U07. Servicios Web

1.- Introducción

Hay ocasiones en que la misma información es utilizada por diferentes aplicaciones. Para compartir la información que gestiona una aplicación, es suficiente con dar acceso a la base de datos en que se almacena. Pero cuantas más aplicaciones utilicen los mismos datos, más posibilidades hay de que se generen errores en los mismos. Además, existen otros inconvenientes:

- Si ya hay una aplicación funcionando, y se ha programado la lógica de negocio correspondiente, ésta funcionalidad no se podrá aprovechar en otras aplicaciones si utilizan directamente la información almacenada en la base de datos.
- Si quieres poner la base de datos a disposición de terceros, éstos necesitarán conocer su estructura. Y al dar acceso directo a los datos, será complicado mantener el control sobre las modificaciones que se produzcan en los mismos.

Gran parte de la información que gestionan las aplicaciones web ya está disponible para que otros la utilicen. Por ejemplo, si se quiere conocer el precio de un producto en una tienda web, basta con buscar ese producto en la página en que se listan los productos. Pero, para que esa misma información la pueda obtener un programa, éste tendría que contemplar un procedimiento para buscar el producto concreto dentro de las etiquetas HTML de la página y extraer su precio.

Para facilitar esta tarea existen los servicios web que permiten que dos aplicaciones intercambien información usando HTTP. La aplicación servidor puede ofrecer un punto de acceso a la información que quiere compartir. De esta forma controla y facilita el acceso a la misma por parte de otras aplicaciones.

Las aplicaciones clientes del servicio no necesitan conocer la estructura interna de almacenamiento. En lugar de tener que programar un mecanismo para localizar la información en la base de datos, tienen un punto de acceso directo a la que les interesa.

En el ejemplo de la tienda web, podríamos obtener la información de un producto (nombre, descripción, precio, etc.) enviando el código del producto.

2.- Servicios Web

Existen numerosos protocolos de nivel aplicación que permiten la comunicación entre aplicaciones a través de internet, como son FTP, HTTP, SMTP, POP3, TELNET, SSH, etc. En todos ellos se definen un servidor y un cliente. El servidor es la aplicación que está esperando conexiones (escuchando) por parte de un cliente. El cliente es la aplicación que inicia la comunicación. Cada uno de estos protocolos tiene asignado un puerto “estándar” (TCP o UDP) en el que escuchan las aplicaciones servidores.

Los servicios web se crearon para permitir el intercambio de información sobre la base del protocolo HTTP (de ahí el término web). En lugar de definir su propio protocolo para transportar las peticiones de información, utilizan HTTP para este fin. La respuesta obtenida no será una página web, sino la información que se solicitó. De esta forma pueden funcionar sobre cualquier servidor web; y, lo que es aún más importante, utilizando el puerto 80 reservado para este protocolo. Por tanto, cualquier aplicación que pueda consultar una página web, podrá también solicitar información de un servicio web.

Hay al menos dos cuestiones que debería resolver un servicio web para poder funcionar adecuadamente:

- Cómo se transmite la información. Si se va a usar HTTP para las peticiones y las respuestas, el cliente y el servidor tendrán que ponerse de acuerdo en la forma de enviarlas. Por ejemplo, qué hace el cliente para indicar que quiere conocer el PVP del artículo con código X, y cómo envía el servidor la respuesta obtenida.
- Cómo se publican las funciones a las que se puede acceder en un servidor determinado. Esto es opcional pero muy útil. El cliente puede saber que la función del servidor que tiene que utilizar se llama `getPVPArticulo`, y que debe recibir como parámetro el código del artículo. Pero si no lo sabe, sería útil que hubiera un mecanismo donde pudiera consultar las funciones que existen en el servidor y cómo se utiliza cada una.

Cada uno de los métodos que podemos utilizar hoy en día para crear un servicio web responde a estas preguntas de formas distintas. Para la primera cuestión, nosotros veremos el protocolo SOAP, que utiliza el lenguaje XML para intercambiar información. En cuanto a la segunda cuestión, la resolveremos con un lenguaje llamado WSDL, que también está basado en XML y fue creado para describir servicios web e indicar cómo se debe acceder a un servicio y utilizarlo.

2.1.- SOAP (Simple Object Access Protocol)

SOAP es un protocolo que indica cómo deben ser los mensajes que se intercambien el servidor y el cliente, cómo deben procesarse éstos, y cómo se relacionan con el protocolo que se utiliza para transportarlos de un extremo a otro de la comunicación (en el caso de los servicios web, este protocolo será HTTP).

Aunque nosotros vamos a utilizar HTTP para transmitir la información, SOAP no requiere el uso de un protocolo concreto para transmitir la información. SOAP se limita a definir las reglas que rigen los mensajes que se deben intercambiar el cliente y el servidor. Cómo se envíen esos mensajes no es relevante desde el punto de vista de SOAP que utiliza XML para transmitirlos entre el cliente (petición) y el servidor (respuesta) web.

Ejemplo. Si implementamos un servicio web para informar sobre el precio de los artículos que se venden en una tienda web, una petición de información para el artículo con código '***KSTMSDHC8GB***' podría ser de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1=""
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <soap:Body>
        <ns1:getPVP>
            <param0 xsi:type="xsd:string">KSTMSDHC8GB</param0>
        </ns1:getPVP>
    </soap:Body>
</soap:Envelope>
```

Y su respectiva respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1=""
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <soap:Body>
        <ns1:getPVPResponse>
            <return xsi:type="xsd:string">10.20</return>
        </ns1:getPVPResponse>
    </soap:Body>
</soap:Envelope>
```

En un mensaje SOAP, como mínimo debe figurar un elemento "**`Envelope`**", que es lo que identifica al documento como un mensaje y se deben declarar al menos los siguientes espacios de nombres:

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

El espacio de nombres que se utilice para el elemento `Envelope` indica la versión del protocolo utilizado. En la versión **1.1** (la del ejemplo anterior), el espacio de nombres es: `http://schemas.xmlsoap.org/soap/envelope/`. En la versión **1.2** se debe utilizar: `http://www.w3.org/2003/05/soap-envelope`.

Al cambiar la versión de también se deben cambiar los espacios de nombres relativos al estilo de codificación. En la versión **1.1**, se debe utilizar: `http://schemas.xmlsoap.org/soap/encoding`/, y en la versión **1.2**: `http://www.w3.org/2003/05/soap-encoding`.

Como primer miembro del elemento `"Envelope"`, puede haber de forma opcional un elemento `Header`. Si existe, puede contener varios elementos con información adicional sobre cómo procesar el mensaje. A continuación debe figurar obligatoriamente un elemento `"Body"`, que es dónde se incluye, dependiendo del tipo de mensaje, la petición o la respuesta.

Sería muy complejo programar un servicio web que procesase el XML recibido en cualquier petición y generase el XML relativo a la respuesta correspondiente. Existen mecanismos de ayuda para restringir y definir qué peticiones pueden hacerse.

2.2.- WSDL (Web Services Description Language)

Una vez creado un servicio web, se puede programar el correspondiente cliente y comenzar a utilizarlo. Como el servicio lo hemos creado nosotros, sabremos cómo acceder a él: dónde está accesible, qué parámetros recibe, cuál es la funcionalidad que aporta, y qué valores devuelve. Si se quiere que el servicio web esté accesible a aplicaciones desarrolladas por otros programadores, deberá indicarse cómo usarlo creando un documento WSDL que describa el servicio.

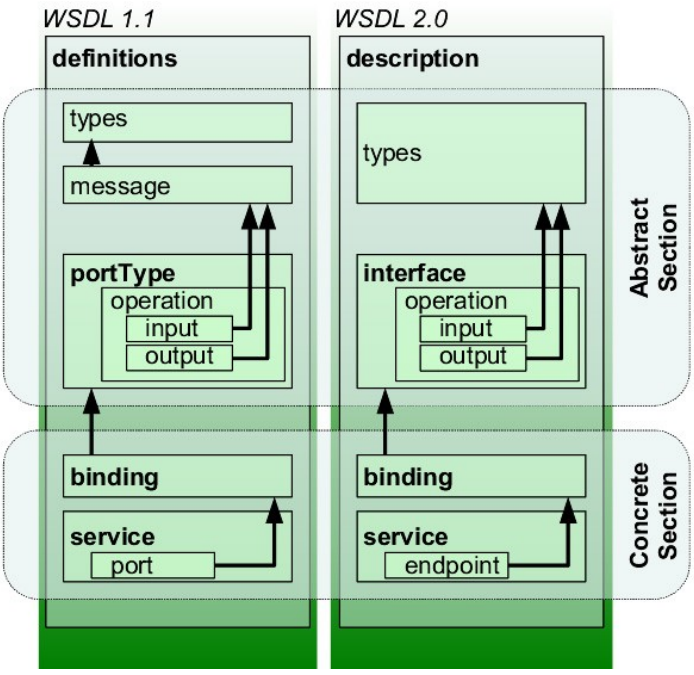
WSDL es un lenguaje basado en XML que utiliza unas reglas determinadas para generar el documento de descripción de un servicio web, que se pone a disposición de los posibles usuarios del servicio (normalmente se accede al documento añadiendo `"?wsdl"` a la URL del servicio).

