www.bi-dss.com

CAPACITACIÓN

# DESARROLLO Y CONSUMO DE WEBSERVICES

Instructor:
Ing. José Fernando Ramos

info@bi-dss.com
Tel. 2203-4770

## CONTENIDO

# INTRODUCCIÓN

El presente documento tiene como finalidad presentar información, conceptos, generalidades y otros aspectos relacionados al curso de desarrollo y consumo de WebServices con metodologías SOAP y REST en un entorno de trabajo con Microsoft Visual Studio 2013 y versiones superiores.

# OBJETIVOS

## GENERAL

Dar a conocer al participante los conceptos, características y mejores prácticas para el desarrollo y consumo de servicios web (WebServices) para cualquier proceso de integración con otras plataformas que utilicen este tipo de arquitectura.

## ESPECIFICOS

- Conocer los conceptos y fundamentos de los WebServices.
- Conocer la estructura, modelos y componentes alrededor del entorno de trabajo con WebServices.
- Utilizar el conocimiento adquirido en el curso con el fin de desarrollar un proyecto de desarrollo y consumo de WebServices.

# GENERALIDADES Y FUNDAMENTOS

## ¿Qué son los WebServices?

Un web service es una vía de intercomunicación e interoperabilidad entre máquinas conectadas en Red. En el mundo de Internet se han popularizado enormemente, ya se trate de web services públicos o privados. Generalmente, la interacción se basa en el envío de solicitudes y respuestas entre un cliente y un servidor, que incluyen datos. El cliente solicita información, enviando a veces datos al servidor para que pueda procesar su solicitud. El servidor genera una respuesta que envía de vuelta al cliente, adjuntando otra serie de datos que forman parte de esa respuesta.

Por tanto, podemos entender un servicio web es como un tráfico de mensajes entre dos máquinas, que permite interactuar con el mismo mediante el intercambio de mensajes *SOAP*, típicamente transmitidos usando serialización *XML* sobre *HTTP* conjuntamente con otros estándares web.


# ESTÁNDARES EMPLEADOS

## *XML*

Abreviación de Extensible Markup Language. El XML es una especificación desarrollada por W3C, permite a los desarrolladores crear sus propios tags, que les permiten habilitar definiciones, transmisiones, validaciones, e interpretación de los datos entre aplicaciones y entre organizaciones.

## *SOAP*

Abreviación de Simple Object Access Protocol, es un protocolo de mensajería construido en XML que se usa para codificar información de los requerimientos de los Web Services y para responder los mensajes “antes” de enviarlos por la red. Los mensajes SOAP son

independientes de los sistemas operativos y pueden ser transportados por los protocolos que funcionan en la Internet, como ser: SMTP, MIME y HTTP.

### *WSDL*

Abreviación de Web Services Description Language, es un lenguaje especificado en XML que se ocupa para definir los Web Service como colecciones de punto de comunicación capaces de intercambiar mensajes, en otras palabras, es un estándar de uso público (no se requiere pagar licencias ni royalties para usarlo).

### *REST*

Abreviación de Representational State Transfer, arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, POST, PUT, otros) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

## Beneficios de Utilizar WebServices (WS)

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen. *El servicio web usa HTTP y los puertos estándares (puerto 80 u otro) para comunicarse, lo que facilita el acceso entre los sistemas.*
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Cada sistema opera como una caja negra donde se intercambian paquetes de información, sin preocuparse por la complejidad inherente de los receptores.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados. *Por ejemplo, una aplicación Windows programada con Visual Studio y C# puede comunicarse con otra instalada en Linux + PHP para intercambiar información.*
- Permite la intercomunicación sin importar la aplicación y lenguaje de programación empleada.

Estas ventajas permiten que diferentes grupos de desarrollo, con diferentes expertise, conocimientos y experiencias en diferentes plataformas puedan colaborar a una solución común, simplemente poniéndose de acuerdo cual es la información que necesitan para interactuar. De esta manera se llegan a soluciones desacopladas, es decir, sistemas que

sean independientes al mantenerse y desarrollarse, pero que aporten hacia una solución común.

