

Principios de Web Services

En este capítulo veremos para qué sirven los Web Services y sus principios describiendo los protocolos como XML, SOAP y WSDL. Por último haremos una breve explicación de la arquitectura orientada a servicios (SOA). Básicamente, veremos la teoría necesaria para poder entender los próximos capítulos.

¿Qué es un Web Service?	14
Protocolo HTTP	14
¿Para qué sirven los Web Services?	17
XML	18
WSDL	19
SOAP	21
Serialización	23
SOA	24
Resumen	25
Actividades	26

¿QUÉ ES UN WEB SERVICE?

Web Services es un estándar de comunicación entre procesos y o componentes, diseñado para ser multiplataforma y multilenguaje, es decir, no importa en qué lenguaje esté programado un **Web Service** como ser **Visual Basic**, **C#** o **java**, o en qué plataforma esté corriendo, ya sea **Windows**, **UNIX** o **Linux** éstos serán accesibles y utilizables por otras aplicaciones desarrolladas en otras plataformas o lenguajes de programación. Antiguamente se utilizaban otros estándares como **DCOM** (*Distributed Component Object Model*) introducido por Microsoft e implementado por otras plataformas, y **CORBA** (*Common Object Request Broker Architecture*) introducido por el **OMG** (*Object Management Group*) e implementado en distintas plataformas, incluido Windows. Estos estándares tenían bastantes problemas de configuración, especialmente en entornos en que se encontraban **firewalls** de por medio en los cuales era imposible (debido a estándares de seguridad de muchas compañías) habilitar ciertos puertos de comunicación para que estos componentes funcionaran. De esta manera la preferencia por utilizar el puerto 80 de **HTTP**, que normalmente se encuentra habilitado en la mayoría de los servidores y **firewalls** debido al uso de navegadores y servidores Web, no traería mayores complicaciones el uso de una tecnología que utilice este protocolo y puerto de **TCP/IP**.

La gran ventaja que trae el protocolo **HTTP** es su esquema de mensajes especialmente diseñado y optimizado para ser utilizado en redes como Internet, a diferencia de las viejas tecnologías como **DCOM** o **CORBA** que necesitaban un tipo de red más estable y local (**LAN**). Por ello es que el **HTTP** es el protocolo preferido para el transporte de mensajes de los **Web Services**.

Protocolo HTTP

El **HTTP** que significa **Hyper Text Transfer Protocol** se compone de 2 mensajes, el primer mensaje es originado por el cliente y es el requerimiento inicial de la comunicación. Este requerimiento llamado **Request** está dividido en una cabecera y un mensaje de texto. En la cabecera el cliente envía información de la pá-



ACCEDIENDO A UN SERVIDOR HTTP UTILIZANDO OTROS PUERTOS

El protocolo **HTTP** no necesariamente tiene que usar el puerto 80, si bien es cierto que éste es el más utilizado, podremos configurar el Servidor Web para que utilice este protocolo en otro puerto y desde un navegador poder acceder a él de la siguiente manera: **http://www.misitio.com:XXXX/** donde XXXX sería el número del puerto.

gina solicitada al servidor y de la aplicación cliente que estamos utilizando en el **User-Agent**, entre otras cosas, como así también los datos de la plataforma en la cual se encuentra corriendo la aplicación cliente.

```
GET http://www.yahoo.com/ HTTP/1.0
Accept: */*
Accept-Language: en-us
Cookie: B=34q6q5l2a8tai&b=3&s=fk; CP=v=60302&br=i&sp=; Q=q1=AAACAAAAAAAAAA--
&q2=RKWGkg--; F=a=TWiKbyosvdy7Bdpk7r4EIGH9hW5YsyP6L0dv9yv81YZ2Y7sh.irDi
lEgD7Sf&b=kLFy; C=mg=1; U=mt=&ux=dQcpeB&un=eaj38hfscrfjp; FPS=dl
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.1)
Host: www.yahoo.com
Proxy-Connection: Keep-Alive
```