El espacio de nombres de un documento WSDL es: `http://schemas.xmlsoap.org/wsdl/`, aunque en un documento WSDL se suelen utilizar también otros espacios de nombres. La estructura de un documento WSDL es la siguiente:

```
<definitions
    name="…"
    targetNamespace="http://…"
    xmlns:tns="http://…"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    …
>
    <types>
        …
    </types>
    <message>
        …
    </message>
    <portType>
        …
    </portType>
    <binding>
        …
    </binding>
    <service>
        …
    </service>
</definitions>
```

El objetivo de cada una de las secciones del documento es el siguiente:

- `types`. Incluye las definiciones de los tipos de datos que se usan en el servicio.
- `message`. Define conjuntos de datos, como la lista de parámetros que recibe una función o los valores que devuelve.
- `portType`. Cada `portType` es un grupo de funciones que implementa el servicio web. Cada función se define dentro de su `portType` como una operación (`operation`).
- `binding`. Define cómo va a transmitirse la información de cada `portType`.
- `service`. Contiene una lista de elementos de tipo `port`. Cada `port` indica dónde (en qué) se puede acceder al servicio web.



Las secciones anteriores son las correspondientes a la versión **1.1** de WSDL. En la versión **2.0**, los conceptos y la nomenclatura cambia ligeramente; por ejemplo, lo que en **1.1** es un `portType` se denomina `interface` en la versión **2.0**.

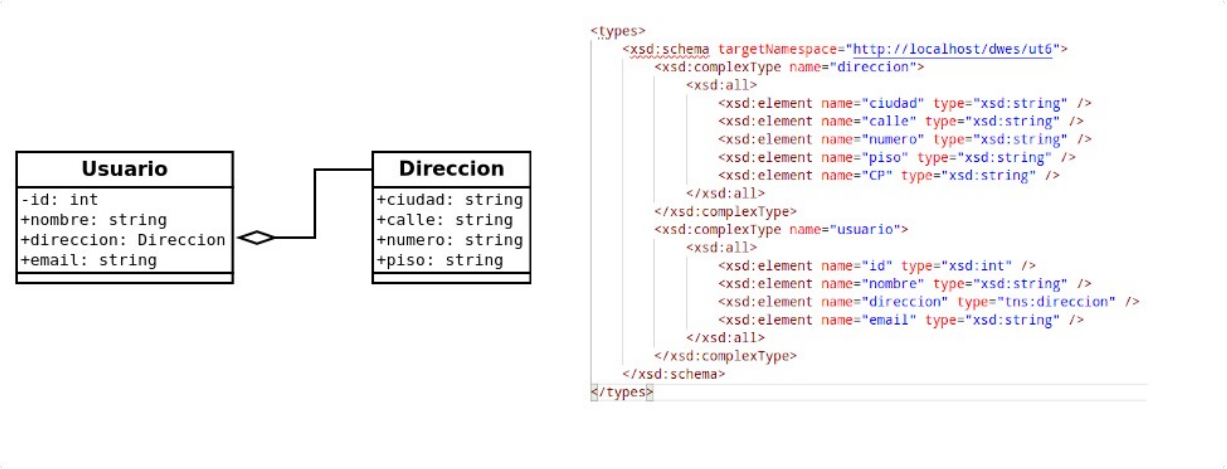
2.2.1.- Types

Existen servicios web sencillos a los que puedes pasar como parámetro un número o una cadena de texto (por ejemplo, las siglas de una moneda, USD), y te devuelven también un dato de un tipo simple, como un número decimal (la tasa de conversión actual). E igualmente existen también servicios web más elaborados, que pueden requerir o devolver un array de elementos, o incluso objetos.

Para crear y utilizar estos servicios, deberás definir los tipos de elementos que se transmiten: de qué tipo son los valores del array, o qué miembros poseen los objetos que maneja. La definición de tipos en se realiza utilizando la etiqueta `types`. Veamos un ejemplo:

```
<types>
    <xsd:schema targetNamespace="http://localhost/dwes/ut6">
        <xsd:complexType name="direccion">
            <xsd:all>
                <xsd:element name="ciudad" type="xsd:string" />
                <xsd:element name="calle" type="xsd:string" />
                <xsd:element name="numero" type="xsd:string" />
                <xsd:element name="piso" type="xsd:string" />
                <xsd:element name="CP" type="xsd:string" />
            </xsd:all>
        </xsd:complexType>
        <xsd:complexType name="usuario">
            <xsd:all>
                <xsd:element name="id" type="xsd:int" />
                <xsd:element name="nombre" type="xsd:string" />
                <xsd:element name="direccion" type="tns:direccion" />
                <xsd:element name="email" type="xsd:string" />
            </xsd:all>
        </xsd:complexType>
    </xsd:schema>
</types>
```

En el código anterior, se definen dos tipos de datos usando `XML Schema`: `direccion` y `usuario`. De hecho, los tipos `dirección` y `usuario` son la forma en que se definen en WSDL las clases para transmitir la información de sus objetos.



En WSDL, las clases se definen utilizando los tipos complejos de `XML Schema`. Al utilizar `all` dentro del tipo complejo, estamos indicando que la clase contiene esos miembros, aunque no necesariamente en el orden que se indica (si en lugar de `all` hubiésemos utilizado `sequence`, el orden de los miembros de la clase debería ser el mismo que figura en el documento).

Los métodos de la clase forman parte de la lógica de la aplicación y no se definen en el documento. Aunque en WSDL se puede usar cualquier lenguaje para definir los tipos de datos, es

aconsejable usar `XML Schema`, indicándolo dentro de la etiqueta `types` e incluyendo el espacio de nombres correspondiente en el elemento `definitions`.

```
<definitions
    name="WSDLusuario"
    targetNamespace="http://localhost/dwes/ut6"
    xmlns:tns="http://localhost/dwes/ut6"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    . . .
>
```

El otro tipo de datos que necesitaremos definir en los documentos son los **arrays**. Para definir un array, no existe en el `XML Schema` un tipo base adecuado que podamos usar. En su lugar, se utiliza el tipo `Array` definido en el esquema `enconding` de SOAP. Podríamos añadir un tipo array de usuarios al documento anterior haciendo:

```
<xsd:complexType name="ArrayOfusuario">
    <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
            <xsd:attribute ref="soapenc:arrayType" arrayType="tns:usuario[]" />
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Al definir un array en WSDL, se debe tener en cuenta que:

- El atributo `arrayType` se utiliza para indicar qué elementos contendrá el array.
- Se debe añadir al documento el espacio de nombres `encoding:`

```
<definitions
    name="WSDLusuario"
    targetNamespace="http://localhost/dwes/ut6"
    xmlns:tns="http://localhost/dwes/ut6"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    . . .
>
```

El nombre del array debería ser `ArrayOfXXX`, dónde `XXX` es el nombre del tipo de elementos que contiene el array. En muchas ocasiones no será necesario definir tipos propios, y por tanto en el documento no habrá sección `types`; será suficiente con utilizar alguno de los tipos propios de `XML Schema`, como `xsd:string`, `xsd:float` o `xsd:boolean`.

2.2.2.- Messages

El siguiente paso es indicar cómo se agrupan esos tipos para formar los parámetros de entrada y de salida. Veámoslo con un ejemplo. Siguiendo con el tipo ***usuario*** que acabamos de definir, podríamos crear en el servicio web una función ***`getUsuario()`*** para dar acceso a los datos de un usuario. Como parámetro de entrada de esa función vamos a pedir el `"id"` del usuario, y como valor de salida se obtendrá un objeto `"usuario"`. Por tanto, debemos definir los siguientes mensajes:

```
<message name="getUsuarioRequest">
    <part name="id" type="xsd:int"/>
</message>
<message name="getUsuarioResponse">
    <part name="getUsuarioReturn" type="tns:usuario"/>
</message>
```

Normalmente por cada función del servicio web se crea un mensaje para los parámetros de entrada, y otro para los de salida. Dentro de cada mensaje, se incluirán tantos elementos `part` como sea necesario. Cada mensaje contendrá un atributo `name` que debe ser único para todos los elementos de este tipo. Además, es aconsejable que el nombre del mensaje con los parámetros de entrada acabe en `Request`, y el correspondiente a los parámetros de salida en `Response`.

En un documento WSDL podemos especificar dos estilos de enlazado: `document` o `RPC`. La selección que hagamos influirá en cómo se transmitan los mensajes dentro de las peticiones y respuestas. Por ejemplo, un mensaje con estilo `document` podría ser:

```
<SOAP-ENV:Body>
    <producto>
        <codigo>KSTMSDHC8GB</codigo>
    </producto>
</SOAP-ENV:Body>
```

Y un mensaje con estilo `RPC` sería por ejemplo:

```
<SOAP-ENV:Body>
    <ns1:getPVP>
        <param0 xsi:type="xsd:string">KSTMSDHC8GB</param0>
    </ns1:getPVP>
</SOAP-ENV:Body>
```

El estilo de enlazado `RPC` está más orientado a sistemas de petición y respuesta que el `document` (más orientado a la transmisión de documentos en formato XML). En este estilo de enlazado, cada elemento `message` de WSDL debe contener un elemento `part` por cada parámetro (de entrada o de salida), y dentro de éste indicar el tipo de datos del parámetro mediante un atributo `type`, como se muestra en el ejemplo anterior.

Además, cada estilo de enlazado puede ser de tipo `encoded` o `literal` (aunque en realidad la combinación `document/encoded` no se utiliza). Al indicar `encoded`, estamos diciendo que vamos a usar un conjunto de reglas de codificación, como las que se incluyen en el propio protocolo (espacio de nombres `"http://schemas.xmlsoap.org/soap/encoding/"`), para convertir en XML los parámetros de las peticiones y respuestas.

El ejemplo anterior de `RPC` es en realidad `RPC/encoded`. Un ejemplo de un mensaje con estilo `RPC/literal` sería:

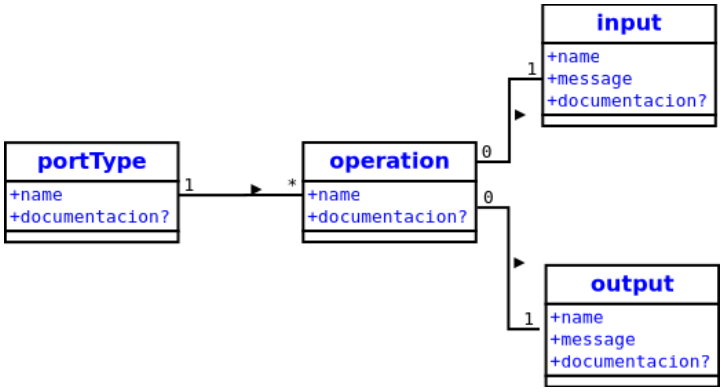
```
<SOAP-ENV:Body>
    <ns1:getPVP>
        <param0>KSTMSDHC8GB</param0>
    </ns1:getPVP>
</SOAP-ENV:Body>
```

En lo sucesivo, trabajaremos únicamente con estilo de enlazado `RPC/encoded`.

Las funciones que se crean en un servicio web, se conocen con el nombre de **operaciones** en un documento. En lugar de definirlas una a una, es necesario agruparlas en lo que en se llama `portType`. Un `portType` contiene una lista de funciones, indicando para cada función (`operation`) la lista de parámetros de entrada y de salida que le corresponden. Por ejemplo:

```
<portType name="usuarioPortType">
    <operation name="getUsuario">
        <input message="tns:getUsuarioRequest"/>
        <output message="tns:getUsuarioResponse"/>
    </operation>
</portType>
```

A no ser que estés generando un servicio web bastante complejo, el documento contendrá un único `portType`. Podrías necesitar dividir las funciones del servicio en distintos `portType` para, por ejemplo, utilizar un estilo de enlazado distinto para las funciones de cada grupo.



Cada `portType` debe contener un atributo `name` con el nombre (único para todos los elementos `portType`). Cada elemento `operation` también debe contener un atributo `name`, que se corresponderá con el nombre de la función que se ofrece. Además, en función del tipo de operación de que se trate, contendrá:

- Un elemento `input` para indicar funciones que no devuelven valor (su objetivo es sólo enviar un mensaje al servidor).
- Un elemento `input` y otro `output`, en este orden, para el caso más habitual: funciones que reciben algún parámetro, se ejecutan, y devuelven un resultado.

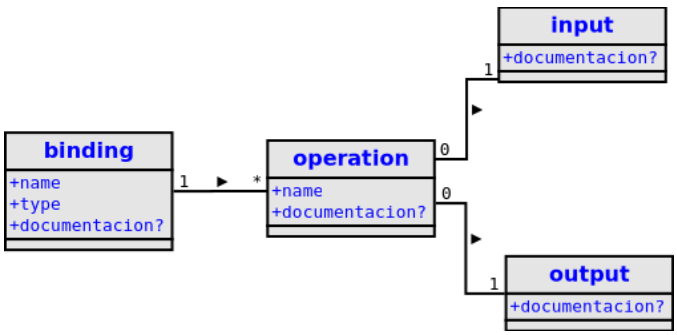
Es posible (pero muy extraño) encontrarse funciones a la inversa: sólo con un parámetro `output` (el servidor envía una notificación al cliente) o con los parámetros `output` e `input` por ese orden (el servidor le pide al cliente alguna información). Por tanto, al definir una función (un elemento `operation`) se debe tener cuidado con el orden de los elementos `input` y `output`.

Normalmente, los elementos `input` y `output` contendrán un atributo `message` para hacer referencia a un mensaje definido anteriormente.

2.2.3.- Binding

El siguiente elemento de un documento es `binding`. Antes comentábamos que existían distintos estilos de enlazado, que influían en cómo se debían crear los mensaje. En el elemento `binding` es dónde debes indicar que el estilo de enlazado de tu documento sea `RPC/encoded`.

Aunque es posible crear documentos con varios elementos `binding`, la mayoría contendrán solo uno (si no fuera así, sus atributos `name` deberán ser distintos). En él, para cada una de las funciones (`operation`) del `portType` que acabamos de crear, se deberá indicar cómo se codifica y transmite la información.



Para el `portType` anterior, podemos crear un elemento `binding` como el siguiente:

```
<binding name="usuarioBinding" type="tns:usuarioPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getUsuario">
        <soap:operation soapAction="http://localhost/dwes/ut6/getUsuario.php?getUsuario" />
        <input>
            <soap:body use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost/dwes/ut6" />
        </input>
        <output>
            <soap:body use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost/dwes/ut6" />
        </output>
    </operation>
</binding>
```

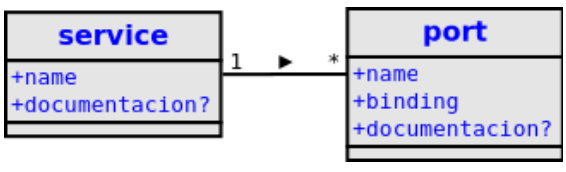
Fíjate que el atributo `type` hace referencia al `portType` creado anteriormente. El siguiente elemento indica el tipo de codificación (`RPC`) y, mediante la URL correspondiente, el protocolo de transporte a utilizar (HTTP). Obviamente, deberás añadir el correspondiente espacio de nombres al elemento raíz:

```
<definitions
    name="WSDLusuario"
    targetNamespace="http://localhost/dwes/ut6"
    xmlns:tns="http://localhost/dwes/ut6"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    . . .
>
```

El elemento `soap:operation` debe contener un atributo `soapAction` con la URL para esa función (`operation`) en particular. Dentro de él habrá normalmente un elemento `input` y otro `output` (los mismos que en la `operation` correspondiente). En ellos, mediante los atributos del elemento `soap:body`, se indica el estilo concreto de enlazado (`encoded` con su `encondingStyle` correspondiente).

2.2.4.- Service

Por último (y a Dios gracias), falta definir el elemento `service`. Normalmente sólo encontraremos un elemento `service` en cada documento. En él, se hará referencia al `binding` anterior utilizando un elemento `port`, y se indicará la en la que se puede acceder al servicio.



Por ejemplo:

```
<service name="usuario">
    <port name="usuarioPort" binding="tns:usuarioBinding">
        <soap:addresslocation="http://localhost/dwes/ut6/getUsuario.php"/>
    </port>
</service>
```

Para finalizar, veamos cómo quedaría el documento correspondiente a un servicio con una única función encargada de devolver el de un producto de la tienda web a partir de su código.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="WSDLgetPVP" targetNamespace="http://localhost/dwes/ut6"
    xmlns:tns="http://localhost/dwes/ut6"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getPVPRequest">
        <part name="codigo" type="xsd:string" />
    </message>
    <message name="getPVPResponse">
        <part name="PVP" element="xsd:float" />
    </message>
    <portType name="getPVPPortType">
        <operation name="getPVP">
            <input message="tns:getPVPRequest" />
            <output message="tns:getPVPResponse" />
        </operation>
    </portType>
    <binding name="getPVPBinding" type="tns:getPVPPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="getPVP">
            <soap:operation soapAction="http://localhost/dwes/ut6/getPVP.php?getPVP" />
            <input>
                <soap:body namespace="http://localhost/dwes/ut6"
                    use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body namespace="http://localhost/dwes/ut6"
                    use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
    </binding>
    <service name="getPVPService">
        <port name="getPVPPort" binding="tns:getPVPBinding ">
            <soap:address location="http://localhost/dwes/ut6/getPVP.php" />
        </port>
    </service>
</definitions>
```

3.- PHP SOAP

De las posibilidades que hay para utilizar SOAP en PHP vamos a utilizar la extensión que viene incluida con el lenguaje a partir de su versión 5: `"PHP SOAP"`. Gracias a `"PHP SOAP"`, se pueden crear y utilizar de forma sencilla servicios web en las aplicaciones. Es compatible con las versiones `"SOAP 1.1/1.2"`, así como con `"WSDL 1.1/2.0"`, aunque no permite la generación automática del documento a partir del servicio web programado.