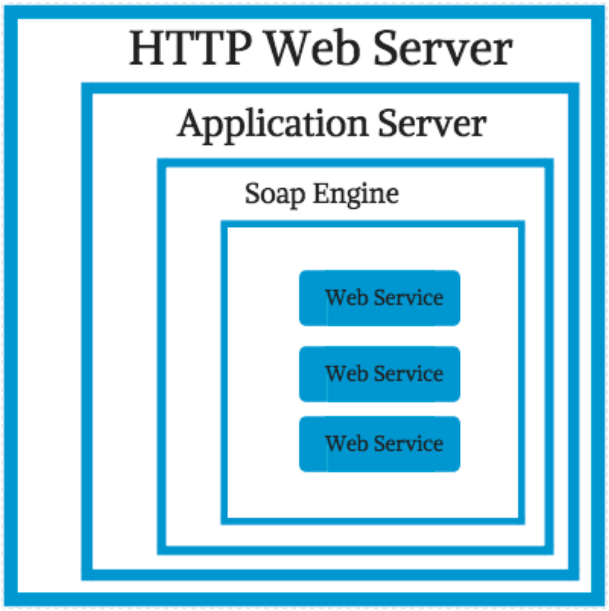
### Arquitectura

- Service Discovery. Responsable de centralizar servicios web en un directorio común de registro y proveer una funcionalidad sencilla para publicar y buscar.
- Service Description. Uno de los aspectos más característicos de los webs services es que se autodescriben. Esto significa que una vez que se ha localizado un Web Service nos proporcionará información sobre que operaciones soporta y cómo activarlo. Esto se realiza a través del Web Services Description Language (WSDL).
- Service Invocation. Invocar a un Web Service implica pasar mensajes entre el cliente y el servidor. SOAP (Simple Object Access Protocol) especifica cómo deberíamos formatear los mensajes request para el servidor, y cómo el servidor debería formatear sus mensajes de respuesta.
- Transport. Todos estos mensajes han de ser transmitidos de alguna forma entre el servidor y el cliente. El protocolo elegido para ello es HTTP (HyperText Transfer Protocol). Se pueden utilizar otros protocolos, pero HTTP es actualmente el más usado.

## ¿CÓMO FUNCIONA UN WEBSERVICE?

1. El Service Provider genera el WSDL describiendo el Web Service y registra el WSDL en el directorio UDDI o Service Registry.
2. El Service Request o la aplicación del cliente requiere un Web Service y se pone en contacto con el UDDI para localizar el Web Service.
3. El cliente, basándose en la descripción descrita por el WSDL, envía un request para un servicio particular al Web Service Listener, que se encarga de recibir y enviar los mensajes en formato SOAP.
4. El Web Service analiza el mensaje SOAP del request e invoca una operación particular en la aplicación para procesar el request. El resultado se escribe de nuevo en SOAP en forma de respuesta y se envía al cliente.
5. El cliente analiza el mensaje de respuesta SOAP y lo interpreta o genera un error si ha habido alguno.

## COMPONENTES DEL WEBSERVICE




- Web Service. Es el software o componente que realiza las operaciones. Este puede estar escrito en cualquier lenguaje: C#, php, JAVA, otros.
- SOAP Engine. El Web Service no sabe interpretar SOAP requests y crear SOAP responses. Para hacer esto hace falta un SOAP engine, un software que se encarga del manejo de estos mensajes. Apache Axis es un ejemplo.
- Application Server. Para funcionar como un servidor que puede recibir requests desde diferentes clientes, el SOAP engine normalmente funciona dentro de un application server. Este es otro software que proporciona un espacio libre para aplicaciones que han de ser accedidas por múltiples clientes. El SOAP engine funciona como una aplicación dentro del application server. Ejemplos son Apache Tomcat server, IIS Server, otros.
- HTTP Web Server. Algunos application servers incluyen funcionalidades HTTP, por lo que se pueden tener Web Services funcionando, instalando simplemente un SOAP engine y un application server. Sin embargo, cuando un application server carece de funcionalidad HTTP es necesario también un HTTP server, más comúnmente llamado Web Server. Es un software que sabe cómo manejar mensajes HTTP.

