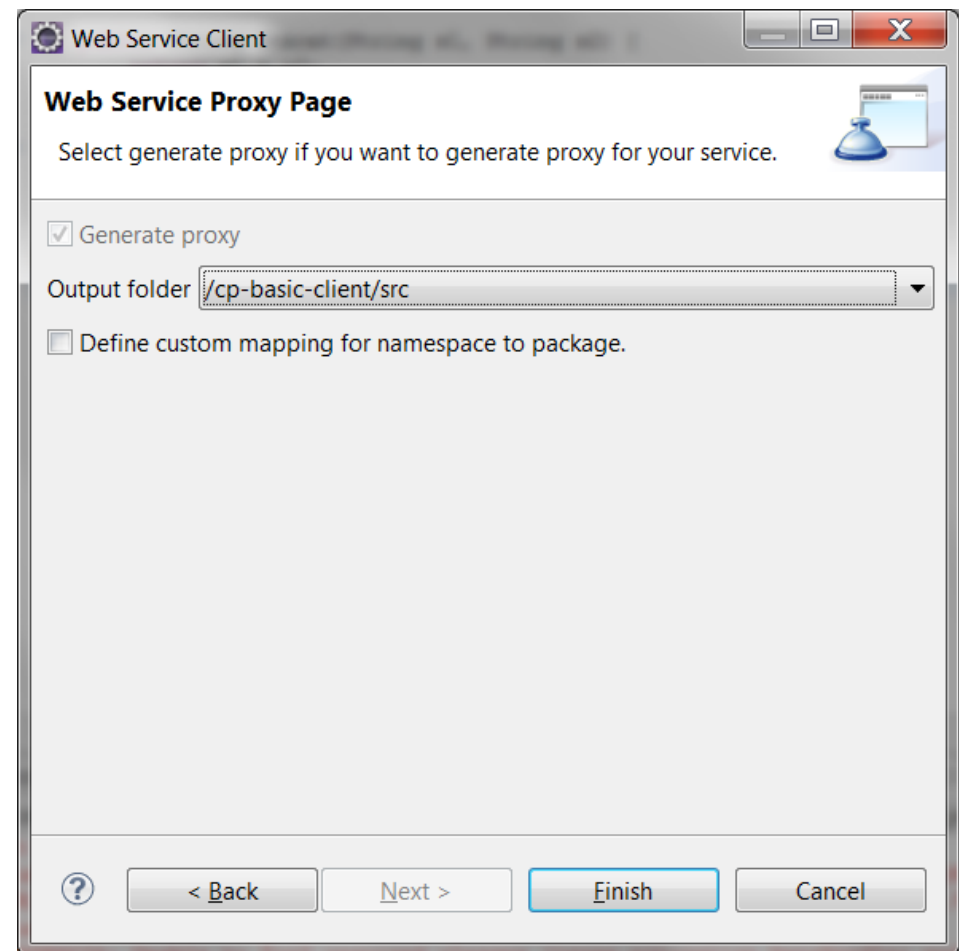


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- En la ventana siguiente, se elige la carpeta src, que es la única opción, y es donde queda el código fuente generado desde el WDSL.