Para poder usar la extensión, deberás comprobar si ya se encuentra disponible (por ejemplo, consultando la salida obtenida por la función `phpinfo()`). Normalmente la extensión se instala al instalar PHP, por lo que no habría que hacer nada para utilizarla.

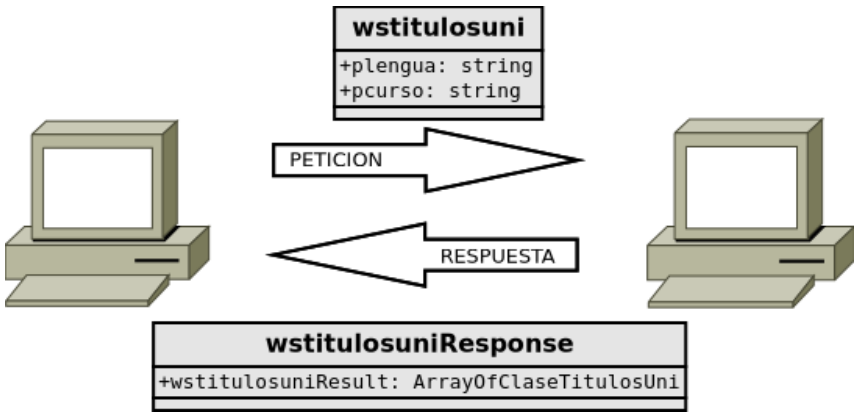
Las dos clases principales que deberás utilizar en tus aplicaciones son "`SoapClient`" y "`SoapServer`". La primera te permitirá comunicarte con un servicio web, y con la segunda podrás crear tus propios servicios.

3.1.- Uso de servicios web

Imaginemos que se quieren conocer lo estudios oficiales que ofrece una Universidad para un curso concreto. Lo primero que se necesita es una universidad que ofrezca esta información via WSDL. Para el siguiente ejemplo usaremos la Universidad de Alicante.

Para crear un cliente del servicio, se deben de conocer los detalles del mismo (como mínimo, los parámetros de entrada y salida que se deben usar, y cuál es la del servicio) y emplear en

el código la clase SoapClient. Los detalles del servicio se encuentran disponibles en "*https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl*".



- El alias del espacio de nombres correspondiente al `XML Schema` que utiliza el documento es:

```
<wsdl:definitions targetNamespace="http://UASI/WS_GESTDOCENTE.wsdl">
```

- En el documento se puede observar lo siguiente, se espera un elemento de tipo "wstitulosuni" que serán dos códigos: "plengua" y "pcurso" de tipo string, además en ese orden:

```
<s:element name="wstitulosuni">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="plengua" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="pcurso" type="s:string"/>
        </s:sequence>
    </s:complexType>
</s:element>
```

- La respuesta "wstitulosuniResponse" será un elemento de tipo "wstitulosuniResult".

```
<s:element name="wstitulosuniResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="wstitulosuniResult" type="tns:ArrayOfClaseTitulosUni"/>
        </s:sequence>
    </s:complexType>
</s:element>
```

- El tipo wstitulosuniResult es un **array** de tipo ClaseTitulosUni (arrayOfClaseTitulosUni). Este **array** tendrá los elementos siguientes:

```
<s:complexType name="ClaseTitulosUni">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="codigo" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="nombre" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="tipo" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="area" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="url" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="imagen" type="s:string"/>
    </s:sequence>
</s:complexType>
```

El estilo de enlazado es "document/literal" (nosotros hemos visto el "RPC/encoded" solamente), por lo que los elementos de tipo "message" tienen un formato distinto. Sin embargo, en base a su contenido (observense los elementos que terminan en "Soap") se puede deducir también que:

- El nombre de la función a la que se debe llamar es "wstitulosuni".

```
<wsdl:operation name="wstitulosuni">
```

- Como parámetro de entrada se tiene que pasar un elemento de tipo "wstitulosuni" (dos string), y devolverá un elemento "wstitulosuniResponse" (un array).

```
<wsdl:message name="wstitulosuniSoapIn">
    <wsdl:part name="parameters" element="tns:wstitulosuni"/>
</wsdl:message>
<wsdl:message name="wstitulosuniSoapOut">
    <wsdl:part name="parameters" element="tns:wstitulosuniResponse"/>
</wsdl:message>
 . . .
<wsdl:portType name="pub_gestdocenteSoap">
    . . .
    <wsdl:operation name="wstitulosuni">
        <wsdl:input message="tns:wstitulosuniSoapIn"/>
        <wsdl:output message="tns:wstitulosuniSoapOut"/>
    </wsdl:operation>
    . . .
</wsdl:portType>
```

- La URL para acceder al servicio, se puede ver más abajo, de igual manera se puede observar que el servicio se ofrece para versión **1.1** y **1.2** de SOAP.

```
<wsdl:service name="pub_gestdocente">
    <wsdl:port name="pub_gestdocenteSoap" binding="tns:pub_gestdocenteSoap">
        <soap:address location="https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx"/>
    </wsdl:port>
    <wsdl:port name="pub_gestdocenteSoap12" binding="tns:pub_gestdocenteSoap12">
        <soap12:address location="https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx"/>
    </wsdl:port>
    . . .
</wsdl:service>
```

Con la información anterior, para utilizar el servicio desde PHP se crea un nuevo objeto de la clase "SoapClient". Como el servicio tiene un documento asociado, en el constructor se indica dónde se encuentra:

```
$cliente=new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
```

Y para realizar la llamada a la función "wstitulosuni", se incluyen los parámetros en un **array**:

```
$cliente=new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");

$parametros=[
    'plengua'=>'es',
    'pcurso'=>'2023'
];
$titulos=$cliente->wstitulosuni($parametros);
```

La llamada devuelve un objeto de una clase predefinida en PHP llamada: StdClass. Si hacemos "var_dump($titulos)" obtenemos lo siguiente:

```
object(stdClass)[2]
  public 'wstitulosuniResult' =>
    object(stdClass)[3]
      public 'ClaseTitulosUni' =>
        array (size=113)
```

Vemos que nos ha devuelto un objeto `stdClass` dentro podemos observar que nos ha devuelto 117, títulos. Si recorremos `$titulos` con el código siguiente:

```
<?php declare(strict_types=1); ?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Titulaciones Universidad de Alicante</title>
    <style>
        th, td { border: 1px solid black;}
        th { background-color: orange; }
        tr:nth-child(even) { background-color: lightblue;}
        tr:nth-child(odd) { background-color: lightgray;}
    </style>
</head>
<body>
    <?php
```

```php
        $cliente=new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
        $parametros=[
            'plengua'=>'es',
            'pcurso'=>'2023'
        ];
        $titulos=$cliente->wstitulosuni($parametros);
        echo "<table>\n";
        echo "<thead><tr><th>Código</th><th>Título</th><th>Tipo</th><th>Área</th></tr></thead>\n";
        echo "<tbody>";
        foreach($titulos->wstitulosuniResult->ClaseTitulosUni as $titulo){
            if ($titulo->tipo=='0') {
                $tipo="Presencial";
            } else {
                $tipo="SemiPresencial";
            }
            echo "<tr><td>$titulo->codigo</td><td><a href='$titulo->url'>$titulo-
>nombre</a></td><td>$tipo</td><td>$titulo->area</td></tr>\n";
            //<td><img src='$titulo->imagen'/></td></tr>\n";
        }
        echo "</tbody>";
        echo "</table>\n";
        echo "<br/><br/>Total títulos: ",count($titulos->wstitulosuniResult->ClaseTitulosUni),"<br/>\n";
    ?>
</body>
</html>
```

Veremos lo siguiente:

Código	Título	Tipo	Área
C001	GRADO EN GEOGRAFÍA Y ORDENACIÓN DEL TERRITORIO	Presencial	Ciencias Sociales y Jurídicas
C002	GRADO EN HISTORIA	Presencial	Artes y Humanidades
C003	GRADO EN HUMANIDADES	Presencial	Artes y Humanidades
C004	GRADO EN TURISMO	Presencial	Ciencias Sociales y Jurídicas
C005	GRADO EN ESTUDIOS ÁRABES E ISLÁMICOS	Presencial	Artes y Humanidades
C006	GRAU EN FILOLOGIA CATALANA	Presencial	Artes y Humanidades
C007	GRADO EN ESTUDIOS FRANCESES	Presencial	Artes y Humanidades
C008	GRADO EN ESPAÑOL: LENGUA Y LITERATURAS	Presencial	Artes y Humanidades
C009	GRADO EN ESTUDIOS INGLESES	Presencial	Artes y Humanidades
C010	GRADO EN TRADUCCIÓN E INTERPRETACIÓN	Presencial	Artes y Humanidades
C011	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. ALEMÁN	Presencial	Artes y Humanidades
C012	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. INGLÉS	Presencial	Artes y Humanidades
C013	GRADO EN TRADUCCIÓN E INTERPRETACIÓN. FRANCÉS	Presencial	Artes y Humanidades
C051	GRADO EN GEOLOGÍA	Presencial	Ciencias
C052	GRADO EN MATEMÁTICAS	Presencial	Ciencias
C053	GRADO EN QUÍMICA	Presencial	Ciencias
C054	GRADO EN BIOLOGÍA	Presencial	Ciencias
C055	GRADO EN CIENCIAS DEL MAR	Presencial	Ciencias
C056	GRADO EN ÓPTICA Y OPTOMETRÍA	Presencial	Ciencias de la Salud
C057	GRADO EN FÍSICA	Presencial	Ciencias
C058	GRADO EN GASTRONOMÍA Y ARTES CULINARIAS	Presencial	Ciencias Sociales y Jurídicas
C101	GRADO EN GESTIÓN Y ADMINISTRACIÓN PÚBLICA	Presencial	Ciencias Sociales y Jurídicas
C102	GRADO EN DERECHO	Presencial	Ciencias Sociales y Jurídicas
C103	GRADO EN CRIMINOLOGÍA	Presencial	Ciencias Sociales y Jurídicas
C104	GRADO EN RELACIONES LABORALES Y RECURSOS HUMANOS	Presencial	Ciencias Sociales y Jurídicas
C105	GRADO EN RELACIONES INTERNACIONALES	Presencial	Ciencias Sociales y Jurídicas
C110	DOBLE GRADO EN DERECHO Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS	Presencial	Ciencias Sociales y Jurídicas
C111	DOBLE GRADO EN DERECHO Y CRIMINOLOGÍA	Presencial	Ciencias Sociales y Jurídicas

NOTA. En PHP existe una clase predefinida en el lenguaje que se llama "stdClass", que no tiene ni propiedades, ni métodos, ni clase padre; es una clase vacía. Podemos usar esta clase cuando necesitamos un objeto genérico al que luego le podremos añadir propiedades. Hay que tener en cuenta, que esta clase no pertenece a la clase de la que heredan todas las clases.

El constructor de la clase "SoapClient" puede usarse de dos formas: indicando un documento WSDL, o sin indicarlo. En el primer caso, la extensión examina la definición del servicio y establece las opciones adecuadas para la comunicación, con lo cual el código necesario para utilizar un servicio es bastante simple.

En el segundo caso, si no se indica en el constructor un documento (bien porque no existe, o porque se necesita configurar manualmente alguna opción), el primer parámetro debe ser "null", y las opciones para comunicarse con el servicio se tendrán que establecer en un **array** que se pasa como segundo parámetro.

Si el servicio dispone del correspondiente documento , en muchos lenguajes de programación existen utilidades que facilitan aún más el desarrollo de aplicaciones que lo utilicen. Nosotros vamos a ver la herramienta Wsdl2PhpGenerator.

Se trata de un script escrito en lenguaje PHP que examina un documento WSDL y genera un fichero PHP específico para comunicarse con el servicio web correspondiente. Su uso es muy sencillo: se ejecuta pasándole como parámetro la URL en que se encuentra el documento WSDL, y como resultado genera un fichero con código PHP.

Lo podemos instalar independientemente o con "`composer`". Se recomienda hacerlo de esta última manera pues podemos indicar entre otras cosas el "`namespace`" para las clases que se creen. El enlace en **Packagist** es el siguiente: [`wsdl2phpgenerator`].

La instalación es sencilla, en la carpeta donde vayamos a hacerlo nos creamos la estructura usual de un proyecto, las carpetas "`public`" y "`src`". La carpeta "`public`" será para lo que vayamos a publicar y en "`src`" meteremos las clases.

Iniciamos **Composer** (el comando es "`composer init`") integramos el `autoload` e instalamos la librería en cuestión, esto último también podemos hacerlo una vez generado el archivo: "`composer.json`" con el comando "`composer require wsdl2phpgenerator/wsdl2phpgenerator`". Al final se nos habrá creado la carpeta "`vendor`" y nuestro archivo: "`composer.json`" deberá ser parecido a este:

```
{
    "name": "usuario/usuario",
    "description": "Prueba wsdl",
    "type": "project",
    "license": "gnu/gpl",
    "config": {
        "optimize-autoloader": true
    },
    "autoload": {
        "psr-4": {
            "Clases\\": "src"
        }
    },
    "authors": [
        {
            "name": "usuario",
            "email": "usuario@correo.es"
        }
    ],
    "require": {
        "wsdl2phpgenerator/wsdl2phpgenerator": "^3.4"
    }
}
```

En la carpeta "`public`" nos crearemos el archivo "`generar.php`" para generarnos las clases a partir del documento . (Una vez generadas deberíamos borrar este archivo) Si consultamos la documetación del proyecto es fácil crear este archivo:

```
<?php
//utilizamos el autoload de composer
require '../vendor/autoload.php';

use Wsdl2PhpGenerator\Generator;
use Wsdl2PhpGenerator\Config;

$generator = new Generator();
$generator->generate(
    new Config([
        'inputFile' => "https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl", //wsdl
        'outputDir' => '../src',  //directorio donde vamos a generar las clases
        'namespaceName' => 'Clases'  //namespace que vamos a usar con ellas (lo indicamos en composer)
    ])
);
```

Se puede observar que ahora en "`src`" tenemos una serie de clases creadas. Ya estamos en condiciones de traernos los datos. Nos creamos el archivo "`titulaciones2019.php`" por ejemplo con el contenido siguiente:

```php
<?php
    require '../vendor/autoload.php';

    use Clases\Pub_gestdocente;
    use Clases\wstitulosuni;

    $service = new Pub_gestdocente();
    $request=new wstitulosuni('es', '2020');
    $titulos=$service->wstitulosuni($request);

    var_dump($titulos);
```

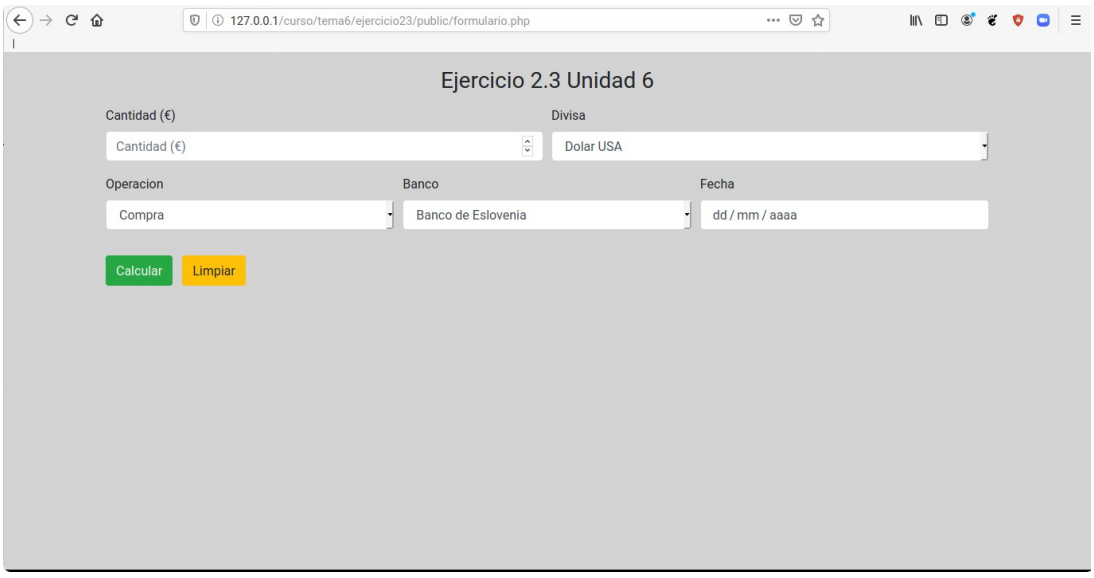
La diferencia es que ahora `$titulos` es un **array** de objetos de la clase: `"ArrayOfClaseTitulosUni"` y no de la clase especial `"stdclass"`.