## ESTRUCTURA DE UN WEBSERVICE


### ¿Qué es XML?

**XML, es el estándar de *Extensible Markup Language*. XML no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.**

**En primer lugar, para entenderlo bien hay que olvidarse un poco, sólo un poco de HTML. En teoría HTML es un subconjunto de XML especializado en presentación de documentos para la Web, mientras que XML es un subconjunto de SGML especializado en la gestión de información para la Web. En la práctica XML contiene a HTML aunque no en su totalidad. La definición de HTML contenido totalmente dentro de XML y por lo tanto que cumple a rajatabla la especificación SGML es XHTML (Extensible, Hypertext Markup Language.**

## PRINCIPALES CARACTERÍSTICAS

- Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos.
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML.
- Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.

- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente web.
- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.
- Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.

## PARTES DE UN DOCUMENTO XML

### *Prólogo*

**Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.**

**El prólogo de un documento XML contiene:**

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

**Ejemplo:** <?xml version="1.0" encoding="UTF-8"?>

### *Cuerpo*

**A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Sin embargo, es necesaria la adquisición de datos para su buen funcionamiento.**

**Ejemplo:**

```
<Edit_Mensaje>
   (...)
</Edit_Mensaje>
```

### *Elementos*

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

### *Atributos*

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

Por ejemplo, un elemento «estudiante» puede tener un atributo «Mario» y un atributo «tipo», con valores «come croquetas» y «taleno» respectivamente.

```
<Estudiante Mario="come croquetas" tipo="taleno">Esto es un día que Mario va paseando...</Estudiante>
```

## ESTRUCTURA

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez por notas. Estas partes se llaman elementos, y se las señala mediante *etiquetas*.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma `<nombre>`, donde nombre es el nombre del elemento que se está señalando.

Ejemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
     <Mensaje>
          <Remitente>
               <Nombre>Nombre del remitente</Nombre>
               <Mail> Correo del remitente </Mail>
          </Remitente>
          <Destinatario>
               <Nombre>Nombre del destinatario</Nombre>
               <Mail>Correo del destinatario</Mail>
          </Destinatario>
          <Texto>
               <Asunto>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Asunto>
               <Parrafo>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Parrafo>
          </Texto>
     </Mensaje>
</Edit_Mensaje>
```

## XML EN LOS SERVICIOS WEB

- Es un estándar abierto es decir que es reconocido mundialmente ya que muchas compañías tecnológicas integran en sus softwares compatibilidad con dicho lenguaje. Esto quiere decir que la gran mayoría de software de escritorio de sistema operativo, aplicaciones móviles permiten la compatibilidad con XML esto lo hace muy potente a la hora de permite la comunicación entre distintas plataformas de software y hardware (y si bien recordamos este es el sentido final de los Servicios Web).
- Simplicidad de sintaxis esto quiere decir que es muy fácil de escribir código en XML y la representación de los datos es casi entendible por cualquier ser humano. Esto lo hace muy flexible a la hora de querer representar datos de cualquier especie, bastara con contar con cualquier editor de texto y aprende unas cuantas intrusiones básicas y ya está en condiciones de escribir código XML el cual será soportado o entendido por cualquier aplicación que pueda leer documentos XML.

El hecho de que XML sea tan fácil de codificar y de entender lo hace el lenguaje ideal para utilizarlo en los servicios Web.

- Independencia del protocolo de Transporte, el hecho de que XML es un lenguaje de Marcado de Texto, no necesita de ningún protocolo de trasporte especial, solo necesita de un protocolo que pueda trasferir texto o documentos simples. Esto nos trae a la memoria que en mercado existen muchos protocolos con estas características como lo son los más conocidos en HTTP y SMTP por nombrar algunos. Volviendo el tema de los servicios Web una de las características de estos es la independencia del protocolo de trasporte.

## ¿Qué es WSDL?

**WSDL (*Web Services Description Language*) es un protocolo basado en XML que describe los accesos al Web Service. Podríamos decir que es el manual de operación de este, porque nos indica cuáles son las interfaces que provee el Servicio web y los tipos de datos necesarios para su utilización.**

**WSDL es el lenguaje propuesto por el W3C para la descripción de Servicios Web y permite describir la interfaz de un servicio web en un formato XML. Una de sus ventajas es que permite separar la descripción abstracta de la funcionalidad ofrecida por un servicio, es decir, de los detalles concretos del mismo, como puede ser el enlace a un protocolo de red o un formato de mensaje concreto que puede ser SOAP, HTTP o MIME.**

**El WSDL describe los servicios Web a través de los mensajes que se intercambian entre el proveedor del servicio y el cliente.**

### ELEMENTOS DEL WSDL

| Nombre | Descripción |
|---|---|
| <definitions> | Comienzo del documento, este tag agrupa a todos los demás elementos. |
| <types> | Se definen los tipos de datos utilizados en los mensajes. |
| <message> | Se definen los métodos y parámetros para realizar la operación. Cada message puede consistir en una o más partes (parámetros). Las partes pueden ser de cualquiera de los tipos definidos en la sección anterior. |
| <portType> | Esta sección es la más importante, ya que definen las operaciones que pueden ser realizadas, y los mensajes que involucran (por ejemplo, el mensaje de petición y el de respuesta). |

| | |
|---|---|
| <binding> | Se definen el formato del mensaje y detalles del protocolo para cada portType. |
| <service> | Colección de puntos finales donde se encuentra alojado el servicio. |

Ejemplo