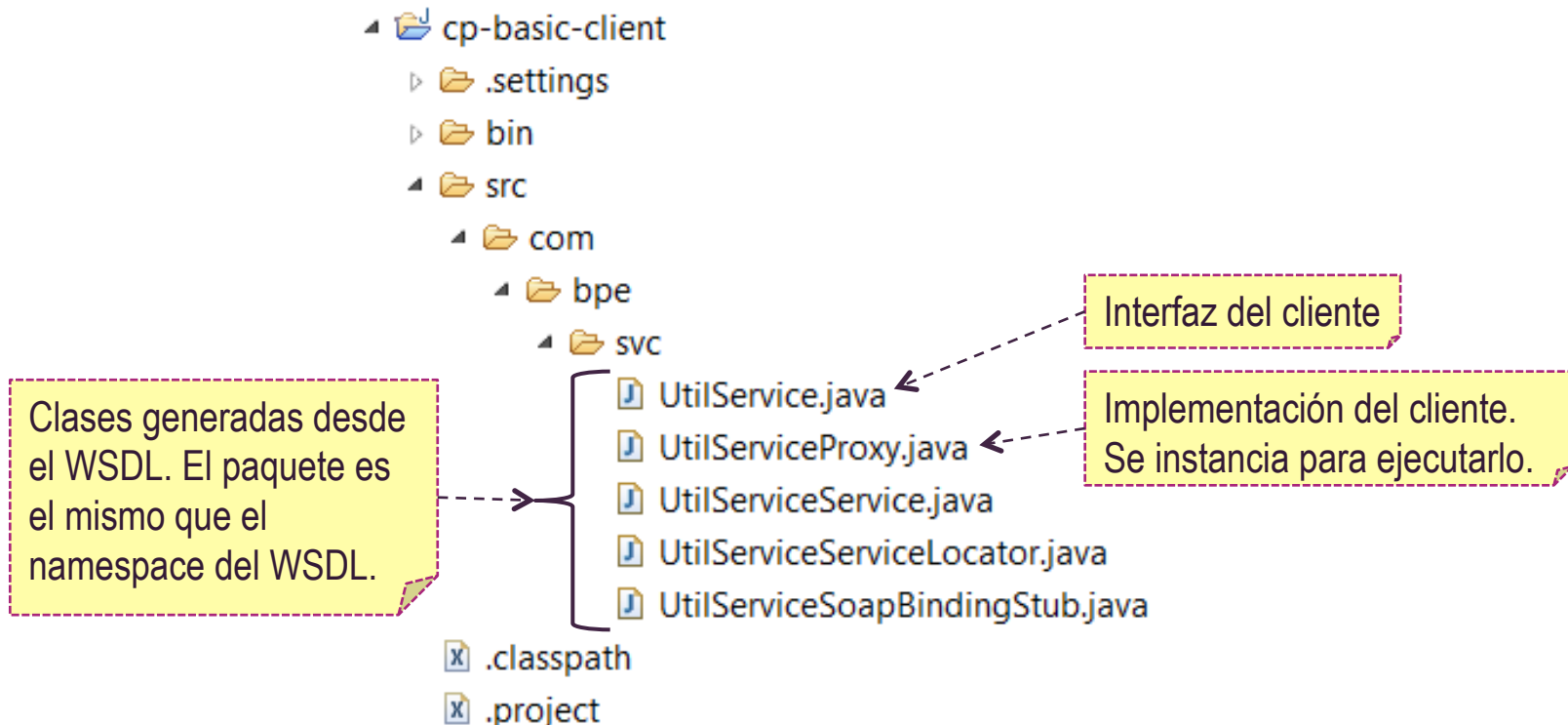


## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Continuando:

- El cliente que se genera contiene un conjunto de clases:



## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Ejecución del cliente:

- La ejecución básica del cliente se realiza instanciando la clase proxy generada. Se pide construir en el mismo paquete de las clases generadas un test `UtilServiceTest`, que verifique una suma y una concatenación:

```
@Test
public void testConcat() {
    try {
        String s1 = "123";
        String s2 = "abc";
        UtilService serv =
            new UtilServiceProxy();
        String c = serv.concat(s1,s2);
        Assert.assertEquals(s1 + s2, c);
    } catch (Exception e) {
        Assert.fail();
    }
}
```

```
@Test
public void testSum() {
    try {
        int n1 = 3;
        int n2 = 4;
        UtilService serv =
            new UtilServiceProxy();
        int s = serv.sum(n1, n2);
        Assert.assertEquals(n1 + n2, s);
    } catch (Exception e) {
        Assert.fail();
    }
}
```

## estándares y protocolos

### Ejercicio práctico: Implementación y consumo de servicio básico

Al ejecutar los test, los mensajes que se envían entre cliente y servidor son los siguientes:

#### concat - request

```
<Envelope>
  <Body>
    <concat xmlns="http://svc.bpe.com">
      <s1>123</s1>
      <s2>abc</s2>
    </concat>
  </Body>
</Envelope>
```

#### concat - response

```
<Envelope>
  <Body>
    <concatResponse xmlns="http://svc.bpe.com">
      <concatReturn>123abc</concatReturn>
    </concatResponse>
  </Body>
</Envelope>
```

#### sum - request

```
<Envelope>
  <Body>
    <sum xmlns="http://svc.bpe.com">
      <n1>3</n1>
      <n2>4</n2>
    </sum>
  </Body>
</Envelope>
```

#### sum - response

```
<Envelope>
  <Body>
    <sumResponse xmlns="http://svc.bpe.com">
      <sumReturn>7</sumReturn>
    </sumResponse>
  </Body>
</Envelope>
```

Nota: Los mensajes reales tienen más contenido. Más adelante se describe cómo obtenerlos.



## estándares

### Servidor, Web Service y operación

Un Web Service que se publica queda disponible en un servidor, por ejemplo "http://www.example.com", en una ruta "/SimpleService", y de este modo su URL, llamada **"end point"**, es "http://www.example.com/SimpleService".

- Se utiliza un único nombre global para el servicio. De este modo, dos servicios distintos podrían tener su propia operación "concat".
- Se puede definir un **"namespace"** asociado a la operación.
- La combinación del namespace y el nombre de la operación forman el llamado **QName** (Qualified name).
- **QName** se aplica también a otros elementos.

Servidor en "http://www.everis.com"

Web Service en la ruta "/SimpleService"

Operación

Nombre local: concat  
Namespace: http://example.com/ss

Operación

Nombre local: sum  
Namespace: http://example.com/ss

## estándares

### Detalles de la operación

Se tiene una operación "concat" que recibe dos parámetros, uno llamado "s1" del tipo string, y otro llamado "s2" también string, y un retorno con la concatenación, también de tipo string.

¿Qué es el tipo "string"?

No es el tipo Java, sino uno estándar definido en forma **neutral**.

Otros tipos de dato neutrales. Su namespace define el QName asociado. El tipo integer podría mapear con un tipo java int, long, short o byte.

#### Operación

```
Local name: concat
Namespace: http://example.com/ss
Parameters:
  s1: string
  s2: string
Return:
  string
```

Tipo dato	Nombre local	namespace
string	string	http://www.w3.org/2001/XMLSchema
integer	int	http://www.w3.org/2001/XMLSchema
...	...	...

## estándares

### Definición de la operación

Utilizando los QName de los parámetros, la interfaz de la operación "concat" puede definirse de la siguiente manera:

#### Operación

```
Nombre local: concat
Namespace: http://example.com/ss
Parámetros:
  s1: string en http://www.w3.org/2001/XMLSchema
  s2: string en http://www.w3.org/2001/XMLSchema
Retorno:
  string en http://www.w3.org/2001/XMLSchema
```

## estándares

### Input message y output message

En los Web Services, la invocación al método recibe un "input message", donde cada parámetro es llamado una "part". El retorno envía un "output message", el cual contiene también una "part". Así, la definición de la operación queda:

#### Operación

```
Local name: concat
Namespace: http://example.com/ss
Input message:
  Part 1:
    Name: s1
    Type: string en http://www.w3.org/2001/XMLSchema
  Part 2:
    Name: s2
    Type: string en http://www.w3.org/2001/XMLSchema
Output message:
  Part 1:
    Name: return
    Type: string en http://www.w3.org/2001/XMLSchema
```

## estándares

### Invocación estilo RPC

Basado en la definición, cuando se envía un mensaje, se puede utilizar el siguiente formato:

El QName del elemento XML es igual al nombre de la operación invocada

foo es el **prefijo del namespace**, representando a "http://example.com/ss" en el mensaje.

```
<foo:concat xmlns:foo="http://example.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concat>
```

Hay un elemento hijo en cada parte. Cada elemento tiene el mismo nombre que la parte.