Ejercicio resuelto (Ya no existe el servicio :( )

Inspecciona el servicio disponible en: `"http://webservices.gama-system.com/exchangerates.asmx?WSDL"`, que ofrece información sobre el cambio de Euros a distintas divisas y viceversa. En el documento `wsdl` que puedes ver cuando pinchas en un enlace aparece una función `"ConvertToForeign"` que da el cambio del Euro a distintas divisas, en una fecha determinada y por un Banco (Esloveno) determinado. Investigando un poco en la página: `"http://webservices.gama-system.com/"` y con ayuda del traductor podemos investigar los parámetros. Se deja en retroalimentación un ayuda sobre este tema.

Con la ayuda de "`wsdl2PhpGenerator`" nos haremos un formulario donde una vez elegida la cantidad en **Euros**, la divisa a la que los cambiaremos, el tipo de operación (compra, venta, intermedio), el banco, de los disponibles, del que buscamos la información y la fecha, nos ofrecerá el cambio buscado o el error caso de haberlo.

Para simplificar ofreceremos el cambio para simplificar solo para tres divisas "Dolar ", "Libra Esterlina" y "Yen Japonés", los códigos son respectivamente `"USD", "GBP", "JPY"`.

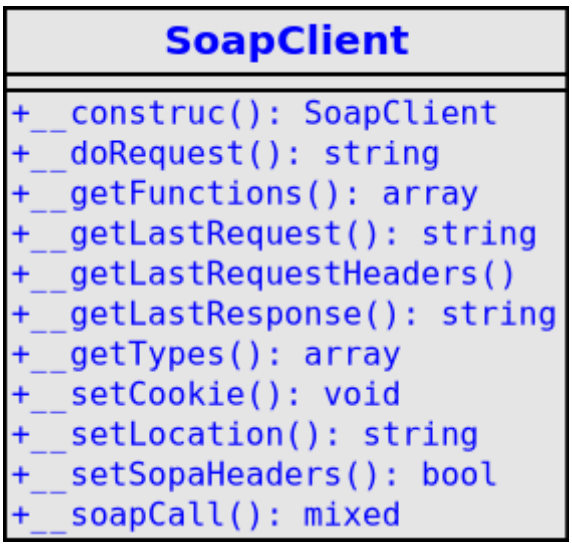


Enlace donde se explica el tipo de dato esperado: `"http://webservices.gama-system.com/descriptions-ER-slo.asp?strFunction=ConvertToForeign#ConvertToForeign"`

En este otro puedes ver los códigos de la monedas extranjeras (`strCurrency`) y la operaciones soportadas (`intRank`), con ayuda del traductor podemos observar que 1 es para la compra. 2 para el intermedio y 3 para la venta.

`"http://webservices.gama-system.com/descriptions-ER-slo.asp?strFunction=Common#Common"`

Échale un vistazo a la solución propuesta y compárala con la tuya: Solución propuesta.



Si estás usando un documento para acceder al servicio web, la clase `SoapClient` implementa dos métodos que muestran parte de la información que contiene; concretamente, los tipos de datos definidos por el servicio, y las funciones que ofrece. Para conocer esta información, una vez creado el objeto, debes utilizar los métodos `__getTypes` y `__getFunctions` respectivamente.

```php
<?php
$cliente = new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
$funciones = $cliente->__getFunctions();
echo "<ul>";
foreach ($funciones as $k => $v) {
    echo "<li><code>$v</code></li>";
}
echo "</ul>";
```

```php
<?php
$cliente = new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
$tipos = $cliente->__getTypes();
echo "<ul>";
foreach ($tipos as $k => $v) {
    echo "<li><code>$v</code></li>";
}
echo "</ul>";
```

El resultado obtenido es:

- wsplanesdepResponse wsplanesdep(wsplanesdep $parameters)
- wsdptoasigResponse wsdptoasig(wsdptoasig $parameters)
- wsareasdptoResponse wsareasdpto(wsareasdpto $parameters)
- wsdptocenResponse wsdptocen(wsdptocen $parameters)
- wsasiplanResponse wsasiplan(wsasiplan $parameters)
- wscalificacionesResponse wscalificaciones(wscalificaciones $parameters)
- wsconvocatoriasResponse wsconvocatorias(wsconvocatorias $parameters)
- wsgruposasiResponse wsgruposasi(wsgruposasi $parameters)
- wsconvemovilidadResponse wsconvemovilidad(wsconvemovilidad $parameters)
- wstitulospropiosuniResponse wstitulospropiosuni(wstitulospropiosuni $parameters)
- wsreqidiomablplanResponse wsreqidiomablplan(wsreqidiomablplan $parameters)

- struct wsagrupaciones { string plengua; string pcurso; string pcodcen; string pcodest; }
- struct wsagrupacionesResponse { ArrayOfClaseAgrupaciones wsagrupacionesResult; }
- struct ArrayOfClaseAgrupaciones { ClaseAgrupaciones ClaseAgrupaciones; }
- struct ClaseAgrupaciones { string idagrupa; string descagr; string enlaceagrp; string codcen; string desccen; string codest; string nomest; string numcurso; string nomturno; }
- struct wsfechaexamenesasi { string plengua; string pcodest; string pcurso; string pcodasi; string pcodconvoc; string porden; }
- struct wsfechaexamenesasiResponse { ArrayOfClaseFechaExamenesAsi wsfechaexamenesasiResult; }
- struct ArrayOfClaseFechaExamenesAsi { ClaseFechaExamenesAsi ClaseFechaExamenesAsi; }
- struct ClaseFechaExamenesAsi { string id; string fecha; string codasi; string nomasi; string codgrp; string codconv; string conv; string observaciones; string codest; string numcurso; string desccurso; string complform; string fechareal; string horaini; string horafin; string aula; string aulasig; string descaula; string dia; string orden; }

Donde las funciones aparecen duplicadas, dado que el servicio web, como ya vimos, ofrece dos versiones de la misma: una para **1.1** y otra para **1.2.**

La extensión "PHP SOAP" también incluye opciones de depuración muy útiles para averiguar qué está pasando cuando la conexión al servicio web no funciona como debería. Para habilitarlas, cuando hagas la llamada al constructor de la clase `SoapClient`, debes utilizar la opción `trace` en el array de opciones del segundo parámetro.

```php
$cliente = new SoapClient(
    "https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl",
    array('trace'=>true)
);
```

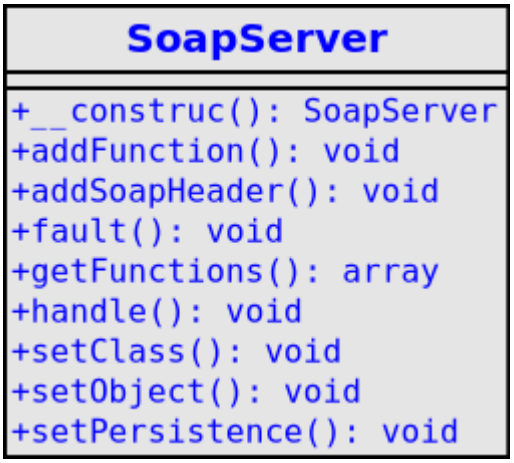
Existen bastantes opciones que se pueden utilizar con el constructor "SoapClient", pero si el servicio web dispone de un documento WSDL, normalmente no se necesitará utilizar ninguna. En caso contrario se debe definir al menos las opciones "location" (la URL en la que se encuentra el servicio) y “uri” (la ruta (**target**) de espacio de nombres (**namespace**) del servicio SOAP).

Una vez activada la depuración, podrás utilizar los siguientes métodos para revisar los últimos mensajes enviados y recibidos.

Métodos para depuración de la clase `SoapClient`

Método	Significado
`__getLastRequest`	Devuelve el correspondiente a la última petición enviada.
`__getLastRequestHeaders`	Devuelve el correspondiente a los encabezados de la última petición enviada.
`__getLastResponse`	Devuelve el correspondiente a la última respuesta recibida.
`__getLastResponseHeaders`	Devuelve el correspondiente a los encabezados de la última respuesta recibida.