```
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
         <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
         </element>
         <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
         </element>
      </schema>
   </types>

   <message name="GetLastTradePriceInput">
       <part name="body" element="xsd1:TradePriceRequest"/>
   </message>

   <message name="GetLastTradePriceOutput">
       <part name="body" element="xsd1:TradePrice"/>
   </message>

   <portType name="StockQuotePortType">
       <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceInput"/>
          <output message="tns:GetLastTradePriceOutput"/>
       </operation>
   </portType>

   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
       <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
       <operation name="GetLastTradePrice">
          <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
          <input>
              <soap:body use="literal"/>
          </input>
          <output>
              <soap:body use="literal"/>
          </output>
       </operation>
   </binding>

   <service name="StockQuoteService">
       <documentation>My first service</documentation>
       <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
          <soap:address location="http://example.com/stockquote"/>
       </port>
   </service>

</definitions>
```

¿CÓMO LO OFRECE?

¿QUÉ OFRECE?

¿CÓMO LO OFRECE?

¿DÓNDE LO OFRECE?

Ejemplo

```
−<definitions name="wsCobranza" targetNamespace="CFSeguros">
 −<types>
  +<schema targetNamespace="CFSOTASE" elementFormDefault="qualified"></schema>
  −<schema targetNamespace="CFSeguros" elementFormDefault="qualified">
     <import namespace="CFSOTASE"/>
    −<element name="wsCobranza.CONSULTA">
     −<complexType>
      −<sequence>
          <element minOccurs="1" maxOccurs="1" name="Banco" type="xsd:string"/>
          <element minOccurs="1" maxOccurs="1" name="Canal" type="xsd:short"/>
          <element minOccurs="1" maxOccurs="1" name="Usuario" type="xsd:string"/>
          <element minOccurs="1" maxOccurs="1" name="Password" type="xsd:string"/>
          <element minOccurs="1" maxOccurs="1" name="Numeroservicio" type="xsd:long"/>
          <element minOccurs="1" maxOccurs="1" name="Campo1" type="xsd:string"/>
          <element minOccurs="1" maxOccurs="1" name="Campo2" type="xsd:string"/>
          <element minOccurs="1" maxOccurs="1" name="Campo3" type="xsd:string"/>
       </sequence>
      </complexType>
    </element>
    −<element name="wsCobranza.CONSULTAResponse">
     −<complexType>
      −<sequence>
          <element minOccurs="1" maxOccurs="1" name="Datosservicio" type="tns1:SDTDatosServicio"/>
       </sequence>
      </complexType>
    </element>
    +<element name="wsCobranza.PAGO"></element>
    +<element name="wsCobranza.PAGOResponse"></element>
    +<element name="wsCobranza.REVERSA"></element>
    +<element name="wsCobranza.REVERSAResponse"></element>
    +<element name="wsCobranza.RETORNASERVICIOS"></element>
    +<element name="wsCobranza.RETORNASERVICIOSResponse"></element>
   </schema>
 </types>
 +<message name="wsCobranza.CONSULTASoapIn"></message>
 +<message name="wsCobranza.CONSULTASoapOut"></message>
 +<message name="wsCobranza.PAGOSoapIn"></message>
 +<message name="wsCobranza.PAGOSoapOut"></message>
 +<message name="wsCobranza.REVERSASoapIn"></message>
 +<message name="wsCobranza.REVERSASoapOut"></message>
 +<message name="wsCobranza.RETORNASERVICIOSSoapIn"></message>