El mensaje de retorno, con el resultado, es el siguiente:

```
<foo:output xmlns:foo="http://example.com/ss">
  abc123
</foo:output>
```

Este estilo de invocación es llamado **RPC (Remote Procedure Call)**. Tanto la operación como los parámetros utilizan los QName para crear los mensajes.

## estándares

### Invocación estilo Document

#### Operación

Local name: concat  
 Namespace: http://example.com/ss  
 Input message:  
   Part 1:  
     Name: concatRequest  
     Element: -----  
 Output message:  
   Part 1:  
     Name: concatResponse  
     Type: string in http://www.w3.org/2001/XMLSchema

```
<xsd:schema targetNamespace="http://example.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Se define un XSD para concatRequest. Los elementos "s1" y "s2" quedan definidos dentro del XSD. El XSD es parte de la definición del servicio.

```
<foo:concatRequest
  xmlns:foo="http://example.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concatRequest>
```

## estándares

### Invocación estilo Document

El estilo **Document** se definen los elementos en un XSD (XML Schema Definition). Cada elemento del objeto "request" queda definido en el XSD, y en el mensaje queda especificado por su QName.

El estilo Document define una única parte para el input message, definida en un XSD. También define una única parte para el output message en el XSD.

#### XSD

```
<xsd:schema targetNamespace="http://example.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="concatRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="s1" type="xsd:string"/>
        <xsd:element name="s2" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

#### Operación

```
Local name: concat
Namespace: http://example.com/ss
Input message:
  Part 1:
    Name: concatRequest
    Element:
Output message:
  Part 1:
    Name: concatResponse
    Type: string in http://www.w3.org/2001/XMLSchema
```

## estándares

### Invocación estilo Document

El mensaje de salida de una invocación con estilo Document es:

```
<foo:concatResponse
  xmlns:foo="http://example.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
  xsi:type="xsd:string">
  abc123
</foo:concatResponse>
```

Atributo utilizado para asignar el tipo XML del elemento, definido en el esquema <http://www.w3.org/2001/XMLSchema-Instance>



## estándares

### Estilo RPC vs. Document

En el estilo RPC, el mensaje no se puede validar, pues no tiene una definición asociada tipo XSD o DTD. Cada elemento es definido en forma individual. Esta es la razón por la cual la **WS-I** (Web Services Interoperability Organization) recomienda siempre utilizar el estilo Document.

#### Estilo RPC

```
<foo:concat
  xmlns:foo="http://example.com/ss"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
  <s1 xsi:type="xsd:string">abc</s1>
  <s2 xsi:type="xsd:string">123</s2>
</foo:concat>
```

#### Estilo Document

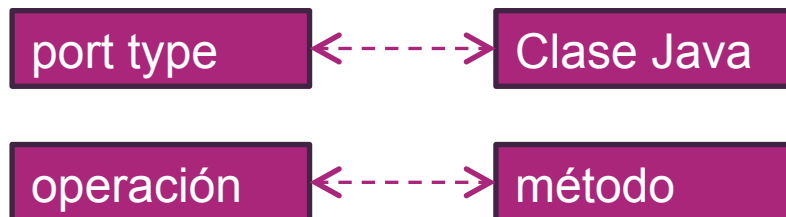
```
<foo:concatRequest xmlns:foo="http://example.com/ss">
  <s1>abc</s1>
  <s2>123</s2>
</foo:concatRequest>
```