En la primera línea vemos el método **HTTP** utilizado, en este ejemplo el **GET**, aunque puede ser también el método **POST**. La única diferencia es que los datos viajarán en el mensaje en vez de en la dirección **URL**. Al realizar un requerimiento **HTTP** usando el método **POST** vemos que los datos se envían a continuación del **header**. Una vez recibido este requerimiento el servidor devuelve una respuesta llamada **Response** la cual también está compuesta por una cabecera y un mensaje. A continuación, vemos la cabecera **HTTP** en respuesta a un requerimiento hecho con el navegador **Microsoft Internet Explorer** al sitio **www.yahoo.com**.

```
HTTP/1.0 200 OK
Date: Wed, 05 Jul 2006 15:25:44 GMT
Content-Type: text/html
Cache-Control: private
P3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV
TAI PSA PSD IVAI IVDi CONi TELo OTPi OUR DELi SAMi OTRI UNRi PUBi IND PHY ONL
UNI PUR FIN COM NAV INT DEM CNT STA POL HEA PRE GOV"
Vary: User-Agent
Set-Cookie: FPB=uq13c3ark12anmfo; expires=Thu, 01 Jun 2006 19:00:00 GMT;
path=/; domain=www.yahoo.com
Set-Cookie: D=ylh=X3oDMTf1dxBkcW1rBF9TAzI3MTYxNDkEcGlkAzExNTIxMTI3NzMEdGVz
dAMwBHRtcGwDaw5kZzXgtaWU-; path=/; domain=.yahoo.com
```

Seguido de la cabecera, el cliente recibirá el mensaje con el código **HTML** necesario para visualizar la página en el navegador.