 +<message name="wsCobranza.RETORNASERVICIOSSoapOut"></message>
 −<portType name="wsCobranzaSoapPort">
  −<operation name="CONSULTA">
     <input message="wsdlns:wsCobranza.CONSULTASoapIn"/>
     <output message="wsdlns:wsCobranza.CONSULTASoapOut"/>
   </operation>
  +<operation name="PAGO"></operation>
  +<operation name="REVERSA"></operation>
  +<operation name="RETORNASERVICIOS"></operation>
 </portType>
 −<binding name="wsCobranzaSoapBinding" type="wsdlns:wsCobranzaSoapPort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  +<operation name="CONSULTA"></operation>
  +<operation name="PAGO"></operation>
  +<operation name="REVERSA"></operation>
  +<operation name="RETORNASERVICIOS"></operation>
 </binding>
 −<service name="wsCobranza">
  −<port name="wsCobranzaSoapPort" binding="wsdlns:wsCobranzaSoapBinding">
     <soap:address location="https://localhost/CFSeguros/awscobranza.aspx"/>
   </port>
 </service>
</definitions>
```

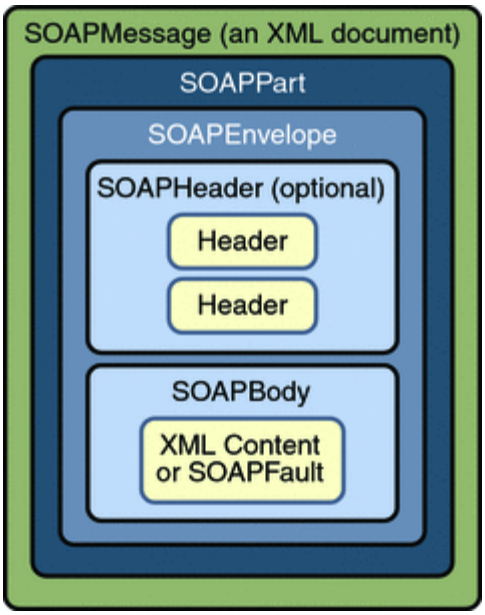
## ¿Qué es SOAP?

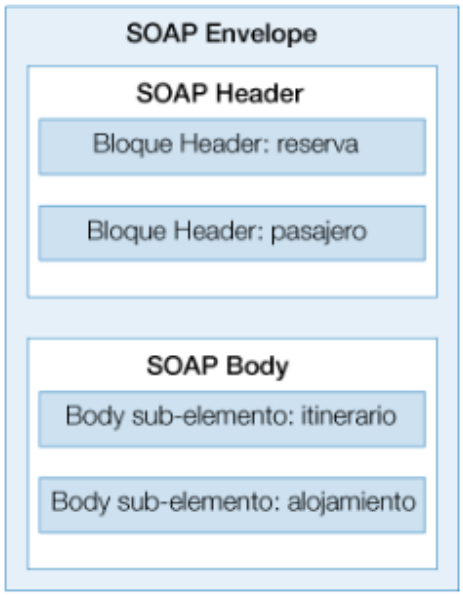
***Simple Object Access Protocol*** **es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por Dave Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros. Está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.**

## VENTAJAS

- No está asociado con ningún lenguaje: los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista, pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java y plataformas como .NET.
- No se encuentra fuertemente asociado a ningún protocolo de transporte: La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- No está atado a ninguna infraestructura de objeto distribuido La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.
- Aprovecha los estándares existentes en la industria: Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.
- Permite la interoperabilidad entre múltiples entornos: SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dichos estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas.

## ANATOMÍA DE UN MENSAJE SOAP




Ejemplo

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reserva xmlns:m="[[http://www.example.org]]"
               env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
               env:mustUnderstand="true">
      <m:referencia>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:referencia>
      <m:fechaYHora>2001-11-29T13:20:00.000-05:00</m:fechaYHora>
    </m:reserva>
    <n:pasajero xmlns:n="http://miempresa.example.com/empleados"
                env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
                env:mustUnderstand="true">
      <n:nombre>Åke Jógvan Øyvind</n:nombre>
    </n:pasajero >
  </env:Header>
  <env:Body>
    <p:itinerario xmlns:p="http://empresaviajes.example.org/reserva/viaje">
      <p:ida>
        <p:salida>Nueva York</p:salida>
        <p:llegada>Los Angeles</p:llegada>
        <p:fechaSalida>2001-12-14</p:fechaSalida>
        <p:horaSalida>última hora de la tarde</p:horaSalida>
        <p:preferenciaAsiento>pasillo</p:preferenciaAsiento>
      </p:ida>
      <p:vuelta>
        <p:salida>Los Angeles</p:salida>
        <p:llegada>Nueva York</p:llegada>
        <p:fechaSalida>2001-12-20</p:fechaSalida>
        <p:horaSalida>media-mañana</p:horaSalida>
        <p:preferenciaAsiento />
      </p:vuelta>
    </p:itinerario>
    <q:alojamiento xmlns:q="http://empresaviajes.example.org/reserva/hoteles">
      <q:preferencia>ninguna</q:preferencia>
    </q:alojamiento>
  </env:Body>
</env:Envelope>
```

