SERVICES

WEB

Índice de contenido

1. Introducción
2. Vocabulario
3. Qué son los servicios WEB
4. Para qué sirven
5. Cómo funcionan
6. ¿Qué es la arquitectura orientada a servicios SOA?
7. ¿Cuál es el diseño y desarrollo de sistemas usando SOA?
8. Cúales son los beneficios de SOA
9. ¿Qué significa el principio de interoperabilidad de servicios web?
10. Crear un servicio Web. Pasos
11. Componentes de los Web Services
12. Arquitectura de los Web Services
13. Componentes de los servidores en una aplicación Web Service
14. **INTRODUCCIÓN**

Los Web Services permiten a diferentes sistemas operativos y aplicaciones comunicarse entre sí a través de mensajes escritos en XML estandarizado

Un Web Service, o Servicio Web, es un método de comunicación entre dos aparatos electrónicos en una red. Es una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas.

Las aplicaciones escritas en varios lenguajes de programación que funcionan en plataformas diferentes pueden utilizar web services para intercambiar información a través de una red. La interoperatividad, por ejemplo, entre Java y Python o Windows y Linux se debe al uso de estándares abiertos.

Como sistema de mensajes se utiliza XML estandarizado. El protocolo más simple para el intercambio de información entre ordenadores es XML-RPC, que emplea XML para llevar a cabo RPCs. RPC, Remote Procedure Call, es un protocolo de red que permite a un programa a ejecutar código en una máquina remota. Los XML-RPC requests son una combinación entre contenido XML y headers HTTP. La simpleza de los XML-RPC hizo que el estándar evolucionase a SOAP, uno de los componentes básicos de los Web Services.

La base de comunicación entre web services es por tanto el lenguaje XML y el protocolo HTTP.

1. **VOCABULARIO**

***REST***

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

***Servicio Web***

Conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Webxix.

***SMTP***

Simple Mail Transfer Protocol, protocolo para transferencia simple de correo”, es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivosxx.

***SOA***

Arquitectura Orientada a Servicios - SOA - Service Oriented Architecture. Paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos.

***SOAP***

Protocolo Simple de Acceso a Objetos, el cual es un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja.

***SOAP-RRSHB***

Bloque de Cabecera SOAP de Representación de Recursos, con el objeto de optimizar el rendimiento de las aplicaciones basadas en Servicios Web, esta tecnología complementaria a SOAP, representa los recursos que se transmiten en esos mensajes SOAP.

***Tomcat***

Servidor Web de aplicaciones, llamado Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por SunMicrosystems).

***UDDI***

*UDDI* (Universal Description, Discovery and Integration), protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.

***WAR***

Un archivo WAR (de Web Application Archive - Archivo de aplicación web) es un archivo JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, librerías de tags y páginas web estáticas (HTML y archivos relacionados) que juntos constituyen una aplicación web.

***WSDL***

Lenguaje de Descripción de Servicios Web , Web Services Description Language, es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.

***WSS***

Web Service Security,WS-Security (Seguridad en Servicios Web) es un protocolo de comunicaciones que suministra un medio para aplicar seguridad a los Servicios Web.

***XML***

XML (Extensible Markup Language), es el formato estándar para los datos que se vayan a intercambiar.

***XSD***

Viene de XML Schema, es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

**3 ¿QUÉ SON LOS SERVICIOS WEB?**

A los servicios web se le entiende por un conjunto de aplicaciones o de tecnologías incluyendo XML, SOAP, WSDL, UDDI con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios específicos. Los proveedores del servicio son los que “exponen los servicios” ofreciendo sus servicios como procedimientos remotos, mientras que los usuarios “consumidores del servicio” solicitan un servicio llamando a estos procedimientos a través de la Web.

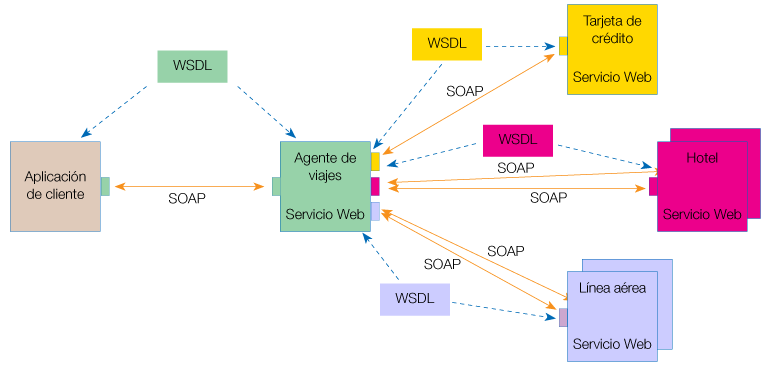
**4 ¿PARA QUÉ SIRVEN?**

Dado que estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, sirven para interactuar entre sí y presentar información dinámica al usuario, esto apoyado en una arquitectura de referencia estándar que facilite la interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar complejas integraciones entre las mismas aplicaciones.

**5 ¿CÓMO FUNCIONAN?**

A partir de la siguiente figura se puede ilustrar el ámbito de acción e interacción de un grupo de servicios web:

Interacción entre servicios web



Fuente: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

En la anterior figura, intervienen los siguientes actores quienes tienen algún tipo de servicio a exponer o consumir: el usuario o cliente, una agencia de viajes, la línea aérea, un hotel y la tarjeta de crédito como medio de pago.

En esta figura, se tiene como fin que un usuario a través de una aplicación web, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus servicios a través de Internet.

La agencia de viajes actúa como un actor intermediario o cliente de otros Servicios Web dado que además de ofrecer a su cliente (usuario) la información requerida, solicita a su vez información a otros recursos (otros Servicios Web) en relación con el hotel y la línea aérea.

Finalmente, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el servicio Web que gestionará el pago.

En todo este proceso de interacción intervienen una serie de tecnologías que hacen posible esta comunicación. Por un lado, interviene el estándar SOAP (Protocolo Simple de Acceso a Objetos), el cual es un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Por otra parte, intervienen los datos o flujo de información que pueden ser transmitidos a través de protocolos de comunicación como HTTP, SMTP, entre otros.

También importante en esta interacción, es el SOAP cuyo fin será especificar el formato de los mensajes, compuesto por una directiva envelope, cuya estructura está formada por los elementos: header (cabecera) y body (cuerpo).

Por otro lado, para optimizar el rendimiento de las aplicaciones basadas en Servicios Web, existen otras tecnologías complementarias a SOAP, que agilizan el envío de los mensajes (MTOM - Mecanismo de Optimización de Transmisión de Mensajes para SOAP) y los recursos que se transmiten en esos mensajes (SOAP-RRSHB - Bloque de Cabecera SOAP de Representación de Recursos).

Otro elemento relevante en esta interacción, es el WSDL (Lenguaje de Descripción de Servicios Web), el cual permite que un servicio y un cliente establezcan un acuerdo en lo que se refiere a los detalles de transporte de mensajes y su contenido (por ejemplo sintaxis y los mecanismos de intercambio de mensajes), a través de un documento procesable por dispositivos, este acuerdo representa una especie de contrato entre el que expone el servicio y el que solicita consume el servicio.

Para hacer frente a la complejidad de los procesos de interacción de las grandes aplicaciones empresariales, surge una tecnología que permite enriquecer las descripciones de las operaciones que realizan sus servicios mediante anotaciones semánticas y con directivas que definen el comportamiento, esto se logra mediante la composición de varios Servicios Web individuales, lo que se conoce como coreografía.

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 Requestor 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.

**6 ¿QUÉ ES LA ARQUITECTURA ORIENTADA A SERVICIOS SOA?**

La Arquitectura Orientada a Servicios (SOA, siglas del inglés Service Oriented Architecture) es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos, que permiten satisfacer necesidades de negocio, las cuales incluyen facilidad y flexibilidad de integración con sistemas legados, entre otros.

Además, permiten crear sistemas de información altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios, entre ellos los servicios web, lo cual facilita la interacción entre diferentes sistemas de información.

**7 ¿CUÁL ES EL DISEÑO Y DESARROLLO DE SISTEMAS USANDO SOA?**

Para el diseño y desarrollo de aplicaciones SOA se aplica la metodología “análisis y diseño orientado a servicios” dentro de un marco de trabajo donde desarrolladores de software deben orientarse ellos mismos a la mentalidad de crear servicios comunes que son orquestados por clientes o middleware para implementar procesos de negocio. Para el desarrollo de dichos servicios web se apoyan en el uso de estándares como: XML, HTTP, SOAP, REST, WSDL, UDDI.