En el siguiente ejemplo, el mensaje está reducido a unas pocas líneas ya que el mensaje de respuesta original es mucho más grande.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head>
<!-- SpaceID=2716149 loc=HR001 noad -->
<script language=javascript>
var now=new Date,t1=0,t2=0,t3=0,t4=0,t5=0,t6=0,cc='',y1p='';t1=now.getTime();
function err(a,b,c) {
var img=new Image;
```

Lo importante de la cabecera de respuesta está en la primera línea: donde dice **HTTP/1.0 200 OK** es el estado del mensaje, donde dice 200 significa el estado exitoso.

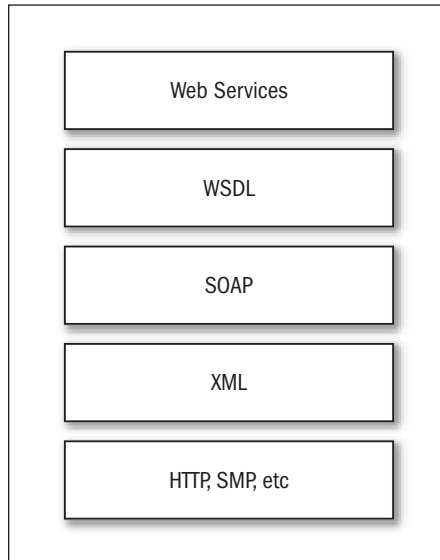


Figura 1. La figura nos muestra el orden jerárquico de las tecnologías y protocolos utilizados por los **Web Services**.



CÓDIGOS DE RESPUESTAS DE HTTP

Para obtener una lista con la referencia de lo que significan los códigos de los distintos tipos de respuestas del protocolo **HTTP** podemos dirigirnos al sitio web:

www.w3.org/Protocols/HTTP/HTRESP.html.

¿PARA QUÉ SIRVEN LOS WEB SERVICES?

El desarrollo y la programación de sistemas orientado a objetos o componentes nos ha llevado a lo largo del tiempo a tener la necesidad de reutilizarlos en diferentes proyectos. Ya sean componentes desarrollados por nosotros o componentes desarrollados por terceras partes. Hasta la existencia de los **Web Services** esta reutilización nos limitaba a un lenguaje de programación o a una plataforma en particular. Por lo tanto, el uso de los **Web Services** nos facilitará la reutilización de funciones de una aplicación en distintas plataformas o lenguajes ya sea para un uso personal en distintos proyectos, para comercializarlos o adquirir prestaciones de terceros.

De la misma forma que anteriormente incluíamos en nuestras aplicaciones referencias a otras librerías como ser Dlls o componentes **ActiveX**, ahora podremos referenciar funciones que se estarán ejecutando en otra computadora o servidor sin importarnos en qué están programados ni en que plataforma están corriendo.

Uno de los ejemplos más comunes del uso de los **Web Services** se encuentra en los sitios web de comercio electrónico, los cuales hacen uso de un **Web Service** para validar los datos de las tarjetas de crédito de sus clientes. Normalmente este **Web Service** es provisto por algún banco o entidad financiera que actúa como intermediario entre el comercio y las tarjetas de crédito. Otro ejemplo podría ser que necesitamos usar el corrector ortográfico del **Microsoft Word** desde un sitio web que creamos en **ASP.NET**. Ahora bien, esto es algo que podemos hacer a través de los **Visual Studio 2005 Tools for the Microsoft Office System**, pero para ello necesitaremos tener instalado el **Microsoft Word** en el servidor Web. Supongamos que por alguna razón no se nos permite instalar el **Microsoft Word** en el servidor Web, pero disponemos de un servidor de aplicaciones en el que tenemos control total y allí podemos instalar el **MS Word**. Para poder utilizar el **MS Word** que está instalado en otro servidor desde nuestra aplicación web podríamos crear un **Web Services** en el servidor de aplicaciones, el cual expone un Web Método público que se encargue de ejecutar el corrector ortográfico de **MS Word**. Teniendo esto podremos utilizar esta funcionalidad desde nuestra aplicación Web a través de un **Web Service** sin haber instalado **MS Word** en el servidor Web. Para poder reutilizar bien los componentes y objetos desarrollados era necesario un lenguaje de programación



PROCOLO SOAP AL DETALLE

Para obtener mayor información y detalles sobre la especificación del protocolo **SOAP** y de todos sus elementos podemos ir a www.w3.org/TR/soap. Allí encontraremos enlaces a sitios que contienen la documentación recomendada por W3C.

orientado a objetos. Ahí nace el lenguaje **c#** de la mano del **.NET framework**, que desde sus inicios se focalizó en proveer una herramienta como el Visual Studio, capaz de crear y consumir **Web Services** de la forma más rápida y sencilla tornando transparentes para el desarrollador protocolos y tipos de mensajes **XML** como **WSDL** y **SOAP**, los que describiremos a continuación.

XML

XML (*Extensible Markup Language*) es un lenguaje utilizado para definir formatos de documentos o mensajes. Éstos están compuestos por **Tags** (palabras entre caracteres `<` y `>`). Estos tipos de documentos han sido aceptados y adoptados por la mayoría de los fabricantes de software para proveer extensibilidad de los datos debido a que no está atado a ninguna plataforma o lenguaje de programación. Podremos ver que el **XML** es muy similar al **HTML**, pero sin embargo el **XML** es más estricto ya que por cada **tag** abierto deberemos tener un **Tag** que lo cierre. El **HTML**, sin embargo, no es tan estricto debido a que los navegadores de hoy en día, como el **Internet Explorer**, tienen la inteligencia de adivinar si un **tag** no fue cerrado correctamente.

De todos modos, si no cerramos un **tag** en un documento **XML** éste no podrá ser interpretado por los **parsers** (clases utilizadas para el manejo de documentos **XML**). Otra diferencia importante entre los documentos **XML** y **HTML** es que los **XML** son sensibles a las mayúsculas y minúsculas mientras los **HTML** no lo son, por ejemplo, un **Tag** `<cliente>` no será igual a un **tag** `<CLIENTE>`.