## MODELO DE INTERCAMBIO

- Los mensajes SOAP son transmisiones unidireccionales desde un emisor a un receptor.
- Se suelen combinar mensajes para implementar patrones, como petición/respuesta.
- Las implementaciones SOAP se pueden optimizar para explotar las características específicas de sistemas de red concretos.

Ejemplo: Petición

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productId>827635</productId>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

Ejemplo: Repuesta

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Toptimate 3-Piece Set</productName>
         <productId>827635</productId>
         <description>3-Piece luggage set.  Black Polyester.</descriptio
         <price>96.50</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```

## ¿Qué es REST?

***Representational State Transfer*** **o traducido a "Transferencia de presentación de estado" es lo que se domina a REST. ¿Y eso es?, una técnica de arquitectura de software usada para construir APIs que permitan comunicar a nuestro servidor con sus clientes usando el protocolo HTTP mediante URIs lo suficientemente inteligentes para poder satisfacer la necesidad del cliente.**

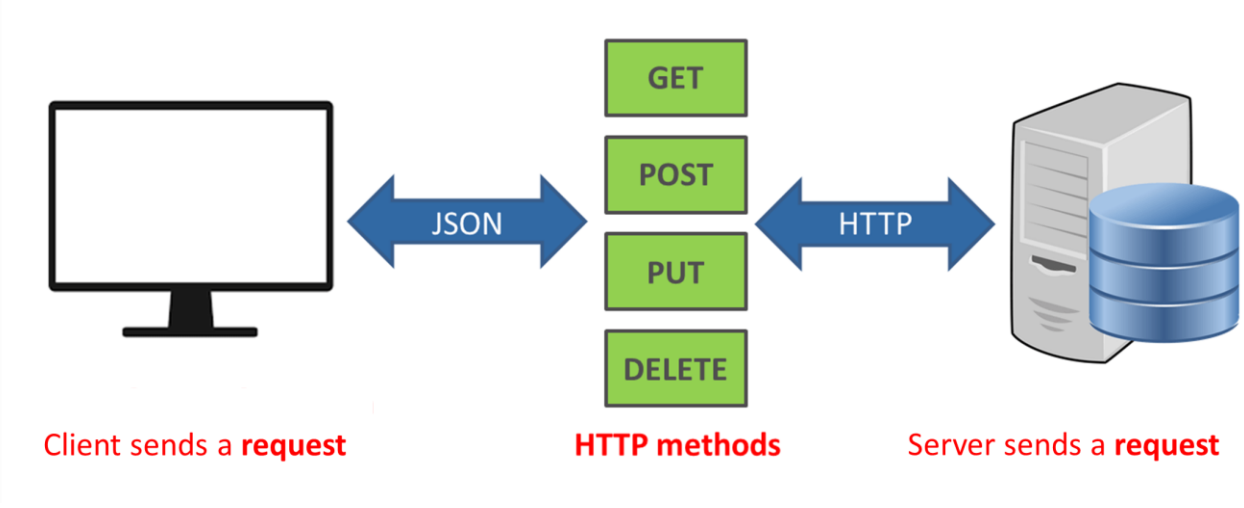
**Cada petición realizada a nuestra API responde a un verbo, y dicho verbo a una operación en común. Mediante los métodos HTTP hacemos las peticiones, lo común es GET y POST, PUT y DELETE.**