Por otra parte, como marco de trabajo está presente la implementación de las aplicaciones SOA, aunque se puede implementar SOA utilizando cualquier tecnología basada en servicios, es altamente recomendable utilizar servicios web (empleando SOAP y WSDL) en su implementación.

SOA define las siguientes capas de software:

-  Aplicaciones básicas: sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.

-  De exposición de funcionalidades: donde las funcionalidades de la capa aplicativa son expuestas en forma de servicios (generalmente como servicios web).

-  De integración de servicios: facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.

-  De composición de procesos: define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio.

-  De entrega: donde los servicios son desplegados a los usuarios finales.

1. **¿CUÁLES SON LOS BENEFICIOS DE SOA?**

Las características propias de SOA permiten a las organizaciones desde la agilidad en su uso como independencia de las plataformas e infraestructuras tecnológicas, lo que le permite integrarse con sistemas y aplicaciones diferentes de forma sencilla.

Sin embargo, los beneficios que puede obtener una organización que adopte SOA son:

- Mejora en los tiempos de realización de cambios en procesos.

- Facilidad para evolucionar a modelos de negocios basados en tercerización.

-  Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores): esto facilita la integración de sistemas y aplicaciones diferentes, lo cual mejora la comunicación y la capacidad de respuesta con sistemas externos.

-  Le otorga “poder” para reemplazar elementos de la capa aplicativa SOA sin disrupción en el proceso de negocio.

-  Facilidad para la integración de tecnologías disímiles.

-  Mejora en la toma de decisiones: es decir, la organización dispone de mayor información y más actualizada, lo que le permite una respuesta rápida y eficaz cuando surgen problemas o cambios

-  Aplicaciones flexibles: la orientación a servicios permite desarrollar aplicaciones con independencia de las plataformas y lenguajes de programación que realizan los procesos.

-  Aplicaciones reutilizables y adaptables: permite que las aplicaciones existentes para ser reutilizadas y adaptadas a nuevos entornos con facilidad.

-  Reducción de costos: el costo de ampliar o crear nuevos servicios se reduce

considerablemente tanto en aplicaciones nuevas como ya existentes.

* Riesgo de migración: es decir, al adaptar SOA a partir de una tecnología existente se siguen utilizando los componentes existentes, por lo que se reduce el riesgo de introducir fallos.

**9 ¿QUÉ SIGNIFICA EL PRINCIPIO DE INTEROPERABILIDAD DE SERVICIOS WEB?**

Dentro de la arquitectura SOA para la implementación de Servicios Web, la interoperabilidad se considera el principio más relevante para permitir la ejecución de servicios Web distribuidos en múltiples plataformas de software y arquitecturas de hardware.

Para ello, existe la Organización para la Interoperabilidad de Servicios Web (Web Services Interoperability Organization), ente cuya misión es fomentar y promover la Interoperabilidad de Servicios Web (Web Services Interoperability - WS-I) sobre cualquier plataforma, sobre aplicaciones, y sobre lenguajes de programación, también actuando como un integrador de estándares para ayudar al avance de los servicios web de una manera estructurada y coherente.

**10. CREAR UN SERVICIO WEB. PASOS**

En el proceso de creación de servicios web, resulta conveniente además de tener los conceptos claros mencionados anteriormente, seguir los siguientes pasos para la implementación de servicios que sirven para intercambiar datos entre aplicaciones.

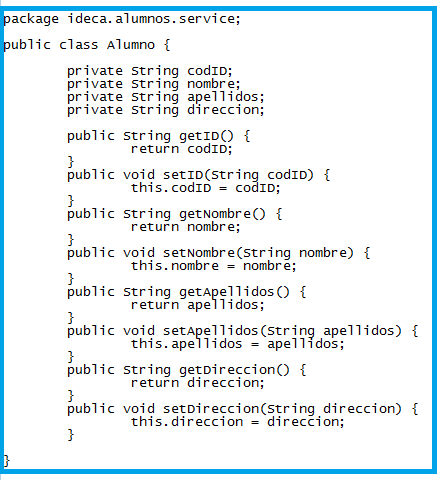
Importante mencionar que se dan los pasos estándar para la creación de los servicios web, sin embargo, existen diversas plataformas para ello, que incorporan sintaxis o directivas complementarias que enriquecen las descripciones de las operaciones y el comportamiento que realizan sus servicios.

**PASO 1. CREAR UN SERVICIO WEB**

Es posible crear un servicio web escribiendo todo el código o bien utilizando alguna aplicación para ello, en nuestro caso se opta por escribir todo el código como forma didáctica para apropiar los conceptos y se utilizará el lenguaje de programación en Java.

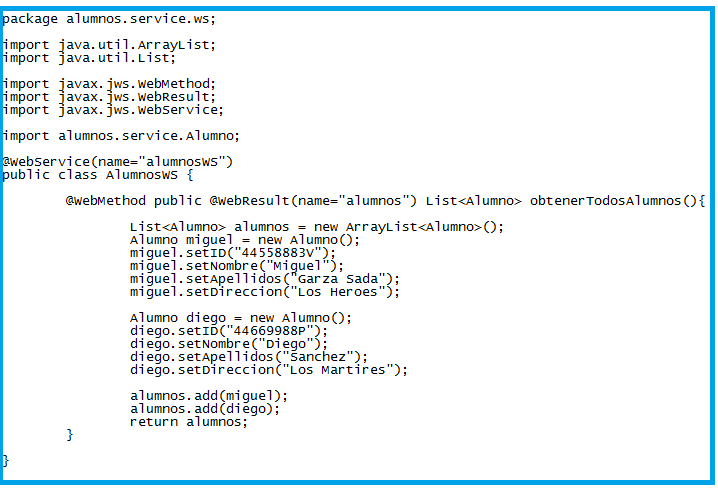
Primero se crearán las clases Java y luego se generen los archivos XSD y los WSDL. Para este ejemplo, se creará una clase en Java llamada “Alumno.java” con los servicios a ofrecer: código ID, nombre, apellidos y dirección.

Clase Java Alumno



Luego se procede a la implementación del servicio web (otra clase Java de nombre AlumnosWS.java) que servirá como intermediario para obtener los servicios de la clase en Java Alumno.

Clase Java AlumnosWS



En este ejemplo, los servicios web se marcan con la notación @WebService(name=”alumnosWS”) seguido del nombre del servicio web. Cada uno de los métodos del servicio Web es un @WebMethod que devuelve un objeto un @WebResult y en caso de tener parámetros serían @WebParam(“nombreParametro”).

**PASO 2. CREE UN WSDL Y XSD DEL SERVICIO WEB**

Seguidamente se crea un archivo .bat (generar\_ws.bat - entorno Windows) para crear el WSDL y el XSD del servicio web:

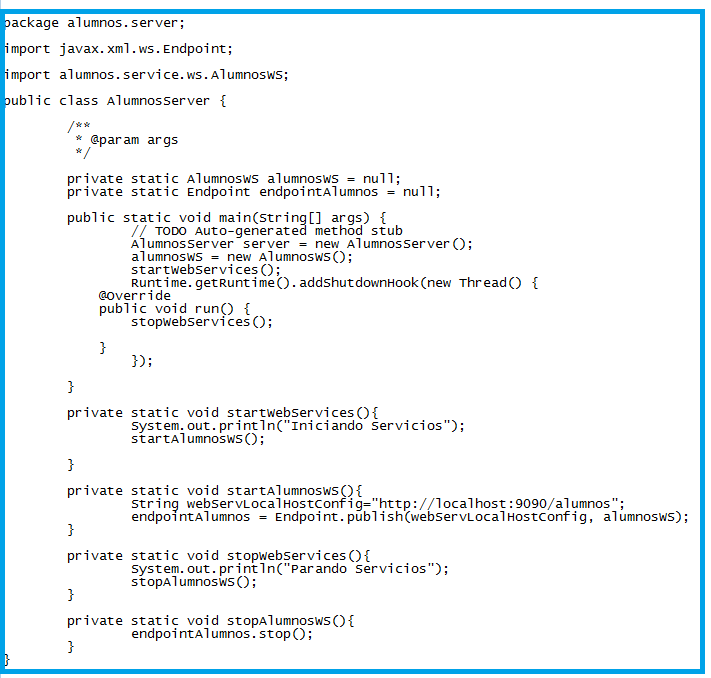
Archivo bat generar\_ws

En este archivo .bat se hace referencia hacia donde están los archivos JDK para ejecutar aplicaciones en Java, se hace referencia también a los archivos .java y se lanza el comando wsgen. Posteriormente, se indica dónde se deben dejar los WSDL y los XSD e indicar cuál es la clase que implementa el servicio WEB.