```
<?xml version='1.0'?>
<schema xmlns='http://www.w3.org/2001/XMLSchema'>
  <objeto>
    <propiedad atributo="valor">
  </propiedad>
</objeto>
```

Aparate de los **Tags** hay otras restricciones y validaciones que los documentos **XML** deberán cumplir, eso depende de un archivo de definiciones llamado **DTD** (*Document Type Definitions*) o de un archivo de esquemas de **XML**, los cuales podrán estar asociados con el documento **XML**. En estos archivos **DTDs** podremos definir si un elemento (también llamado nodo) deberá al menos aparecer una vez, una o más veces, una a ninguna vez. A su vez, cada elemento podrá tener uno o más atributos, eso depende también del archivo de definiciones o esquema. Estos documentos **XML** son la base de los documentos **WSDL** y **SOAP**, base de los **Web Services**.

WSDL

WSDL (*Web Services Description Language*) es un documento **XML** que se utiliza para describir los mensajes **SOAP** y cómo estos mensajes son intercambiados. Es decir, supongamos que creamos un **Web Services** y queremos que otras aplicaciones lo utilicen, las otras aplicaciones deberán acceder a un documento **WSDL** en donde podrán conocer los métodos que expone nuestro **Web Service** y cómo acceder a ellos, es decir, cuáles son los nombres de los métodos y qué tipos de parámetros espera cada uno de ellos. Todo documento **WSDL** está compuesto por un elemento raíz llamado **definitions** que a su vez está compuesto de los siguientes elementos:

- **types** define el tipo de esquema a ser utilizado, usualmente **XML**.
- **message** en este elemento se definen los mensajes de entrada y salida en forma abstracta entre el servidor y el cliente. Normalmente habrá varios elementos de este tipo para cada protocolo (**HTTP**, **SOAP**) y para cada **Web Method**.
- **portType** define los tipos de mensajes a intercambiar entre el cliente y el servidor, Éste puede ser de los tipos **Request-response**, en el cual por cada requerimiento se envía una respuesta, o **One-Way**, donde el Web service sólo recibe requerimientos pero no envía respuestas; **Solicit-Response**, la inversa del **Request-response** ya que el que solicita el requerimiento es el servidor en vez del cliente y por último **Notification** inverso al **One-Way**, donde el que manda el mensaje es sólo el servidor.

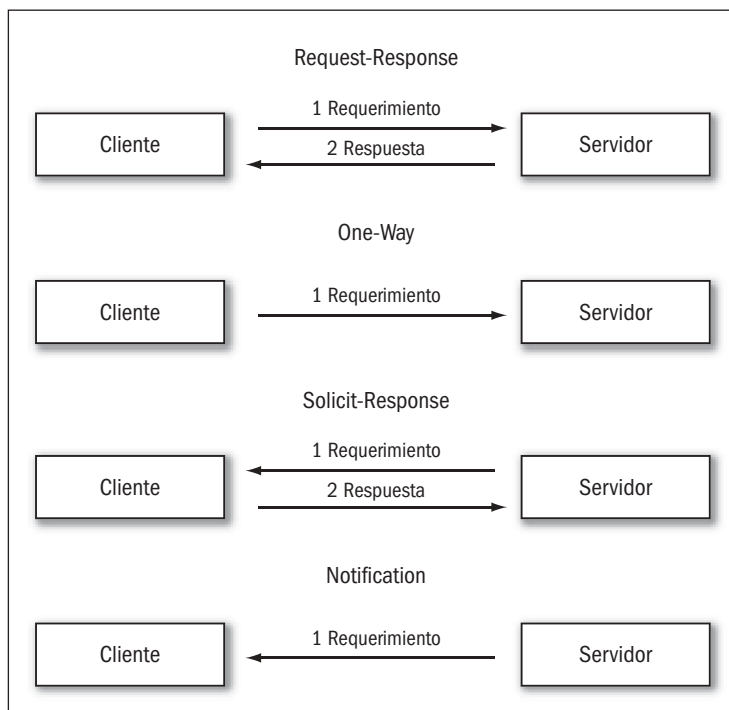


Figura 2. Vemos los distintos tipos de **portType** con sus órdenes y direcciones.

- **binding** en este elemento se establecen las definiciones de los vínculos de los protocolos como **SOAP** a un tipo de vínculo en particular. Por ejemplo, para el protocolo **SOAP** tendremos en atributo **soapAction** del elemento **soap:operation** la **URL** que tendremos que utilizar para invocar un **Web Method**.
- **service** en este elemento se informa el punto de acceso a los servicios para cada uno de los protocolos a través de un elemento **address**.

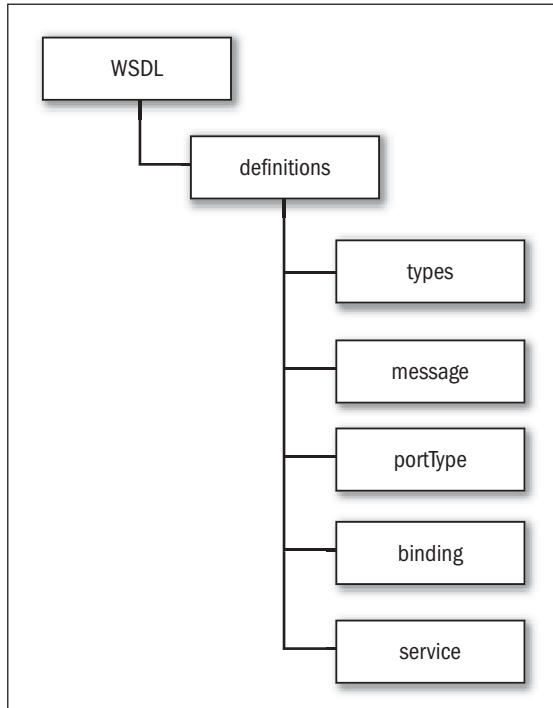


Figura 3. La figura nos muestra las relaciones de los distintos elementos que componen un documento **WSDL**.

Los documentos **WSDL** serán necesarios para que el **Visual Studio** o alguna otra herramienta de programación pueda generar una clase llamada **proxy**, que utilizaremos desde nuestra aplicación cliente sin necesidad de entender o pasar



WSDL AL DETALLE

Para los que quieran entrar en detalle sobre los distintos elementos de la especificación de **WSDL** pueden dirigirse al siguiente link: www.w3.org/TR/wsdl. La información es muy completa, y nos permitirá comprender en profundidad este lenguaje.

este tipo de documentos. Por otro lado, cuando creemos un proyecto del tipo **Web Service** veremos cómo el **Visual Studio** ya crea estos documentos **WSDL** automáticamente, con lo cual nos facilitará bastante la tarea de crearlos.

SOAP

SOAP (*Simple Object Access Protocol*) es el protocolo base de los **Web Services**. Este protocolo está basado en **XML** y no se encuentra atado a ninguna plataforma o lenguaje de programación. A su vez, también es el protocolo más aceptado por la mayoría de las plataformas.

Si bien **SOAP** es un protocolo, éste no es exactamente un protocolo de comunicación entre mensajes como lo es el **HTTP**, por ejemplo. Básicamente, **SOAP** son documentos **XML** y necesitaremos utilizar algún otro protocolo para la transmisión de estos documentos como ser el protocolo **HTTP** o cualquier otro protocolo de comunicación capaz de transmitir textos. Los mensajes **SOAP** están compuestos por un tag principal llamado **Envelope**, que está dividido en una cabecera o **Header** y en un cuerpo o **Body**.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <miCabecera>miValor</miCabecera>
  </soap:Header>
  <soap:Body>
    <MiMetodo>
      <MiParametro>miValor</MiParametro>
    </MiMetodo>
  </soap:Body>
</soap:Envelope>
```