| Método | Descripción |
|---|---|
| POST | Utilizado para enviar datos, típicamente para operaciones de "insert". |
| GET | Utilizado para obtener datos a través de consultas "select". |
| PUT | Utilizado para actualizar datos, típicamente para operaciones de "update". |
| DELETE | Utilizado para borrar datos, típicamente para operaciones de "delete". |

## CARACTERÍSTICAS

- Los servicios web desarrollados con REST tiene como característica devolver las respuestas a las peticiones en formatos de documentos XML o JSON, ya que es el lenguaje de intercambio de información más usado.
- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

## MODELO




## REPRESENTACIÓN URI

| | | |
|---|---|---|
| GET | /blog/categories/ | Returns list of blog categories |
| POST | /blog/categories/ | Creates a new blog category |
| DELETE | /blog/categories/{id} | Deletes blog category |
| GET | /blog/categories/{id} | Returns a category with a list of posts |
| PUT | /blog/categories/{id} | Updates a blog category |

## BENEFICIOS

1. Separación entre el cliente y el servidor

   El protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

2. Visibilidad, fiabilidad y escalabilidad.

   La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

3. No importa la tecnología
   EST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

# DESARROLLO Y CONSUMO

## Servicio Web (asmx)

### EJERCICIO

**Crear un servicio web de "Participantes" al curso de Desarrollo de WebServices**

### *Pasos:*

1. Creación del modelo de datos.
2. Creación del Proyecto Web en C# (ServiceParticipantes).
3. Creación de las clases u objetos para el servicio: Participante
4. Creación de un servicio web para obtener el nombre de un participante del curso.
5. Creación de un servicio web para obtener el listado de un participante.
6. Crear los WebMethods.
7. Prueba de funcionamiento a nivel local.
8. Publicación del servicio en IIS.
9. Crear un proyecto web para consumir el servicio web.

## Servicio Web (WCF)

### EJERCICIO

**Crear un servicio web de "Operaciones Financieras", el cual un Banco o Cooperativa pueda realizar las operaciones de Consulta a una Cuenta y Depósito.**

1. Creación del modelo de datos.
2. Creación del proyecto WCF en C# (ServiceFinanciero).
3. Creación de las clases o modelos de Cliente, Cuenta, Deposito.

4. Creación de los Contratos del Servicio (IServiceFinanciero): ListadoClientes, ConsultarCliente, ListadoCuentas, ConsultarCuenta, DepositarFondos.
5. Crear la implementación de los contratos (IServiceFinanciero).
6. Prueba de funcionamiento a nivel local con SoapUI.
7. Publicación del servicio en IIS.

## Servicio Web (REST)

### EJERCICIO

**Crear un servicio web de "Operaciones Financieras", el cual un Banco o Cooperativa pueda realizar las operaciones de Consulta a una Cuenta y Depósito.**

1. Creación del proyecto WEB API en C# (ServiceFinanciero).
2. Creación de los modelos de Cliente, Cuenta, Deposito y atributos.
3. Creación de los controladores (ApiController) y operaciones (Routes): ConsultarCliente, ConsultarCuenta, DepositarFondos, de acuerdo a los métodos disponibles (GET, POST).
4. Prueba de funcionamiento a nivel local con SoapUI.
5. Publicación del servicio en IIS.

## Consumir Servicio Web (MVC)

### EJERCICIO

**Crear un proyecto web con arquitectura MVC para realizar el Consumo de los Servicios Web WCF de "Operaciones Financieras"**

1. Creación del proyecto WEB MVC en C# (MVCServiceFinanciero).
2. Creación de las pantallas: Listado de Clientes, Listado de Cuenta del Cliente, Información de la Cuenta, Operación Depositar Fondos.
3. Codificar o programar el funcionamiento de las pantallas.
4. Prueba de funcionamiento a nivel local.
5. Publicación del proyecto web en IIS.