**PASO 3. PUBLIQUE EL SERVICIO WEB**

Ahora, para publicar el servicio web se creará otra clase java llamada “AlumnosServer.java”:

Clase Java AlumnosServer



Esta clase se utiliza para iniciar los servicios Web y se asigna una URL http://localhost:9090/alumnos. Ahora, para para probar se invoca al método java main y se digita en el browser http://localhost:9090/alumnos?wsdl, como resultado debe salir un archivo en formato XML, si es así, es que está bien publicado el servicio web y el WSDL es accesible.

**PASO 4. CREE UN CLIENTE DE SERVICIO WEB**

Luego se procede a crear un servicio cliente para invocar al nuevo servicio web. Aquí se hace necesario crear un Proyecto Web apoyándonos en la aplicación Eclipse. Seguidamente, se inicia el servidor para que el servicio Web esté accesible, y se crea el siguiente archivo .bat y se ejecuta para que se creen los proxies del servicio WEB:

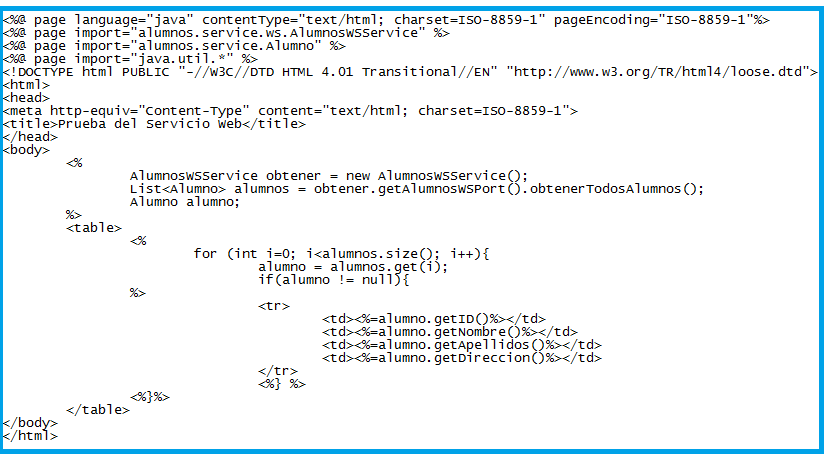
Archivo bat importar\_ws

page19image18632352

En este archivo .bat se indica donde se encuentra los archivos ejecutables de Java JDK, como la ruta del proyecto web creado desde Eclipse, y como último parámetro se indica la URL del archivo WSDL de donde se obtienen los datos de los servicios.

Para mostrar los datos de los servicios se crea un archivo Java con extensión JSP “index.jsp”

Archivo JSP index



Finalmente, usando el browser se invoca este archivo index.jsp y se muestran los datos, esto se realiza a través de una aplicación Tomcat (Servidor Web de aplicaciones), desplegando el Proyecto Web y se invoca el servicio como una aplicación web normal. Para el caso del ejemplo la URL sería http://localhost:8080/AlumnoWServiceClient/ como resultado se muestran en pantalla todos los datos invocados a través de los servicios de la clase Alumno creada inicialmente.

**PASO 5. CREE EL SERVICIO WEB CON SOPORTE WS- SECURITY**

Un servicio web seguro es aquel que utiliza autenticación mediante usuario y contraseña utilizando la directiva WSS – Web Service Security, este define cómo utilizar los tokens de seguridad, XML Signature y XML Encryption en los mensajes SOAP para proporcionar autenticación, confidencialidad e integridad a los Servicios Web.

Para la implementación de la seguridad en un servicio web, se procederá a utilizar varias aplicaciones como es el módulo Rampart para Axis2, el cual proporciona la implementación necesaria para WS-Security, y que combinando otras tecnologías (Axis2, Maven, Rampart, JUnit) se puede generar y probar, en este caso servicios web seguros de un modo relativamente sencillo.

El entorno base para generar y probar este otro servicio web de manera segura es:

* Máquina Virtual Java: JDK 1.5.0\_14 de Sun Microsystems
* Servidor Web: Apache Tomcat 6.0.16
* Motor de servicios web Apache Axis2 1.4.1 con el módulo de Rampart 1.4
* IDE Eclipse 3.3

Instalación de Axis2 con soporte WS-Security:

Como primera parte se requiere realizar la instalación y configuración del módulo web de Axis2 el cual actuará como motor de servicios web, para ello habría que descargar la distribución en WAR de Axis2 que viene comprimida, y luego descomprimir el fichero “axis2- 1.4.1-war.zip” para copiar en el directorio “webapps” del Servidor Web de Aplicaciones Apache Tomcat el archivo axis2.war. Al iniciar los servicios del Tomcat se desplegará esta aplicación y con ello se dispone el motor de servicios web funcionando.

Para la configuración del Motor de servicios web Apache Axis2 1.4.1, se pueden remitir al enlace “Web Services con Axis. Configuración y ejemplo”.

Para incluir las extensiones que soporten WS-Security es necesario descargar la implementación a href=”http://ws.apache.org/rampart/”>**Apache Rampart**, concretamente la versión 1.4. Este módulo es el módulo de seguridad para axis2.

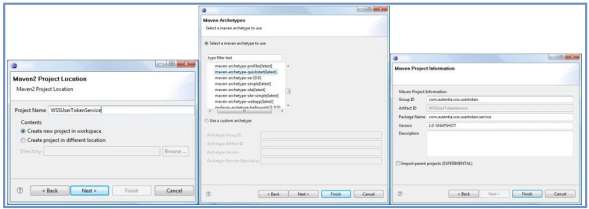
Para instalar el módulo de Rampart, se descomprime el fichero ZIP previamente descargado, y copiar todas las librerías (JAR’s) del directorio “lib” de la distribución de Rampart al directorio “WEB-INF/lib” de nuestra instalación de Axis2, y los ficheros del directorio “modules” de la distribución de Rampart al directorio “WEB-INF/modules” de la instalación de Axis2.

Con esta primera instalación de axis, se considera que se tiene instalado el motor de servicios web con el soporte de seguridad definido en WS-Security.

Creación del servicio web con soporte WS-Security:

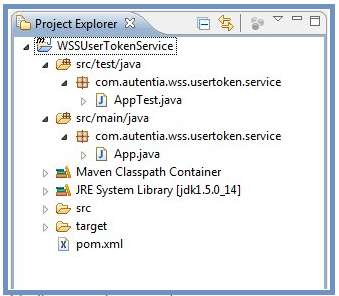
Para crear el servicio web seguro se debe crear inicialmente un proyecto en Maven, esto para la gestión de dependencias, como también utilizar el plugin de empaquetamiento para generar archivos “.aar” (Axis ARchive, equivalente a los “.jar”).

Creación proyecto de Maven para el servicio web



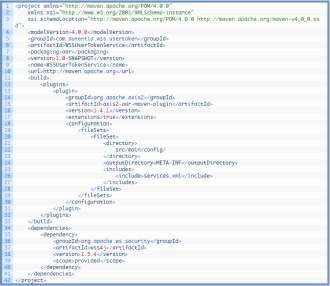
A partir de la siguiente estructura generada, se procede a eliminar las clases “App.java” y “AppTest.java”.

Estructura de directorios del proyecto de Maven



Ahora, se procede a modificar el archivo “pom.xml” para utilizar el plugin “axis2-aar-maven- plugin”. Con este plugin se procede a empaquetar el proyecto directamente como un archivo “aar” para desplegar como servicio web directamente. En el archivo “services.xml” que a continuación se creara en el directorio “src/main/config” incluya el directorio “META-INF” el cual servirá como descriptor del servicio para Axis2. También se creará la dependencia con WSS4J, necesaria para incluir la seguridad al servicio web.