## estándares

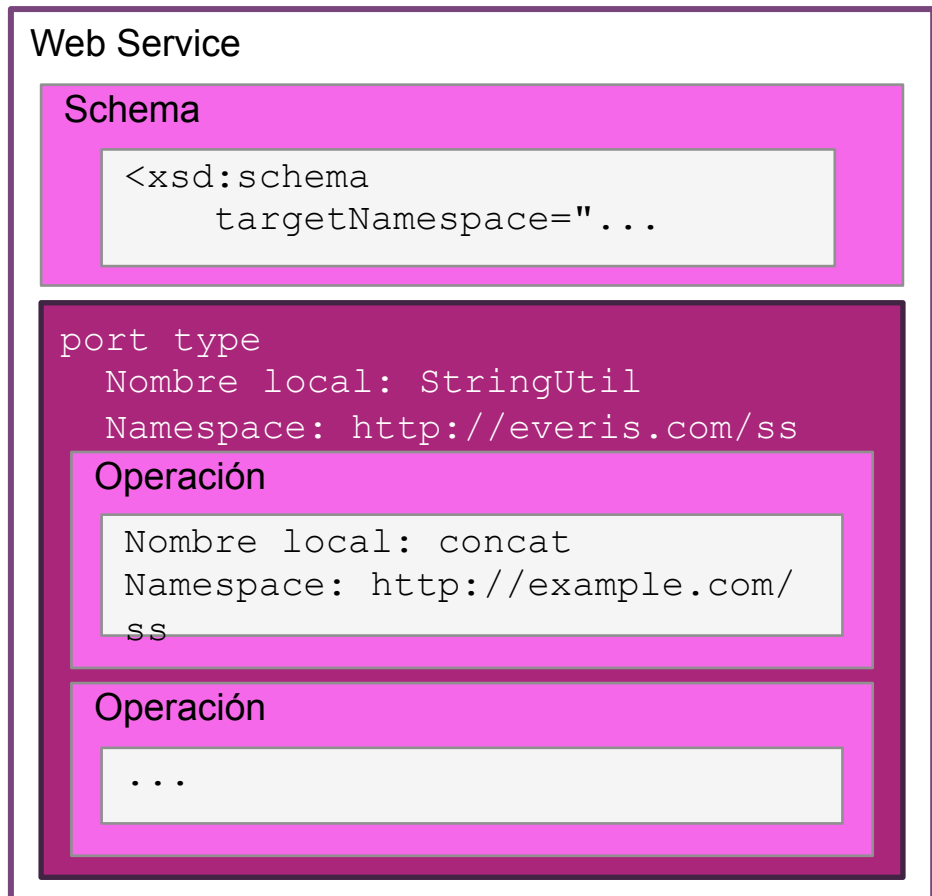
### Operaciones y port type

¿Cómo determinar la operación a partir del mensaje de entrada?

En la definición, existe un "port type", que es como una clase Java en la que cada operación es un método.



El nombre del port type también es un QName, es decir, tiene un namespace asociado.



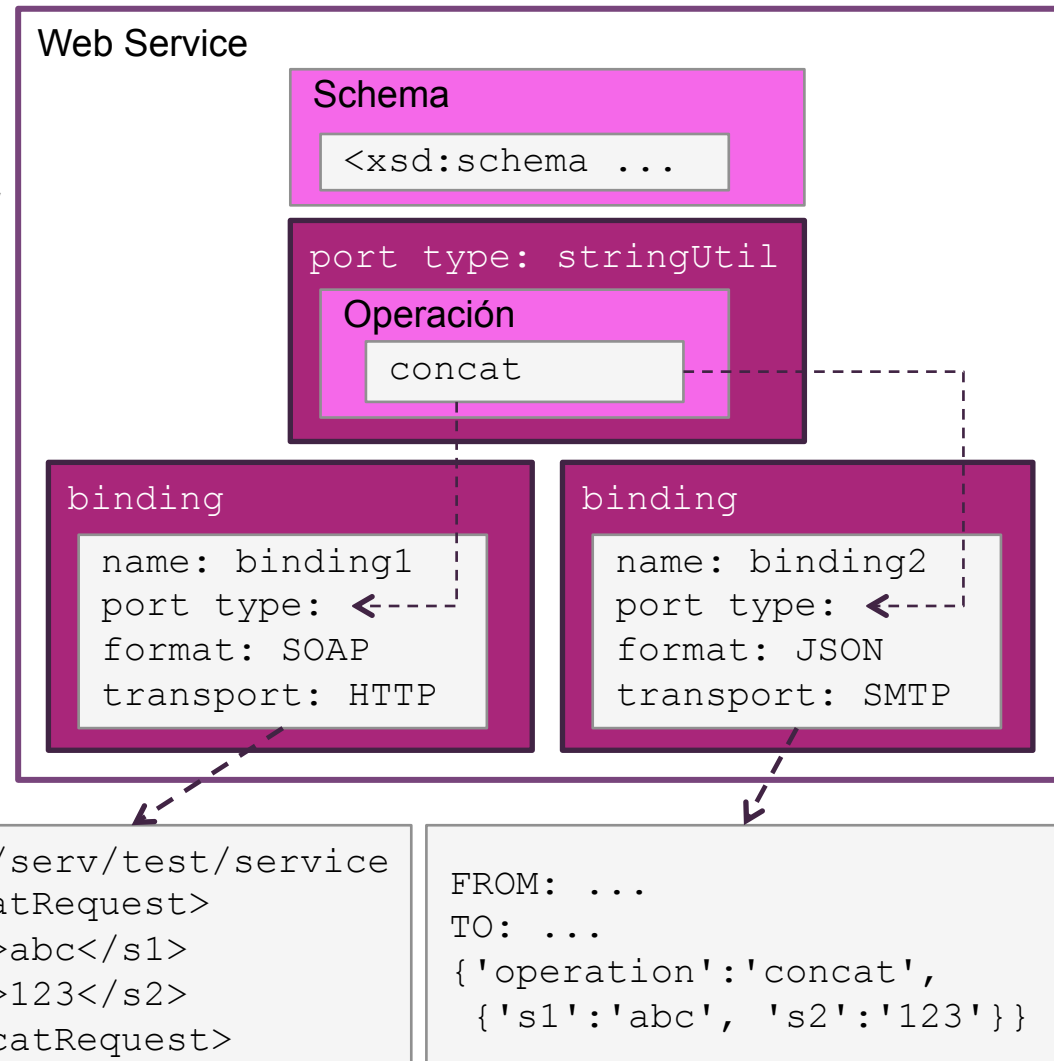
## estándares

### Binding

Un port type puede utilizar distintos formatos de mensaje. El formato XML utilizado es llamado **SOAP** (Simple Object Access Protocol), aunque se podría utilizar, por ejemplo, un formato JSON.

A su vez, los mensajes pueden ser transportados de distintas formas. Lo normal es con **HTTP**, aunque se podría utilizar SMTP.