3.2.- Creación de servicios web



En PHP SOAP, para crear un servicio web, se debe utilizar la clase `SoapServer`. Veamos un ejemplo sencillo. En una carpeta por ejemplo "Tema07/ServicioWeb" crearemos las carpetas "clienteSoap" y "servidorSoap" (lógicamente puedes usar los nombre que quieras). Dentro de "servidorSoap" crearemos el archivo "servidor.php" con el contenido siguiente:

```php
<?php
    declare(strict_types=1);
    class Operaciones
    {
        public function resta(float $a, float $b):float {
            return $a - $b;
        }
        public function suma(float $a,float $b):float {
```

```
            return $a + $b;
        }
        public function saludo(string $texto):string {
            return "Hola $texto";
        }
    }
    $uri='http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidorSoap';
    $parametros=['uri'=>$uri];
    try {
        $server = new SoapServer(NULL, $parametros);
        $server->setClass('Operaciones');
        $server->handle();
    } catch (SoapFault $f) {
        die("error en server: " . $f->getMessage());
    }
?>
```

El código anterior crea un servicio web con tres funciones: `suma`, `resta y saludo`. Cada función recibe unos parámetros y devuelve un valor. Para añadir las funciones de la clase **"Operaciones"** a nuestro `"Servidor Soap"` hemos añadido `"$server->setClass('Operaciones")`. Si hubiésemos implementados las funciones directamente, sin usar una clase, tendríamos que haberlas añadido usando `"$server->addFunction('nombre')"` para cada función, antes de `"$server->handle()"`. El método `handle` es el encargado de procesar las peticiones, recogiendo los datos que se reciban utilizando `POST` por HTTP.

Para consumir este servicio, en la misma carpeta crearemos el fichero `"consumir.php"` con el siguiente código:

```
<?php
    declare(strict_types=1);
    $url = 'http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php';
    $uri = 'http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidorSoap';
    $paramSaludo = ['texto' => "Manolo"];
    $param = ['a' => 51, 'b' => 29];
    try {
        $cliente = new SoapClient(null, ['location' => $url, 'uri' => $uri, 'trace'=>true]);
    } catch (SoapFault $ex) {
        echo "Error: ".$ex->getMessage();
    }
    $saludo = $cliente->__soapCall('saludo', $paramSaludo);
    $suma = $cliente->__soapCall('suma', $param);
    $resta=$cliente->__soapCall('resta', $param);
    echo $saludo. " La suma es: $suma y la resta es: $resta";
    //También podríamos hacer
    echo "<br/>Alternativa:<br/>\n";
    $saludo = $cliente->saludo($paramSaludo['texto']);
    $suma = $cliente->suma($param['a'],$param['b']);
    $resta=$cliente->resta($param['a'],$param['b']);
    echo $saludo. " La suma es: $suma y la resta es: $resta";
?>
```

El servicio creado no incluye un documento WSDL (por eso al crear el servidor se ha puesto "null" como primer parámetro) para describir sus funciones. Sabes que existen los métodos suma, resta y saludo, y los parámetros que debes utilizar con ellos, porque conoces el código interno del servicio. Un usuario que no tuviera esta información, no sabría cómo consumir el servicio.

Utilizamos el método mágico "__soapCall()", básicamente se le pasa el nombre del método que se está llamando y los parámetros de dicho método en un array. Podíamos haber hecho directamente: "$cliente->suma(4, 5), $cliente->resta(4,5), $cliente->saludo("Nombre")".

Al igual que sucedía con SoapClient al programar un cliente, cuando utilizas SoapServer puedes crear un servicio sin documento asociado (como en el caso anterior, de nuevo "null" ha sido el primer parámetro), o indicar el documento correspondiente al servicio; pero antes deberás haberlo creado.

El primer parámetro del constructor indica la ubicación del WSDL correspondiente. El segundo parámetro es una colección de opciones de configuración del servicio. Si existe el primer

parámetro, ya no hace falta más información. PHP SOAP utiliza la información del documento para ejecutar el servicio. Si, como en el ejemplo, no existe , deberás indicar en el segundo parámetro al menos la opción "uri", con el espacio de nombres destino del servicio.

En ambos casos, y sobre todo cuando estamos en un entorno de desarrollo, es recomendable usar un bloque "try catch()" para capturar posibles excepciones.

El constructor "SoapServer" permite indicar, además de uri, otras opciones en el segundo parámetro. Por ejemplo, la opción soap_version indica si se va a usar **1.1** o **1.2.**

Para crear un documento de descripción del servicio, tendrás que seguir los pasos vistos anteriormente.

Al programar un servicio web, es importante cambiar en el fichero "php.ini" la directiva "soap.wsdl_cache_enabled" a **"0"**. En caso contrario, con su valor por defecto (**"1"**) los cambios que realices en los ficheros no tendrán efecto de forma inmediata.

El elemento raíz del documento será:

```
<definitions
    name="Operaciones"
    targetNamespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php"
    xmlns:tns="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
>
```

En este caso no se necesita definir ningún tipo nuevo, por lo que no habrá sección types. Los elementos message serán:

```
<message name="restaIn">
    <part name="a" type="xsd:float"/>
    <part name="b" type="xsd:float"/>
</message>
<message name="restaOut">
    <part name="return" type="xsd:float"/>
</message>
<message name="sumaIn">
    <part name="a" type="xsd:float"/>
    <part name="b" type="xsd:float"/>
</message>
<message name="sumaOut">
    <part name="return" type="xsd:float"/>
</message>
<message name="saludoIn">
    <part name="texto" type="xsd:string"/>
</message>
<message name="saludoOut">
    <part name="return" type="xsd:string"/>
</message>
```

El portType:

```
<portType name="OperacionesPort">
    <operation name="resta">
      <input message="tns:restaIn"/>
      <output message="tns:restaOut"/>
    </operation>
    <operation name="suma">
      <input message="tns:sumaIn"/>
      <output message="tns:sumaOut"/>
    </operation>
    <operation name="saludo">
      <input message="tns:saludoIn"/>
      <output message="tns:saludoOut"/>
    </operation>
</portType>
```

Suponiendo que el servicio web está en el fichero "servidor.php", dentro de las carpetas "ServicioWeb" y "Tema07/Ejemplos" la parte correspondiente al "binding" será:

```
<binding name="OperacionesBinding" type="tns:OperacionesPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="resta">
        <soap:operation soapAction="http://localhost/unidad6/servidorSoap/servidor.php#resta" />
        <input>
            <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost/unidad6/ServidorSoap/servidor.php" />
        </input>
        <output>
            <soap:body use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php" />
        </output>
    </operation>
    <operation name="suma">
        <soap:operation soapAction="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php#suma" />
        <input>
            <soap:body
            use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php" />
        </input>
        <output>
            <soap:body
            use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php" />
        </output>
    </operation>
    <operation name="saludo">
        <soap:operation soapAction="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php#saludo" />
        <input>
            <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php" />
        </input>
        <output>
            <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://php.localhost/Tema07/Ejemplos/ServicioWeb/servidor.php" />
        </output>
    </operation>
</binding>
```

Y para finalizar, el elemento service:

```
<service name="OperacionesService">
    <port name="OperacionesPort" binding="tns:OperacionesBinding">
      <soap:address location="http://localhost/unidad6/servidorSoap/servidor.php"/>
    </port>
</service>
```

Aunque el fichero parece extenso, se repite casi lo mismo para cada función. En el apartado siguiente veremos que lo podemos generar automáticamente con una librería.

El fichero completo quedaría:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
    name="Operaciones"
    targetNamespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php"
    xmlns:tns="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

    <message name="restaIn">
        <part name="a" type="xsd:float"/>
        <part name="b" type="xsd:float"/>
    </message>
    <message name="restaOut">
        <part name="return" type="xsd:float"/>
    </message>
    <message name="sumaIn">
        <part name="a" type="xsd:float"/>
        <part name="b" type="xsd:float"/>
    </message>
    <message name="sumaOut">
        <part name="return" type="xsd:float"/>
    </message>
```

```
    <message name="saludoIn">
        <part name="texto" type="xsd:string"/>
    </message>
    <message name="saludoOut">
        <part name="return" type="xsd:string"/>
    </message>
    <portType name="OperacionesPort">
        <operation name="resta">
            <input message="tns:restaIn"/>
            <output message="tns:restaOut"/>
        </operation>
        <operation name="suma">
            <input message="tns:sumaIn"/>
            <output message="tns:sumaOut"/>
        </operation>
        <operation name="saludo">
            <input message="tns:saludoIn"/>
            <output message="tns:saludoOut"/>
        </operation>
    </portType>
    <binding name="OperacionesBinding" type="tns:OperacionesPort">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="resta">
            <soap:operation soapAction="http://localhost/unidad6/servidorSoap/servidor.php#resta" />
            <input>
                <soap:body
                    use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://localhost/unidad6/ServidorSoap/servidor.php" />
            </input>
            <output>
                <soap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php" />
            </output>
        </operation>
        <operation name="suma">
            <soap:operation soapAction="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php#suma"
/>
            <input>
                <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php" />
            </input>
            <output>
                <soap:body
                use="encoded"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php" />
            </output>
        </operation>
        <operation name="saludo">
            <soap:operation
soapAction="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php#saludo" />
            <input>
                <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php" />
            </input>
            <output>
                <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php" />
            </output>
        </operation>
    </binding>
    <service name="OperacionesService">
        <port name="OperacionesPort" binding="tns:OperacionesBinding">
            <soap:address location="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor.php"/>
        </port>
    </service>
</definitions>
```

Es aconsejable definir una clase que implemente los métodos que queramos publicar en el servicio.