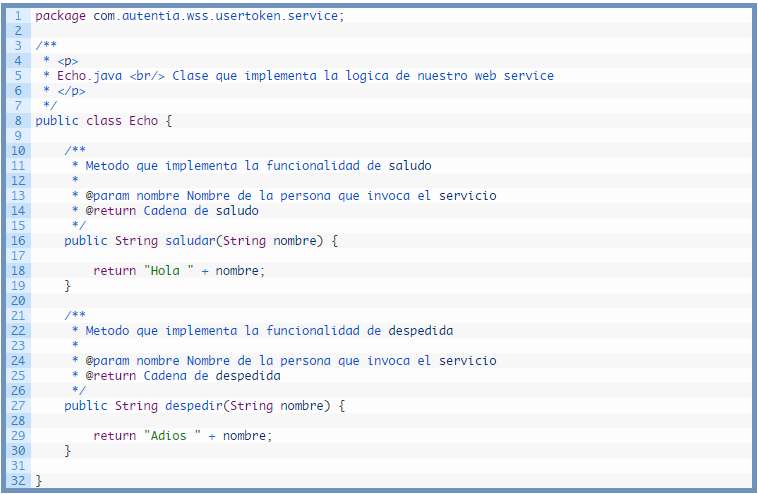
Modificación archivo pom.xml



**PASO 6. CREE LA CLASE QUE EXPONE EL SERVICIO DE SEGURIDAD**

Para este ejemplo se creará la clase Java que actuará como servicio de echo “Echo.java”, para luego indicar la configuración de seguridad en el archivo descriptor del servicio.

Clase en Java que expone el servicio de seguridad



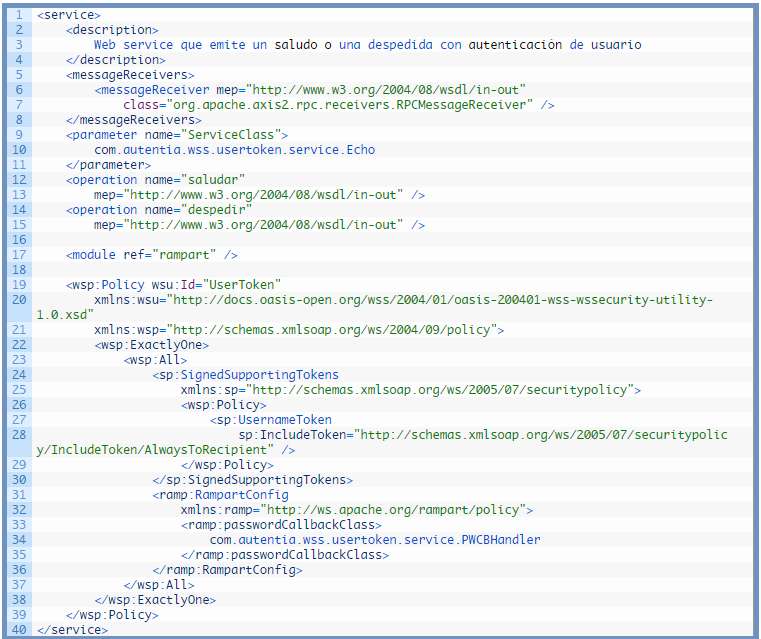
**PASO 7. CREE EL DESCRIPTOR DEL SERVICIO DE SEGURIDAD A DESPLEGAR**

Para desplegar el servicio en Axis2, se creará un archivo descriptor del mismo. En este caso se sigue la estructura indicada en el archivo “pom.xml”, y se crea el archivo “services.xml” en el directorio “src/main/config”.

En este archivo, además de describir el servicio, se incluye una política de seguridad para comprobar la autenticación de quién está llamando al servicio. Para ello, primero se indica la configuración de seguridad de autenticación mediante usuario y contraseña, y luego se configura adecuadamente el módulo de Rampart, que va a ser el encargado de resolver la seguridad del servicio web.

En la configuración de Rampart, se indica la clase responsable de comprobar los usuarios y contraseñas con el elemento “passwordCallBackClass”.

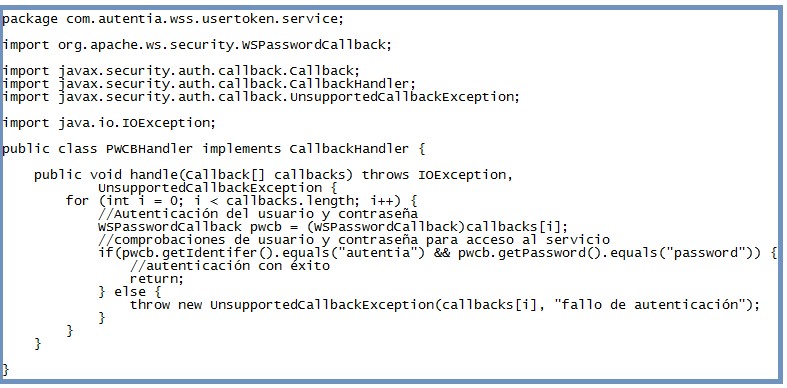
Descriptor del servicio de seguridad



**PASO 8. CREE LA CLASE DE COMPROBACIÓN DE USUARIO Y CONTRASEÑA**

A continuación se crea otra clase en Java para implementar el servicio que comprueba el usuario y contraseña de las peticiones de servicio. La clase será “com.autentia.wss.usertoken.service.PWCBHandler” configurada en el descriptor del servicio para que se encargue de dicha tarea. Esta clase debe heredar de “javax.security.auth.callback.CallbackHandler” y el tipo de “callback” que va a procesar es del tipo “org.apache.ws.security.WSPasswordCallback”.

Clase comprobación de usuario y contraseña



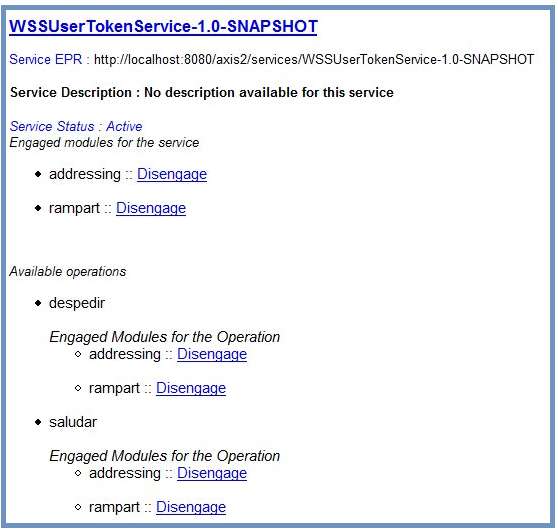
**PASO 9. DESPLIEGUE EL SERVICIO WEB CON SOPORTE WS- SECURITY**

Para desplegar el anterior servicio se procede a empaquetar ejecutando desde la consola el siguiente comando de Maven, situándonos en el directorio raíz del proyecto:

⮚mvn package

Con esta sentencia se genera el fichero “WSSUserTokenService-1.0-SNAPSHOT.aar” en el directorio “target” del proyecto web de seguridad. Luego se copia dicho fichero al directorio “WEB-INF/services” de la instalación de Axis2 y se desplegará el servicio que se puede comprobar en la administración web de Axis2, allí se puede observar que tiene vinculado el módulo de Rampart.