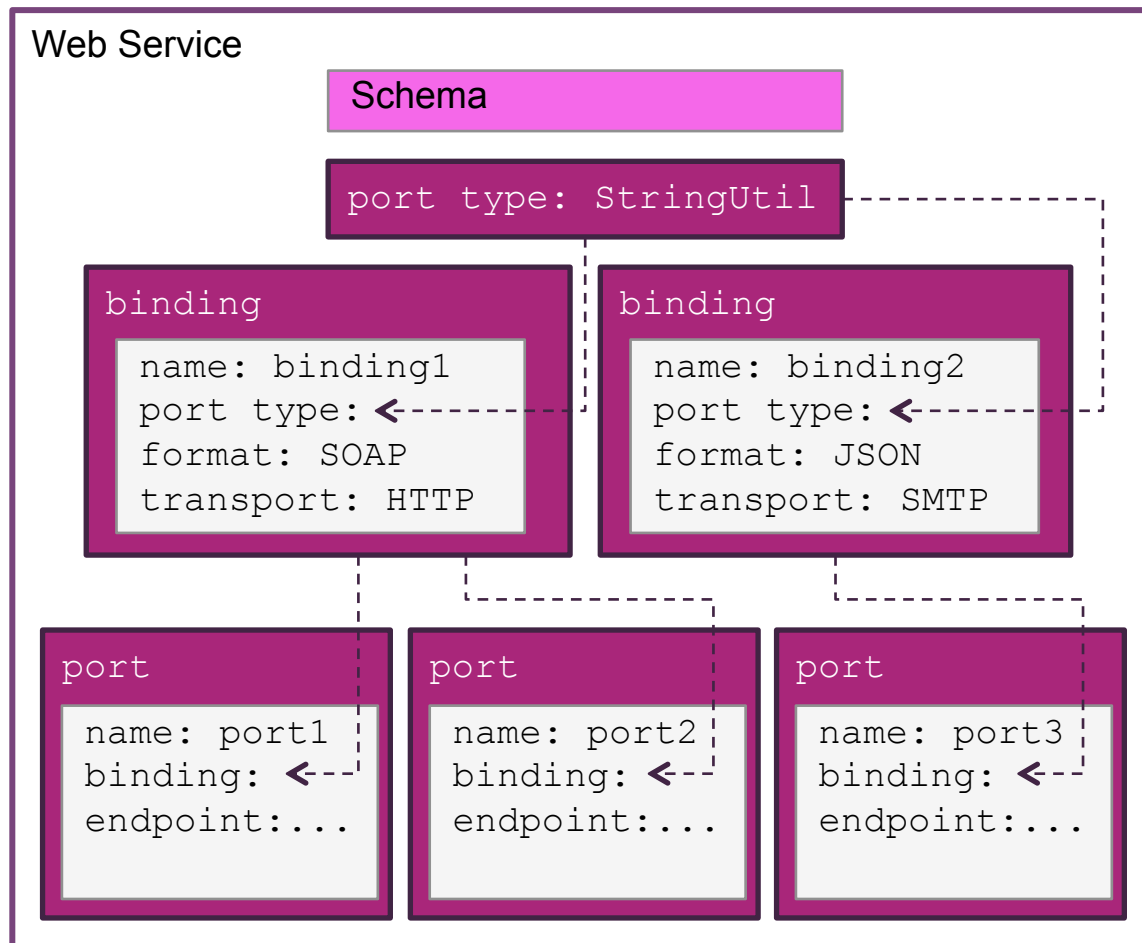
El **mapeo** entre port type, formato y transporte es el **binding**.



## estándares

### Port

- Un servicio puede desplegarse de varias maneras, en función del binding seleccionado.
- El **mapeo** entre el **binding** y el **end point** asociado al servicio es el **port**.
- Con esto, se pueden acotar los protocolos de formato y transporte en función de la tecnología utilizada.



## estándares

### Target namespace

Se han definido algunos namespace para los elementos del servicio: operación, port type, mensajes. Existe un namespace específico en el servicio para colocar todos estos elementos, llamado el "**target namespace**".

La nomenclatura de un namespace debe ser única a nivel global. Se debe utilizar una **URI** (Uniform Resource Identifier):

- **URL** (Uniform Resource Locator), el más utilizado. Ej:  
`http://example.com/ss`
- **URN** (Uniform Resource Name), como alternativa. Ej:  
`urn:example.com:ss`

#### Web Service

Target namespace: `http://example.com/ss`

Schema

`port type: stringUtil`

`binding: binding1`

`binding: binding2`

`port: port1`

`port: port2`

`port: port3`

## estándares

### WSDL

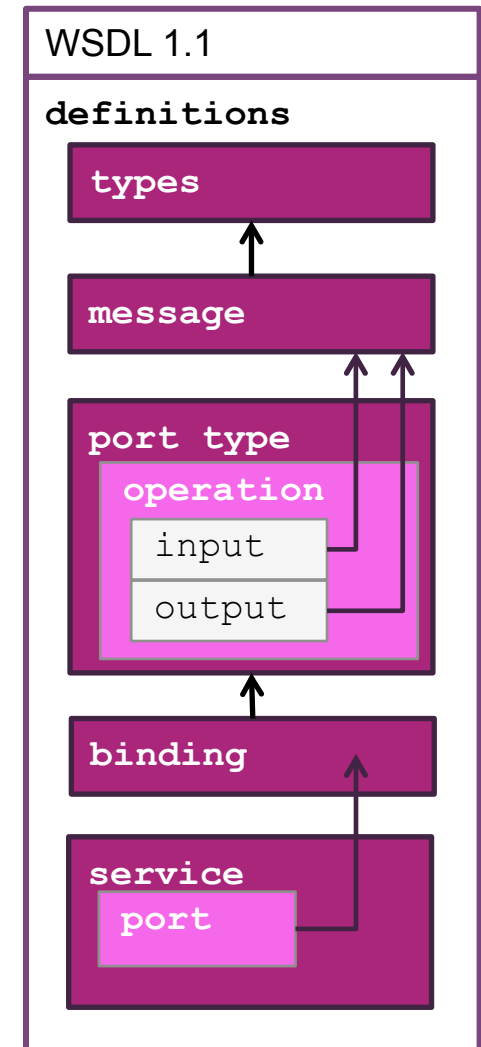
Una vez presentados todos los conceptos básicos, se define WSDL.