III WSDL.EXE

Esta es una herramienta que es parte del **.NET framework SDK**. Por medio de línea de comando podremos generar las clases **proxies** necesarias para consumir un **Web Service**.

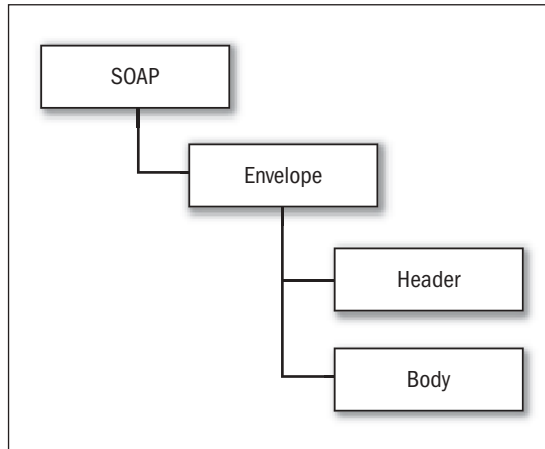


Figura 4. La figura muestra las relaciones de los diferentes elementos de un documento **SOAP**.

Dentro del elemento **Body** estarán los elementos correspondientes al **Web Method** que queramos invocar y además podrá haber o no un elemento en común llamado **Fault**, que nos indicará que ha ocurrido un error y la razón de éste.