Vista despliegue del servicio de seguridad



**PASO 10. CREE EL CLIENTE PARA EL SERVICIO WEB DE SEGURIDAD**

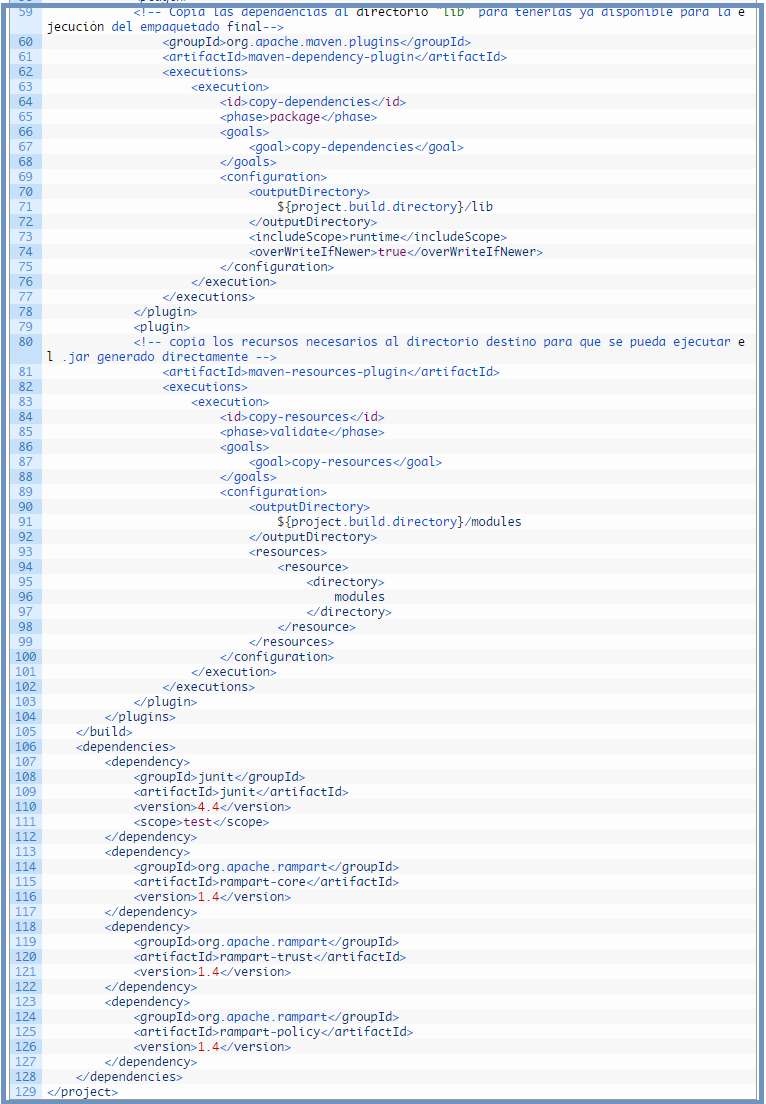
Para crear el servicio de cliente, se debe crear otro proyecto web desde Maven, el nombre de la clase del servicio cliente será “WSSUserTokenClient” y el paquete de las clases será “com.autentia.wss.usertoken.client”.

Modificación del pom.xml: para esta modificación se debe modificar el archivo “pom.xml” del proyecto. Dentro de la modificación se debe realizar la generación automática de las clases clientes del servicio para ser utilizadas directamente por el servicio cliente creado. También, se crearan las dependencias necesarias, en este caso dependencias con los distintos módulos de Rampart, para la seguridad en los servicios web. De tal forma que el archivo“pom.xml” queda de la siguiente forma:

Modificación Cliente de Seguridad Parte 1 – pom.xml



Modificación Cliente de Seguridad Parte 2 – pom.xml



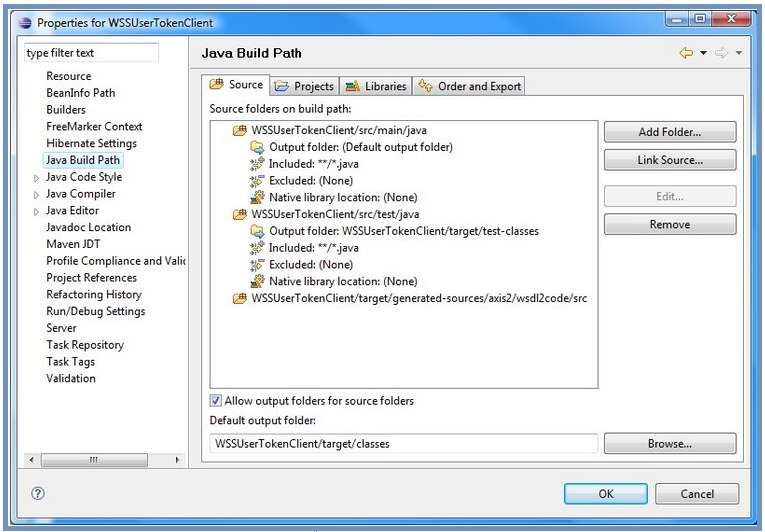
**Creación de la clase Cliente**

Antes de crear la clase cliente se debe generar las clases clientes para acceder al servicio, para ello se ejecuta el siguiente comando de Maven, situándonos en el directorio raíz del proyecto:

⮚mvn wsdl2code:wsdl2code

Como resultado de ejecutar la anterior sentencia, se dispone de un código generado en el directorio “target/generated-sources/axis2/wsdl2code/src”, por lo que desde el proyecto de eclipse se debe indicar que dicho directorio es un directorio de código fuente. A continuación se ilustra cómo se hace en las propiedades del proyecto:

Vista propiedades para WSSUserTokenClient

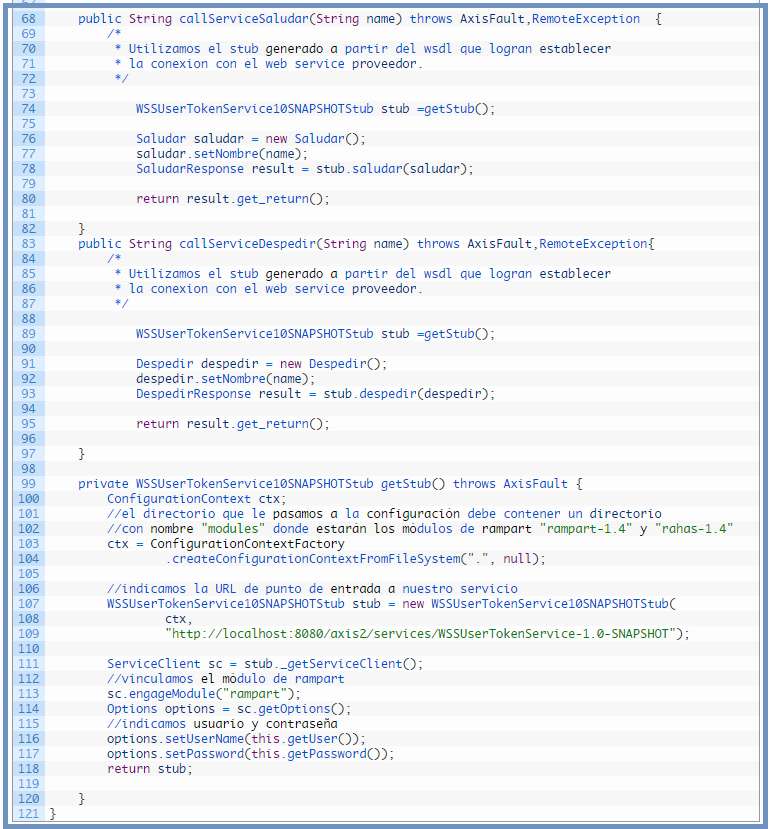


Con lo anterior, ya se procede a crear la clase cliente “EchoClient.java” que tendrá el siguiente código:

Clase en Java del Cliente de Seguridad - Parte 1



Clase en Java del Cliente de Seguridad - Parte 2



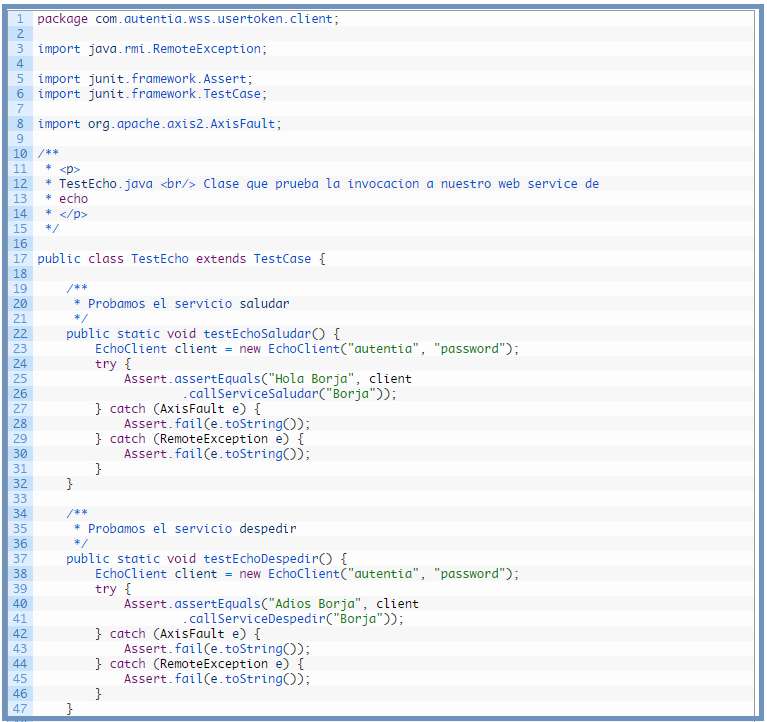
**Módulos de Rampart**

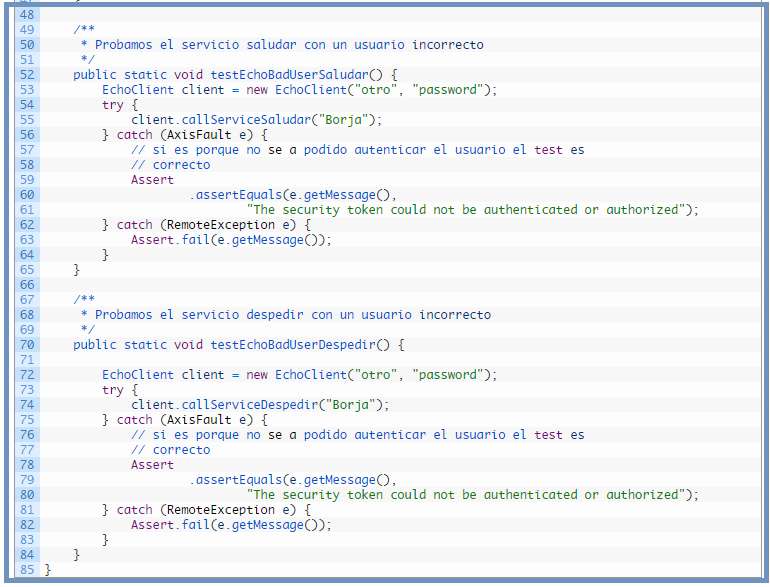
Desde el código de la clase anterior, se puede apreciar la vinculación a la petición del módulo de Rampart; luego cuando se crea el contexto de Rampart se le pasa como parámetro el “path” del directorio padre que contiene un directorio con nombre “modules” donde se encuentran los módulos “rampart-1.4” y “rahas-1.4”. Para el caso del ejemplo, se ha creado un directorio “modules” en el proyecto web, donde se han copiado dichos módulos.

**PASO 11. EJECUTE LAS PRUEBAS UNITARIAS AL SERVICIO WEB DE SEGURIDAD**

Para comprobar que el cliente del servicio web de seguridad funciona correctamente se creara una clase como prueba unitaria utilizando la aplicación JUnit. Esta clase se ubicará dentro del directorio destinado para las pruebas “src/test/java”, para que luego no se incluya en el JAR que se genere. Así entonces, la clase de prueba “TestEcho.java” tendrá el siguiente código:

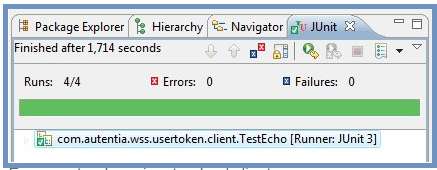
Clase de prueba unitaria al cliente del servicio web de seguridad – Parte 1



Clase de prueba unitaria al cliente del servicio web de seguridad – Parte 2

Realizado lo anterior, se procede a ejecutar los test que se han programado, pulsando sobre la clase “TestEcho.java” con el botón derecho y seleccionando “Run as –> JUnit Test”.

Vista ejecutar pruebas unitarias



**PASO 12. REALICE EL EMPAQUETADO Y EJECUCIÓN DEL SERVICIO CLIENTE**

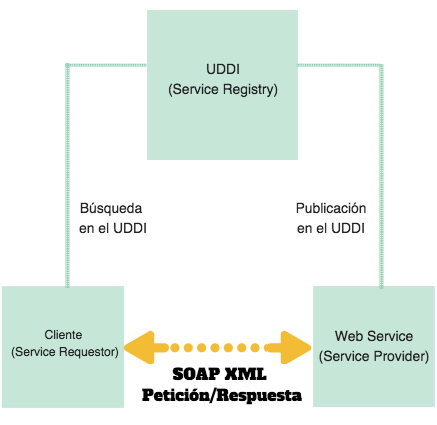
Finalmente, una vez evidenciado con las pruebas unitarias que el cliente está funcionando, se procede a generar un JAR para que se ejecute el servicio cliente directamente; esto se hace ejecutando el siguiente comando de Maven, ubicándose en el directorio raíz del proyecto.

⮚mvn package

Al ejecutar la sentencia anterior, se genera el archivo “WSSUserTokenClient-1.0- SNAPSHOT.jar” en el directorio “target” del proyecto, como también se han copiado todos los módulos de Rampart en el directorio “target/modules” y los JAR’s de todas las dependencias en el directorio “target/lib”. Con ello, desde la consola de ejecución de comandos en Windows, desde el directorio “target” del proyecto, se puede ejecutar directamente el servicio cliente con la sentencia:

⮚java -jar WSSUserTokenClient-1.0-SNAPSHOT.jar

1. **COMPONENTES DE LOS WEB SERVICES**



Los web services estandarizados funcionan con los siguientes componentes:

* SOAP - Simple Object Access Protocol

SOAP es un protocolo escrito en XML para el intercambio de información entre aplicaciones. Es un formato para enviar mensajes, diseñado especialmente para servir de comunicación en Internet, pudiendo extender los HTTP headers. Es una forma de definir qué información se envía y cómo mediante XML. Básicamente es un protocolo para acceder a un Web Service.

* WSDL - Web Services Description Language

WSDL es un lenguaje basado en XML para describir los servicios web y cómo acceder a ellos. Es el formato estándar para describir un web service, y fue diseñado por Microsoft e IBM. WSDL es una parte integral del estándar UDDI, y es el lenguaje que éste utiliza.

* UDDI - Universal Description, Discovery and Integration

**Una vez definido el servicio web, necesitamos darlo a conocer a la comunidad para que sepan de su existencia. UDDI se va a encargar de ello**

UDDI es un estándar XML para describir, publicar y encontrar servicios web. Es un directorio donde las compañías pueden registrar y buscar servicios web. Es un directorio de interfaces de servicios web descritos en WSDL que se comunican mediante SOAP.

La especificación UDDI (Universal Description, Discovery, and Integration) define un modo de publicar y encontrar información sobre servicios Web. UDDI tiene dos funciones: (1) es un protocolo basado en SOAP que define cómo se comunican los clientes UDDI con registros y (2) es un conjunto en particular de registros duplicados globalmente.

En el registro de un servicio intervienen cuatro tipos de estructuras de datos principales:

* El tipo de datos *businessEntity* contiene información sobre la empresa que tiene un servicio publicado.
* El tipo de datos *businessService* es una descripción de un servicio Web.
* El tipo de datos *bindingTemplate* contiene información técnica para determinar el punto de entrada y especificaciones de construcción para invocar un servicio Web.
* El tipo de datos *tModel* proporciona un sistema de referencia que ayuda a descubrir servicios Web y actúa como una especificación técnica de un servicio Web.

UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen. A través de UDDI, se puede publicar y descubrir información de una empresa y de sus servicios. Se puede utilizar sistemas taxonómicos estándar para clasificar estos datos y poder encontrarlos posteriormente en función de la categorización. Lo más importante es que UDDI contiene información sobre las interfaces técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, se puede interactuar con UDDI tanto en tiempo de diseño como de ejecución para descubrir datos técnicos de los servicios que permitan invocarlos y utilizarlos. De este modo, UDDI sirve como infraestructura para una colección de software basado en servicios Web.

* ¿Por qué UDDI? ¿Por qué resulta necesario un registro de este tipo? Teniendo en cuenta que existe una colección de software de miles (quizás millones) de servicios Web, se nos plantean varias cuestiones difíciles:
* ¿Cómo se descubren los servicios Web?
* ¿Cómo se categoriza la información de forma coherente?
* ¿Cómo repercute esto en la localización?
* ¿Cómo afecta a las tecnologías de propietario? ¿Cómo se puede garantizar la interoperabilidad del mecanismo de descubrimiento?
* ¿Cómo se puede interactuar en tiempo de ejecución con este mecanismo de descubrimiento cuando mi aplicación depende de un servicio Web?

La iniciativa UDDI surgió como respuesta a estas preguntas. Varias empresas, incluidas Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP y unas trescientas más (para obtener un listado completo, consulte UDDI: Community [en inglés]), unieron sus esfuerzos para desarrollar una especificación basada en estándares abiertos y tecnologías no propietarias que permitiera resolver los retos anteriores. El resultado, cuya versión beta se lanzó en diciembre de 2000 y estaba en producción en mayo de 2001, fue un registro empresarial global alojado por varios nodos de operadores en el que los usuarios podían realizar búsquedas y publicaciones sin coste alguno.   
  