El **WSDL** (**W**eb **S**ervices **D**escription **L**anguage) agrupa en un formato XML los elementos que forman la definición de un Web Service.

Cuando un web service está publicado, el WSDL del servicio se puede obtener consultando una URL. En Java, normalmente es la del servicio seguida de "?wsdl" (ver nota):

**`http://<server>:<port>/servicio?wsdl`**

La versión de WSDL utilizada por los frameworks es la 1.1. La versión 2.0 es diferente, y soportada sólo por algunos frameworks.



## estándares

### WSDL

- Un WSDL puede ser interpretado visualmente con algo de experiencia.
- En la práctica no es necesario. Las herramientas de generación de código y las que ejecutan los web services lo utilizan.
- La estructura de un WSDL tiene los elementos presentados anteriormente, en un formato XML. En la versión 1.1 son:
  - types
  - message
  - port type
  - binding
  - port
  - service

Estos elementos van desde los más específicos a los más generales. Para facilitar la lectura, a veces se recomienda hacerlo en orden invertido, es decir, desde "service" hacia "types" (desde **abajo hacia arriba**).

## estándares

### WSDL

1. **types**: define estructuras de datos utilizadas en mensajes, a través de un bloque con formato XSD.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="..."
  xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/ ...>
  <wsdl:types>
    <schema ... />
    <element name="...">
      ...
    </schema>
  </wsdl:types>
```

Como cualquier XSD, puede importar otros XSD, y combinar varios namespaces. Este XSD permite conocer los detalles de los tipos de datos utilizados tanto en los parámetros como en el retorno.



## estándares

### WSDL

- 2. message:** por cada operación, agrupa las partes asociadas a los parámetros de entrada a las operaciones (request), y la del valor de retorno (response).

```
<wsdl:message name="metodoRequest">
  <wsdl:part element="..." name="..." />
</wsdl:message>
<wsdl:message name="metodoResponse">
  <wsdl:part element="..." name="..." />
</wsdl:message>
```

## estándares

### WSDL

3. **port type**: se asocia a la clase, e internamente definen las operaciones, lo que en términos de web services equivale a los métodos de la clase.

```
<wsdl:portType name="Servicio">
  <wsdl:operation name="metodo">
    <wsdl:input message="impl:metodoRequest"
      name="metodoRequest" />
    <wsdl:output message="impl:metodoResponse"
      name="metodoResponse" />
  </wsdl:operation>
  ...
</wsdl:portType>
```

Método asociado a la operación

Referencias a los mensajes

Por cada operación, define el mensaje de entrada y el de salida, los que se mapean a los elementos "message" presentados.

## estándares

### WSDL

- 4. binding:** Se refiere a la vinculación entre los mensajes, el formato, y el protocolo de transporte con el que se envían.