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>Client.Security</soap:faultcode>
      <soap:faultstring>Acceso denegado.</soap:faultstring>
      <soap:faultactor>http://miwebservice.com</soap:faultactor>
      <soap:detail>
        <miError>no se pudo acceder al archivo prueba.txt</miError>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
  
```



MICROSOFT Y SOA

Aquellos que quieren profundizar sus conocimientos sobre la arquitectura **SOA** y sobre cómo Microsoft recomienda utilizarla con sus productos como Visual Studio, pueden visitar la página <http://msdn.microsoft.com/architecture/soa>.

Serialización

Serialización es el proceso de convertir un dato binario a una representación de texto usando caracteres del tipo **ASCII**, por ejemplo. Debido a que **SOAP** está basado en **XML**, éste a su vez no soporta el envío de cualquier carácter ya que algunos caracteres son utilizados para control de los mensajes, como por ejemplo el carácter cero sería utilizado para la finalización de una cadena de caracteres (**string**). Si lo que queremos enviar como parámetro es un objeto complejo o un objeto que contiene una imagen, entonces los datos que por su naturaleza se encuentran en binario deberán ser transformados a una cadena de caracteres. Esta técnica es conocida como **Encode**, hay varias formas realizar una **serialización** y esta forma estará determinada por el atributo **encoding** que se especifica en la cabecera **XML** de un mensaje **SOAP**. Si vemos en el ejemplo mencionado anteriormente observaremos que éste utiliza el **uft-8**.

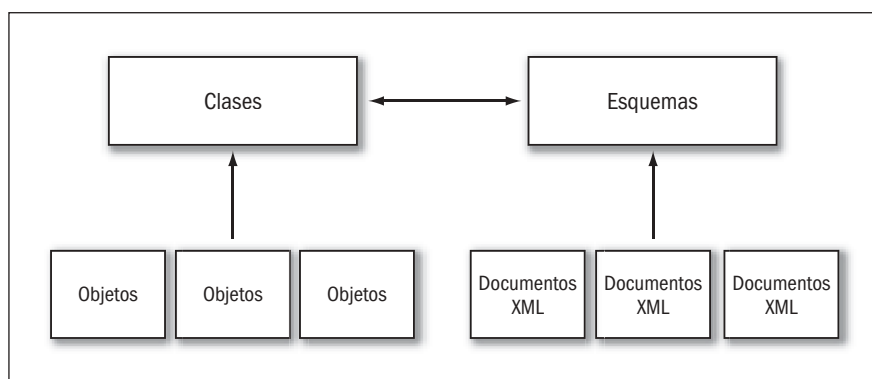


Figura 5. Equivalencia de las clases y objetos luego de ser serializados donde las clases se convierten en esquemas y los objetos en documentos **XML** y viceversa.

La misma clase **Proxy** que nos es generada por el **visual Studio** para ser utilizada en la aplicación cliente que consumirá nuestro **Web Service** es la que se encarga de serializar los objetos antes de ser transmitidos por el protocolo **HTTP** hacia el **Web Method**, como también de deserializar la respuesta. La **figura 6** nos muestra todos los pasos del ciclo de vida de un **Web Service**.

▶ MAS INFORMACIÓN SOBRE WEB SERVICES

Para obtener mayor información y detalles sobre los últimos avances de **Microsoft** con respecto a los **Web Services** podemos ir a <http://msdn.microsoft.com/webservices>. En este sitio encontraremos muchísimas herramientas para desarrolladores.

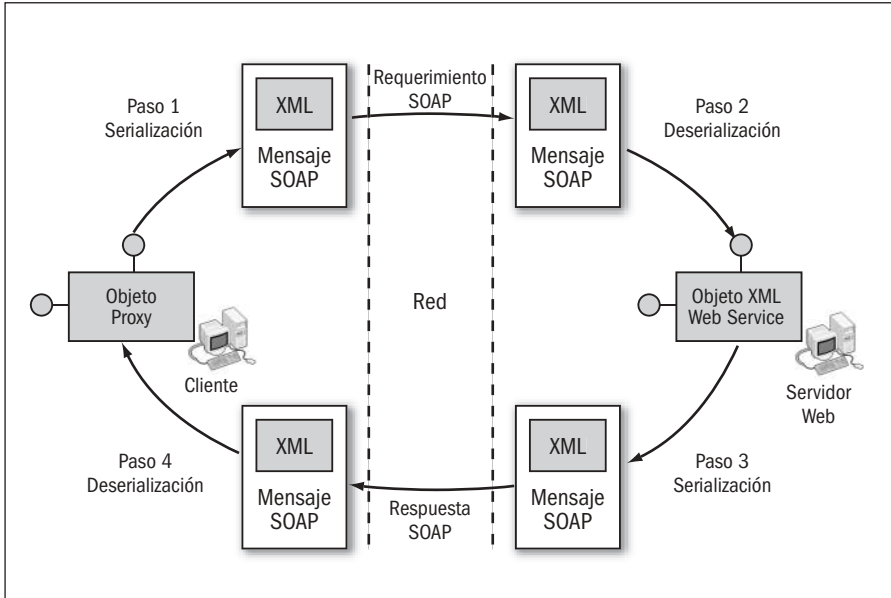


Figura 6. Ciclo de vida de un **Web Service** en el que se serializa y se deserializa en cada uno de los pasos.

SOA