A partir de la creación de esta infraestructura para servicios Web, los datos sobre estos servicios se pueden encontrar de forma sistemática y confiable en una capacidad universal totalmente independiente de proveedores. Se pueden llevar a cabo búsquedas categóricas precisas utilizando sistemas de identificación y taxonómicos extensibles. La integración de UDDI en tiempo de ejecución se puede incorporar a las aplicaciones. Como resultado, se fomenta el desarrollo de un entorno de software de servicios Web.   
  
**¿Cómo funciona UDDI?**

La información de UDDI se aloja en nodos de operador, empresas que se han comprometido a ejecutar un nodo público conforme a la especificación que rige el consorcio UDDI.org. En la actualidad existen dos nodos públicos que se ajustan a la versión 1 de la especificación UDDI: Microsoft aloja uno e IBM el otro. Hewlett Packard se ha comprometido a alojar un nodo bajo la versión 2 de la especificación. Los operadores del host deben replicar datos entre ellos a través de un canal seguro, para conseguir la redundancia de la información en el registro UDDI. Se pueden publicar los datos en un nodo y descubrirlos en otro tras la réplica. Actualmente, la réplica se produce cada 24 horas. En el futuro, este intervalo entre réplicas se reducirá, ya que habrá más aplicaciones que dependan de los datos de UDDI.

Resulta importante observar que no existen requisitos de propietario respecto al modo en que el operador del host implementa su nodo. El nodo sólo se debe ajustar a la especificación UDDI. El nodo de Microsoft (http://uddi.microsoft.com/default.aspx [en inglés]), por ejemplo, se ha escrito por completo en C# y se ejecuta en producción en tiempo de ejecución en lenguaje común .NET Beta 2. El código de base se beneficia claramente de la compatibilidad nativa con SOAP y de la serialización que ofrecen las clases de sistema .NET. En el lado del servidor, el nodo del operador Microsoft utiliza Microsoft® SQL Server 2000 como almacén de datos. Creo que basta con mencionar que IBM utiliza tecnologías diferentes para ejecutar su nodo, ¿verdad? No obstante, los dos nodos se comportan exactamente igual, ya que se ajustan al mismo conjunto de llamadas a API XML basadas en SOAP. Las herramientas de los clientes pueden interoperar con ambos nodos sin problemas. Por eso, el nodo público UDDI constituye un claro ejemplo de que el modelo de servicios Web XML funciona en entornos heterogéneos.   
  
El próximo paso para comprender la iniciativa UDDI consiste en ver qué datos se almacenan en UDDI y cómo se estructuran. UDDI es relativamente ligero; se ha diseñado como registro, no como depósito. La diferencia, aunque sutil, resulta esencial. Un registro redirige al usuario a recursos, mientras que un depósito sólo almacena información. El registro Microsoft® Windows® puede servir de ejemplo: contiene las configuraciones y parámetros básicos, pero, en última instancia, su función es la de dirigir la aplicación a un recurso o binario. Buscar un componente COM basándonos en su Id. de programa nos conducirá a un Id. de clase, que a su vez nos dirigirá a la ubicación del binario.

UDDI se comporta de forma similar: como el registro de Windows, se basa en identificadores únicos globales (GUID) para garantizar la capacidad de búsquedas y determinar la ubicación de recursos. En última instancia, las consultas a UDDI conducen a una interfaz (un archivo .WSDL, .XSD, .DTD, etc.) o a una implementación (como un archivo .ASMX o .ASP) ubicadas en otro servidor. Por tanto, UDDI puede responder a este tipo de preguntas:

* "¿Qué interfaces de servicios Web basadas en WSDL se han publicado y establecido para un sector determinado?"
* "¿Qué empresas han escrito una implementación basada en una de estas interfaces?"
* "¿Qué servicios Web, categorizados de algún modo, se ofrecen actualmente?"
* "¿Qué servicios Web ofrece una empresa determinada?"
* "¿Con quién se debe poner en contacto el usuario para utilizar los servicios Web de una empresa?"
* "¿Cuáles son los detalles de implementación de un servicio Web concreto?"

**WSDL y UDDI**

WSDL se ha convertido en una pieza clave de la pila de protocolos de los servicios Web. Por eso, es importante saber cómo colaboran UDDI y WSDL y por qué la idea de interfaces frente implementaciones forma parte de cada protocolo. WSDL y UDDI se diseñaron para diferenciar claramente los metadatos abstractos y las implementaciones concretas. Para entender cómo funcionan WSDL y UDDI resulta esencial comprender las consecuencias de esta división.

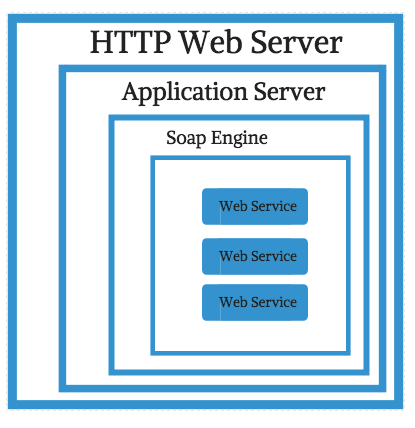
Por ejemplo, WSDL distingue claramente los mensajes de los puertos: los mensajes (la sintaxis y semántica que necesita un servicio Web) son siempre abstractos, mientras que los puertos (las direcciones de red en las que se invoca al servicio Web) son siempre concretos. No es necesario que un archivo WSDL incluya información sobre el puerto. Un archivo WSDL puede contener simplemente información abstracta de interfaz, sin facilitar datos de implementación concretos, y ser válido. De este modo, los archivos WSDL se separan de las implementaciones.

Una de las consecuencias más interesantes de esto es que pueden existir varias implementaciones de una única interfaz WSDL. Este diseño permite que sistemas dispares escriban implementaciones de la misma interfaz, para garantizar así la comunicación entre ellos. Si tres empresas diferentes implementan el mismo archivo WSDL y una parte del software de cliente crea el código auxiliar/proxy a partir de esa interfaz, dicho software se podrá comunicar con las tres implementaciones con el mismo código de base, cambiando simplemente el punto de acceso.   
  
UDDI establece una distinción similar entre la abstracción y la implementación con el concepto de tModels. La estructura tModel, abreviatura de "Technology Model" (modelo de tecnología), representa huellas digitales técnicas, interfaces y tipos abstractos de metadatos. El resultado de los tModels son las plantillas de enlace, que son la implementación concreta de uno o más tModels. Dentro de una plantilla de enlace se registra el punto de acceso de una implementación particular de un tModel. Del mismo modo que el esquema de WSDL permite separar la interfaz y la implementación, UDDI ofrece un mecanismo que permite publicar por separado los tModels de las plantillas de enlace que hacen referencia a ellos. Por ejemplo, un grupo industrial o de estándares publica la interfaz canónica para un sector particular y, a continuación, varias empresas escriben implementaciones de esta interfaz. Obviamente, cada una de estas implementaciones haría referencia al mismo tModel. Los archivos WSDL constituyen un ejemplo perfecto de tModel de UDDI.

1. **ARQUITECTURA DE LOS WEB SERVICES**

* Service Discovery. Responsable de centralizar servicios web en un directorio común de registro y proveer una funcionalidad sencilla para publicar y buscar. UDDI se encarga del Service Discovery.
* Service Description. Uno de los aspectos más característicos de los web 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.

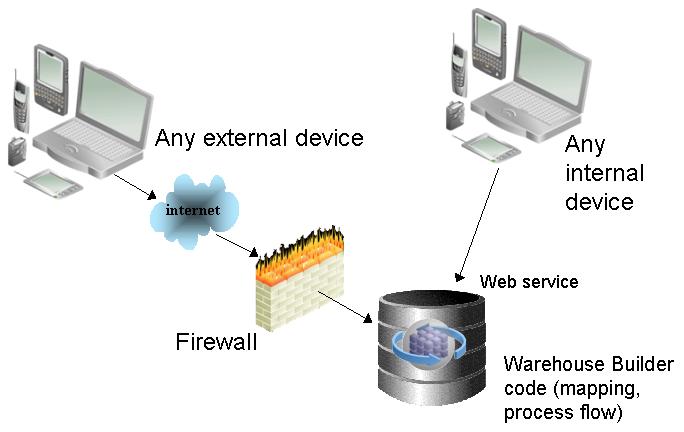
1. **Componentes de los servidores en una aplicación Web Service**



* Web Service. Es el software o componente que realiza las operaciones. Si está escrito en Java, estas operaciones se realizarán en lenguaje Java. Los clientes invocarán estas operaciones enviando mensajes SOAP.
* 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, Java Servlet y JSP container.
* HTTP 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. Los dos más populares en la actualidad son Apache HTTP Server y Nginx.