```
<wsdl:binding name="ServicioSoapBinding"
              type="impl:Servicio">
  <wsdlsoap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="metodo">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="metodoRequest">
      <wsdlsoap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="metodoResponse">
      <wsdlsoap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  ...
</wsdl:binding>
```



## estándares

### WSDL

- 5. **port**: publicación de un binding, con su operación, formato y transporte asociados, a través de una dirección (end point) definida por una URL.
- 6. **service**: definición del servicio, que agrupa las definiciones de los port.

Referencia al binding

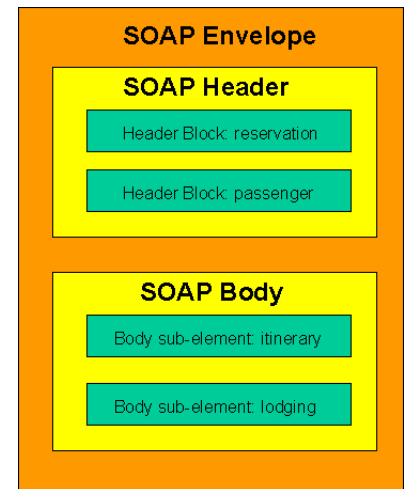
```
<wsdl:service name="ServicioService">
  <wsdl:port binding="impl:ServicioSoapBinding" name="Servicio">
    <wsdlsoap:address
      location="http://serv:port/App/services/Servicio" />
  </wsdl:port>
</wsdl:service>
```

## protocolos

### SOAP

**SOAP** (Simple Object Access Protocol):

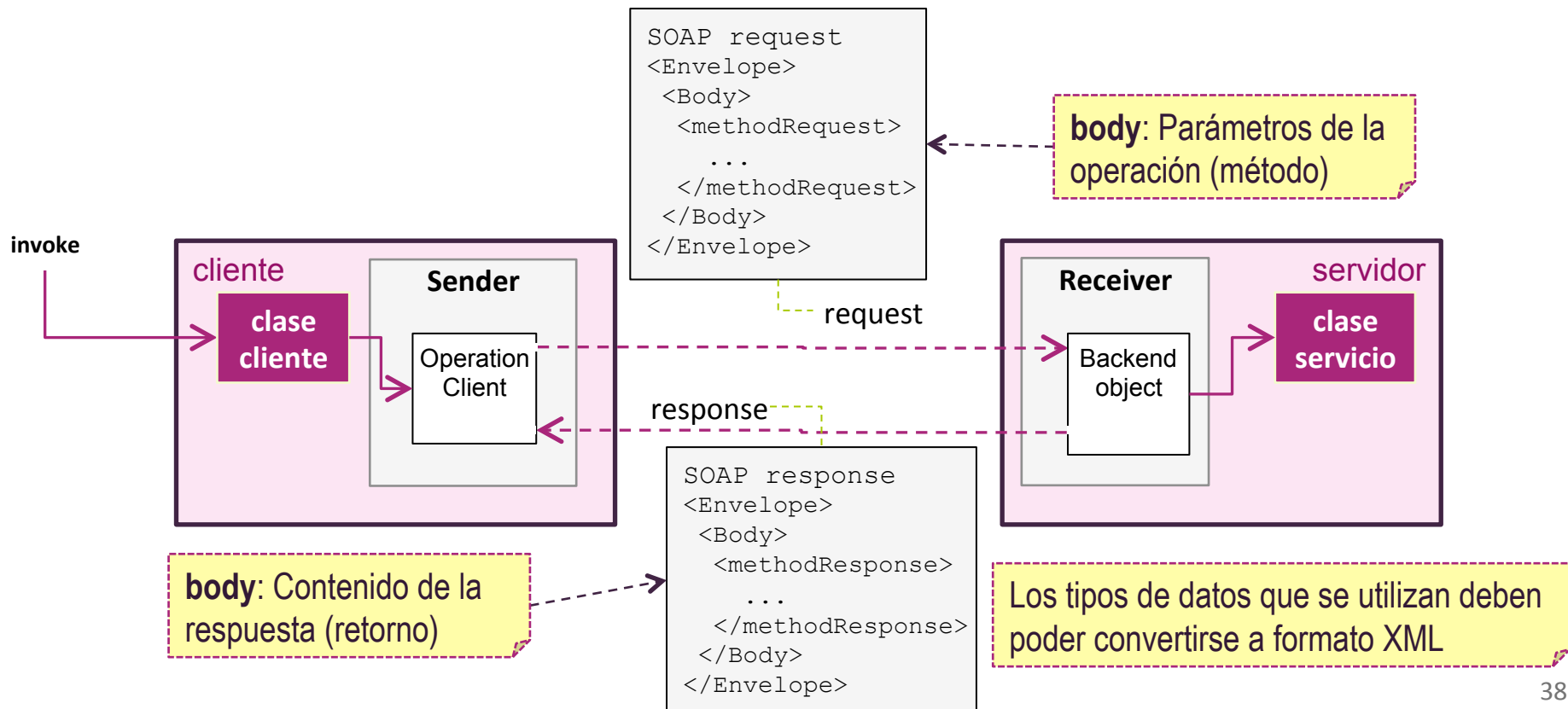
- Protocolo estándar que define la forma en que objetos de diferentes procesos pueden comunicarse por medio de intercambio de datos en formato XML.
- Es un protocolo muy utilizado en los Web Services.
- Consta de:
  - Tag principal llamado **Envelope**, que contiene:
    - **Header** (opcional): se utiliza para enviar información complementaria, como por ejemplo la de seguridad con WS-Security.
    - **Body** (obligatorio): contiene la información del mensaje propiamente tal.



## protocolos

### SOAP

Cuando un Web Service es invocado vía HTTP desde un cliente, se envían dos mensajes SOAP, el request y el response.



## protocolos

### SOAP

- Ejemplo: servicio que concatena dos String

`concat("abc", "123") -> "abc123"`

Header podría omitirse

#### request

```
<Envelope xmlns="...">
  <Header />
  <Body>
    <foo:concatRequest
      xmlns:foo="...">
      <s1>abc</s1>
      <s2>123</s2>
    </foo:concatRequest>
  </Body>
</Envelope>
```

#### response

```
<Envelope xmlns="...">
  <Header />
  <Body>
    <foo:concatResponse
      xmlns:foo="...">
      abc123
    </foo:concatResponse>
  </Body>
</Envelope>
```

## protocolos

### SOAP

#### Características:

- Está respaldado por el W3C para el intercambio de mensajes XML entre aplicaciones y sobre él se sustentan los servicios web.
- Es un protocolo de alto nivel, que define la estructura del mensaje y ciertas reglas básicas para su procesamiento, y es totalmente independiente del protocolo de transporte.
- Esto permite que los mensajes SOAP sean intercambiados mediante HTTP, SMTP, JMS, etc. Normalmente, HTTP es el más utilizado.





## protocolos

### HTTP

**HTTP** (HyperText Transfer Protocol), protocolo de transporte.

- Es el más utilizado en el envío de mensajes con Web Services.
- En términos generales, es un protocolo para intercambiar o transferir "hypertext", que son estructuras lógicas de información en formato de texto.
- HTTP define métodos, que indican la acción que se quiere ejecutar sobre un recurso. Los más utilizados son GET y POST.
- Ejemplo de mensajes SOAP transferidos con HTTP:

#### request

```
GET /service/ss HTTP/1.1
Host: www.example.com

<Envelope xmlns="...">
<Body>...</Body></Envelope>
```

#### response

```
HTTP/1.1 200 OK
Server: ...
Content-Type:...