```php
class Operaciones {
    public function resta(float $a, float $b):float {
        return $a - $b;
    }
    public function suma(float $a,float $b):float {
        return $a + $b;
    }
    public function saludo(String $texto):string {
        return "Hola $texto";
    }
}
```

Y que al hacerlo de esta forma, en lugar de añadir una a una las funciones, podemos añadir la clase completa al servidor utilizando el método `setClass` de `SoapServer`. Lo ideal es que la clase con las funciones la implementemos en un archivo aparte y la llamemos con `"require"`.

```php
require '../src/Operaciones.php';
$uri='http://localhost/unidad6/servidorSoap';
$parametros=['uri'=>$uri];
try {
    $server = new SoapServer(NULL, $parametros);
    $server->setClass('Operaciones');
    $server->handle();
} catch (SoapFault $f) {
    die("error en server: " . $f->getMessage());
}
```

En lugar de una **clase**, también es posible indicar un **objeto** para procesar las peticiones utilizando el método `"setObject"` de la clase `"SoapServer"`.

`"PHP SOAP"` no genera el documento de forma automática para los servicios que se creen, existen algunos mecanismos que nos permiten generarlo, aunque siempre es aconsejable revisar los resultados obtenidos antes de publicarlos. Una de las formas más sencillas es utilizar la librería `php2wsdl`, disponible desde **Composer.**

Esta librería revisa los comentarios que hayas añadido al código de la clase que quieras a publicar (debe ser una clase, no funciones aisladas), y genera como salida el documento correspondiente.

Para que funcione correctamente, es necesario que los comentarios de las clases sigan un formato específico: el mismo que utiliza la herramienta de documentación `PHPDocumentor`.

`PHPDocumentor` es una herramienta de código libre para generación automática de documentación, similar a `Javadoc` (para el lenguaje **Java**). Si comentamos el código de nuestras aplicaciones siguiendo unas normas, `PHPDocumentor` es capaz de generar, a partir de los comentarios que introduzcamos en el código mientras programamos, documentación en diversos formatos. En el enlace dejado puedes consultar la documentación.

Aunque esta herramienta está en **Packagist**, para instalarla nos recomiendan bajarnos el fichero correspondiente. Si vas a usar `PHPDocumentor` asegúrate de tener activada e instalada la extensión "`php-iconv`" .

Para generar correctamente el documento con `php2wsdl` los métodos deben estar comentados siguiendo las reglas de `PHPDocumentor`. Los comentarios se deben ir introduciendo en el código distribuidos en bloques, y utilizando ciertas marcas específicas como `"@param"` para indicar un parámetro y `"@return"` para indicar el valor devuelto por una función, además para indicar los métodos que queremos que se generen en utilizar usaremos `"@soap"`.

```php
class Operaciones{
    /**
     * @soap
     * @param float $a
     * @param float $b
```

```
     * @return float
     */
    public function resta(float $a, float $b) :float{
        return $a - $b;
    }
    /**
     * @soap
     * @param float $a
     * @param float $b
     * @return float
     */
    public function suma($a, $b){
        return $a + $b;
    }
    /**
     * @soap
     * @param  string $texto
     * @return string
     */
    public function saludo($texto){
        return "Hola $texto";
    }
}
```

Para utilizar `Php2wsdl` lo integraremos a nuestro proyecto con `Composer`, ademas utilizaremos el `"autoload"` y espacios de nombre. Seguiremos la estructura de siempre, una carpeta `"src"` (`namespace` Clases) para las clases, la carpeta `"public"` donde estará `"consumir.php"` y la carpeta `"servidorSoap"` donde estará el archivo `"servidor.php"` .

Para integrarlo podemos hacerlo en el `"composer init"` buscando la dependencia o a posteriori con el comando: `"composer require php2wsdl/php2wsdl"`

Para generar el documento a partir de la clase `Operaciones` anterior, debes crear un nuevo fichero, por ejemplo hemos creado en la carpeta `"public"` el fichero `"generarwsdl.php"` con el siguiente código:

```
declare(strict_types=1);
require_once("./Operaciones.php");
require_once("../../../vendor/autoload.php");
use PHP2WSDL\PHPClass2WSDL;
$class = "Operaciones";
$serviceURI = "http://php.localhost//PHP/Tema07/Ejemplos/ServicioWeb/servidor.php";
$wsdlGenerator = new PHP2WSDL\PHPClass2WSDL($class, $serviceURI);
// Generate the WSDL from the class adding only the public methods that have @soap annotation.
$wsdlGenerator->generateWSDL(true);
// Dump as string
$wsdlXML = $wsdlGenerator->dump();
// Or save as file
$wsdlXML = $wsdlGenerator->save('./servidor2.wsdl');
```

Es decir, crear un nuevo objeto de la clase `PHPClass2WSDL`, e indicar como parámetros:

- El nombre de la clase que gestionará las peticiones al servicio.
- El `URI (serviceURI)` en que se ofrece el servicio.

A continuación llamamos al método `"generateWSDL()"` con `"true"` esto es para que solo convierta los métodos públicos con el parámetro `"@soap"`

El método `"save()"` obtiene como salida el documento WSDL de descripción del servicio. Le indicamos nombre y ruta donde guardarlo, en la ruta elegida el `Apache` tiene que tener permiso de escritura, si no, no podrá crearlo. O para evitar problemas de permisos puedes ejercutar en línea de comandos, por ejemplo: php generarwsdl.php

Revisa el documento creado, pues posiblemente tengas que realizar algunos cambios (por ejemplo, pasar el formato de codificación a UTF-8, o cambiar el nombre de alguna de las clases que contiene y de su constructor respectivo).

Cuando esté listo, publícalo con tu servicio. Para ello, copia el fichero obtenido en una ruta accesible vía web, e indica la en que se encuentra cuando instancies la clase `SoapServer`. Nos creamos el archivo `"servidor2.php"` para utilizar el archivo XSDL, que acabamos de generar.

```php
$server = new SoapServer("http://localhost/unidad6/servidorSoap/servicio.wsdl");
```

Si añades a la del servicio el parámetro `GET wsdl`, verás el fichero de descripción del servicio. En nuestro caso la URL sería: "http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor2.php". Se deja de ejemplo el código de como quedaría el “servidor2.php” y un archivo `"cliente2.php"` donde vemos algunos ejemplos de como consumir los servicios:

“servidor2.php”

```php
declare(strict_types=1);
if (isset($_GET['wsdl'])) {
    header('Content-Type: text/xml');
    echo file_get_contents("./servidor.wsdl");
    exit();
}
require_once("../../../vendor/autoload.php");
$uri='http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor2.wsdl';
//$parametros=['uri'=>$uri];
try {
    $server = new SoapServer($uri);
    $server->setClass('Operaciones');
    $server->handle();
} catch (SoapFault $f) {
    die("error en server: " . $f->getMessage());
}
```

“cliente2.php”

```php
declare(strict_types=1);
$url = 'http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor2.php';
$uri = 'http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidorSoap';
$paramSaludo = ['texto' => "Manolo"];
$param = ['a' => 51, 'b' => 29];
try {
    //$cliente = new SoapClient(null, ['location' => $url, 'uri' => $uri, 'trace'=>true]);
    $cliente = new SoapClient("${url}?wsdl");
} catch (SoapFault $ex) {
    echo "Error: ".$ex->getMessage();
}
$saludo = $cliente->__soapCall('saludo', $paramSaludo);
$suma = $cliente->__soapCall('suma', $param);
$resta=$cliente->__soapCall('resta', $param);
echo $saludo. " La suma es: $suma y la resta es: $resta";
//También podríamos hacer
echo "<br/>Alternativa:<br/>\n";
$saludo = $cliente->saludo($paramSaludo['texto']);
$suma = $cliente->suma($param['a'],$param['b']);
$resta=$cliente->resta($param['a'],$param['b']);
echo $saludo. " La suma es: $suma y la resta es: $resta";
```

En la variable "`$url`" del archivo `"cliente2.php"` podríamos haber puesto `$url="http://php.localhost/PHP/Tema07/Ejemplos/ServicioWeb/servidor2.wsdl`" y hubiese funcionado igual.