SOA (*Service Oriented Architecture*) es, sin dudas, la arquitectura de desarrollo de sistemas más de moda de estos tiempos. Básicamente se trata de un cambio de mentalidad a la hora de desarrollar un sistema y pensar cada uno de sus módulos como un servicio. Esto nos facilitará el uso de los **Web Services**. Entonces tendremos un conjunto de servicios que se comunicarán unos con otros intercambiando distintos tipos de datos e involucrando más de un servicio, eso depende de la operación que necesitemos realizar. Es necesario destacar que **SOA** no es un producto o una tecnología sino más bien una moderna forma de diseñar nuestras aplicaciones o sistemas. Para llevar a cabo esta forma de diseño encontraremos una gran variedad de productos y tecnologías disponibles, como ser la tecnología de **Web Services** y el producto Visual Studio (que nos ayudará a llevar a cabo soluciones basadas en la arquitectura **SOA**). Si miramos otras compañías que son competencia de **Microsoft**, como **Sun** o **IBM**, veremos que ellas también están realizando una inversión muy interesante en dar a conocer esta arquitectura y ayudar a implementarla cada uno a través de sus productos. Basta sólo con entrar a msdn.microsoft.com o developers.sun.com/channel/ y realizar una búsqueda por *Service Oriented Architecture* y obtendremos un montón de resultados sobre los cuales podría escribirse más de un libro. Los **Web Services** son, si bien no los

únicos, los recursos más utilizados en esta arquitectura. Por eso, aprender a desarrollarlos y entenderlos nos ayudará mucho a pensar en servicios. **SOA** tiene 4 principios básicos que detallaremos a continuación:

- **Límites:** los servicios están demarcados por límites específicos y la única forma de comunicarse con ellos será a través de la tecnología o protocolos expuestos por ellos.
- **Autonomía:** cada servicio deberá comportarse de forma autónoma. Por ejemplo, esto quiere decir que si mi sistema hace uso de un servicio que consulta una base de datos, mi sistema no tendrá por qué saber ni conocer a qué tipo de base de datos se está conectando este servicio ni de qué manera lo hace. Así obtendremos un diseño totalmente desacoplado.
- **Contratos:** aquí se definen cómo serán utilizados los servicios y de qué manera intercambiarán los datos y mensajes.
- **Políticas:** cada servicio deberá definir las políticas de su uso, como por ejemplo, que se deberá utilizar el protocolo **HTTP** o que se requerirá el uso de transacciones para ser utilizado.

RESUMEN

Al comenzar hemos visto la teoría necesaria para entender cómo funcionan los Web Services, para qué se los utiliza. Repasamos el protocolo HTTP, que es el más utilizado por los Web Services y, por otro lado, vimos los tipos de documentos WSDL y SOAP usados por los Web Services y sus protocolos. Dimos una breve introducción a la serialización de objetos y por último dimos un vistazo a la arquitectura orientada a servicios y sus fundamentos.



TEST DE AUTOEVALUACIÓN

1 ¿Para qué sirven los Web Services?

2 ¿Cómo está compuesto un mensaje del protocolo HTTP?

3 ¿Cuáles son los usos más comunes de los Web Services?

4 ¿En qué se diferencian los documentos XML de los HTML?

5 ¿Qué es un archivo DTD?

6 ¿Cómo está compuesto un documento WSDL?

7 ¿Qué significa que un tipo de mensaje es One-Way?

8 ¿Qué es el protocolo SOAP?

9 ¿Para qué se utiliza la serialización?

10 ¿Cuáles son los principios de SOA?