<Envelope xmlns="...">
<Body>...</Body></Envelope>
```

## consideraciones generales

### compatibilidad SOAP

Los parámetros y el retorno de cada método (operación) de un web service deben ser compatibles, es decir, representables a través de una estructura XML, para ser enviado en un mensaje SOAP. Por ejemplo:

- **Primitivo** (boolean, byte, char, short, int, long, float, double) o sus respectivos wrapper (Boolean, Byte, Character, ...)
- **String**
- **Date y Calendar**
- **BigInteger y BigDecimal**
- Arrays de las clases anteriores
- Clases que tengan como atributos a las clases anteriores, tipo java beans.
- Combinaciones de las anteriores, como un array de beans, o un bean que contiene un array de otro bean.

## consideraciones generales

### compatibilidad SOAP

No se deben utilizar, por ejemplo:

- Elementos tipo Collection (Collection, List, Set o sus implementaciones). Se deben reemplazar por arrays.
- Elementos tipo Map (y sus implementaciones).
- Clases que utilizan tipos genéricos.
- Componentes no Serializables o activas, como un ResultSet (JDBC).

Estas condiciones existen además porque el destino del mensaje puede ser una plataforma con una tecnología distinta a Java, así que aunque existiera una forma de representarlos, no sería recuperable en el destino.

En el caso específico de JAX-WS, como se ve más adelante, permite el uso de objetos tipo List, porque los convierte internamente en arrays. Aún así, por compatibilidad no es recomendable utilizarlos.

## consideraciones generales

### construcción top down vs. bottom up

La construcción de un web service se realiza normalmente con una de las dos opciones siguientes, ambas válidas y con sus ventajas relativas:

- **Top down (o Contract first):** Se construye el WSDL, utilizando algún editor especializado, y luego se generan las clases java y archivos de configuración con alguna herramienta.



- **Bottom up:** Se construye la clase java que implementa el servicio, incluyendo las clases de los parámetros y retorno (si no son básicas), y luego se genera el WSDL y los archivos de configuración.



## consideraciones generales

construcción top down vs. bottom up

Comparación:

- **Top down:**

- Mayor flexibilidad para definir tipos de datos, opciones de binding, entre otros.
- Mayor complejidad en la implementación.

- **Bottom up:**

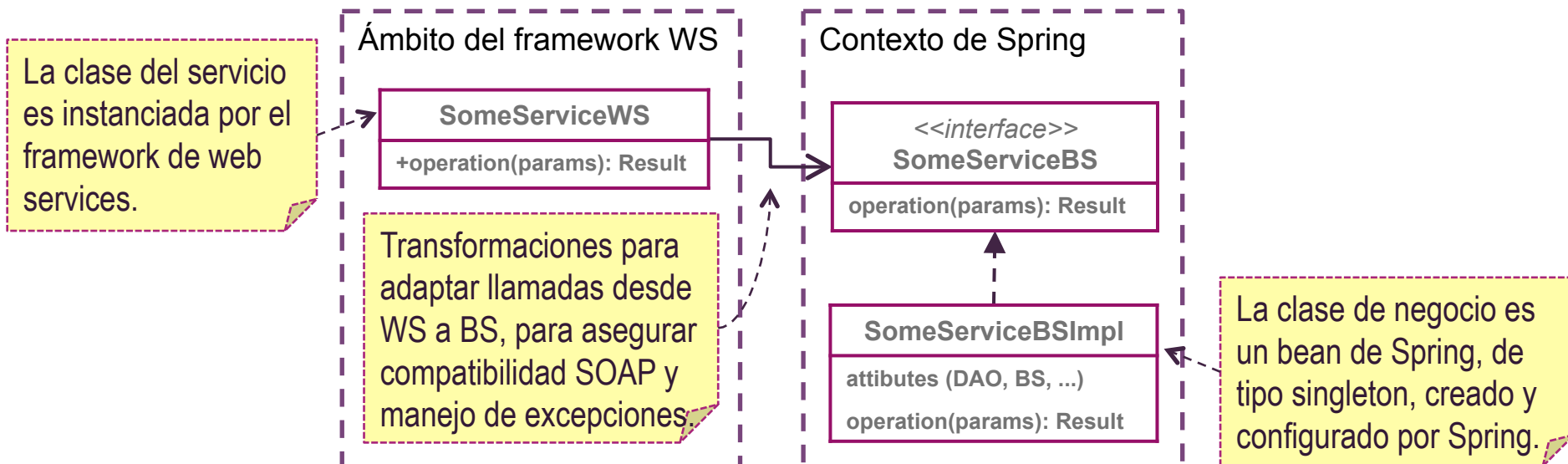
- Más sencilla de utilizar, ya que normalmente existe más experiencia en implementar clases java.
- Menor flexibilidad, ya que hay características que no se pueden definir en la clase.

Conclusión: elegir la más conveniente según cada caso. Si hay plataformas distintas, conviene que sea Top down.

## consideraciones generales

### compatibilidad con Spring

Cuando se utiliza Spring para la implementación de lógica de negocio, se deben tener en cuenta las siguientes consideraciones para publicarla como Web Service:



## consideraciones generales

### compatibilidad con Spring

Comentarios y conclusiones:

- Normalmente el bean de Spring es **Singleton**, y su ciclo de vida es manejado por el propio framework Spring.
- En cambio, el objeto asociado al Web Service publicado tiene un ciclo de vida manejado por el framework de Web Services utilizado. Casi todos instancian el objeto cada vez que lo utilizan.
- Por lo tanto, lo recomendable es siempre crear una clase adaptadora, **externa a Spring**, que sea la que publique el Web Service, excepto en los casos en que el framework está integrado con Spring, como CXF.
- La clase adaptadora permite además hacer las conversiones para asegurar la compatibilidad SOAP de los objetos enviados como parámetro y el retorno.
- La clase adaptadora permite el manejo de excepciones por separado de la clase de negocio.

## consideraciones generales

### utilización de frameworks

En los siguientes apartados se utilizan los frameworks Java más conocidos para Web Services. El enfoque es **completamente práctico**, por lo que prácticamente todo el contenido **está en los ejercicios**. Para cada framework se realizan las siguientes actividades:

- Breve descripción teórica.
- Explicación sobre publicación del servicio utilizando Bottom Up y Top Down.
- Caso práctico de publicación con Bottom Up.
- Caso práctico de publicación con Top Down.
- Explicación sobre consumo del servicio.
- Caso práctico de consumo.

Se incluye la utilización de las herramientas normalmente utilizadas en proyecto. La base lógica es el caso de negocio, lo que le da un enfoque más real a los ejercicios.