**¿Qué es y para qué sirve un web service?**

La mayoría de los sitios webs grandes (Facebook, MySpace, Microsoft) usan aplicaciones que utilizan servicios webs (*web services*).



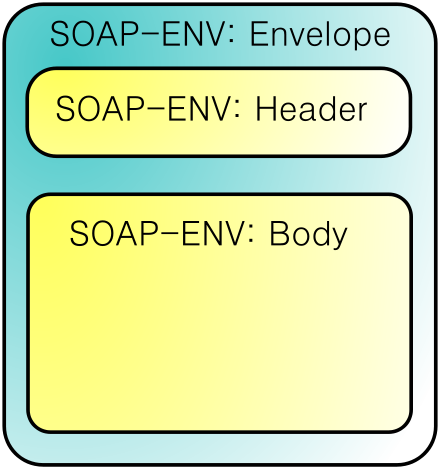
Un *web service* es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en [redes](http://culturacion.com/etiqueta/redes/) de ordenadores como [internet](http://culturacion.com/etiqueta/internet/).

De una manera más clara se podría decir que un web service es una función que diferentes servicios o equipos utilizan; es decir, solo se envían parámetros al servidor (lugar donde está alojado el web service) y éste responderá la petición. Entre algunas que se manejan de utilizar servicios webs en las aplicaciones destacan las siguientes:

* Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
* Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
* Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.
* Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
* Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta, la W3C, por tanto no hay secretismos por intereses particulares de fabricantes concretos y se garantiza la plena interoperabilidad entre aplicaciones.

La principal ventaja de utilizar un servicio [web](http://culturacion.com/etiqueta/web/) es que son bastante prácticos debido a que **son independientes de las aplicaciones**.

Simple Object Access Protocol

[](https://commons.wikimedia.org/wiki/File:SOAP.svg)

Estructura de un mensaje SOAP

**SOAP** (originalmente las siglas de *Simple Object Access Protocol*) es un [protocolo](https://es.wikipedia.org/wiki/Protocolo_(inform%C3%A1tica)) [estándar](https://es.wikipedia.org/wiki/Norma_(tecnolog%C3%ADa)) que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos [XML](https://es.wikipedia.org/wiki/XML). Este protocolo deriva de un protocolo creado por [Dave Winer](https://es.wikipedia.org/w/index.php?title=Dave_Winer&action=edit&redlink=1) en 1998, llamado [XML-RPC](https://es.wikipedia.org/wiki/XML-RPC). SOAP fue creado por [Microsoft](https://es.wikipedia.org/wiki/Microsoft), [IBM](https://es.wikipedia.org/wiki/IBM) y otros. Está actualmente bajo el auspicio de la [W3C](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium). Es uno de los protocolos utilizados en los [servicios Web](https://es.wikipedia.org/wiki/Servicios_Web).

**Índice**

* [1Características](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Caracter%C3%ADsticas)
* [2Historia](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Historia)
* [3Estructura del mensaje](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Estructura_del_mensaje)
* [4Modelo de procesado](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Modelo_de_procesado)
* [5SOAP sobre correo electrónico](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#SOAP_sobre_correo_electr%C3%B3nico)
* [6Ejemplos de mensajes SOAP](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Ejemplos_de_mensajes_SOAP)
* [7Ventajas y desventajas](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Ventajas_y_desventajas)
  + [7.1Ventajas](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Ventajas)
  + [7.2Desventajas](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Desventajas)
* [8Casos de uso](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Casos_de_uso)
* [9Implementación de un servicio web SOAP](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Implementaci%C3%B3n_de_un_servicio_web_SOAP)
* [10Véase también](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#V%C3%A9ase_tambi%C3%A9n)
* [11Referencias](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Referencias)
  + [11.1Recomendaciones de W3C](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Recomendaciones_de_W3C)
  + [11.2Artículos](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Art%C3%ADculos)
* [12Enlaces externos](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#Enlaces_externos)

Enlaces externos

Características[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=1)]

Básicamente SOAP es un **paradigma de mensajería de una dirección sin estado**, que puede ser utilizado para formar protocolos más complejos y completos según las necesidades de las aplicaciones que lo implementan. Puede formar y construir la capa base de una "[pila de protocolos](https://es.wikipedia.org/wiki/Pila_de_protocolos) de web service", ofreciendo un framework de mensajería básica en el cual los web services se pueden construir. Este protocolo está basado en XML y se conforma de tres partes:

* **Sobre (envelope)**: el cual define qué hay en el mensaje y cómo procesarlo.
* **Conjunto de reglas de codificación** para expresar instancias de tipos de datos.
* **La Convención** para representar llamadas a procedimientos y respuestas.

El protocolo SOAP tiene tres características principales:

* **Extensibilidad** (seguridad y WS-routing son extensiones aplicadas en el desarrollo).
* **Neutralidad** (bajo protocolo de transporte [TCP](https://es.wikipedia.org/wiki/Transmission_Control_Protocol) puede ser utilizado sobre cualquier protocolo de aplicación como [HTTP](https://es.wikipedia.org/wiki/HTTP), [SMTP](https://es.wikipedia.org/wiki/SMTP) o [JMS](https://es.wikipedia.org/wiki/JMS)).
* **Independencia** (permite cualquier modelo de programación).

Como ejemplo de cómo el modelo SOAP pueda ser utilizado, consideraremos un mensaje SOAP que podría ser enviado a un web service para realizar la búsqueda de algún precio en una base de datos, indicando para ello los parámetros necesitados en la consulta. El servicio podría retornar un documento en formato XML con el resultado, un ejemplo, precios, localización o características. Teniendo los datos de respuesta en un formato estandarizado procesable (en inglés "parsable"), éste puede ser integrado directamente en un sitio Web o aplicación externa.

La arquitectura SOAP está formada por varias capas de especificación: [MEP](https://es.wikipedia.org/w/index.php?title=Message_Exchange_Patterns&action=edit&redlink=1) (*Message Exchange Patterns*) para el formato del mensaje, enlaces subyacentes del protocolo de transporte, el modelo de procesamiento de mensajes, y la capa de extensibilidad del protocolo. SOAP es el sucesor de [XML-RPC](https://es.wikipedia.org/wiki/XML-RPC), a pesar de que toma el transporte y la neutralidad de la interacción, así como el *envelope* / *header* / *body*, de otros modelos (probablemente de [WDDX](https://es.wikipedia.org/wiki/WDDX)).

Historia[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=2)]

La preocupación por los sistemas distribuidos y de cómo diferentes máquinas podían comunicarse entre sí surgió en la década de los 90. Hasta ese momento, era suficiente con que las aplicaciones de un mismo ordenador pudieran establecer una comunicación.

En 1990, surgieron los modelos [COM](https://es.wikipedia.org/wiki/Component_Object_Model) y [CORBA](https://es.wikipedia.org/wiki/CORBA). El primero, Component Object Model fue creado por Microsoft, y el segundo, CORBA, por el Object Management Group. No obstante, estos dos modelos presentaban un hándicap muy importante: no eran fácilmente interoperables ya que las dos máquinas que llevaran a cabo la comunicación debían soportar COM o CORBA, por tanto únicamente se podía utilizar con dos máquinas COM o dos máquinas CORBA. Más adelante, Microsoft creó DCOM y Sun, [RMI](https://es.wikipedia.org/wiki/Java_Remote_Method_Invocation) (Remote Method Invocation). Aunque estos métodos permitían establecer una conexión entre ordenadores a través de la red, tampoco eran interoperables ya que RMI está disponible únicamente para Java, y por tanto, es dependiente del lenguaje de programación. Por todo ello, Microsoft empezó a interesarse por la computación distribuida basada en [XML](https://es.wikipedia.org/wiki/XML) en el año 1997. Su objetivo era terminar con los problemas de interoperabilidad de las soluciones anteriores y permitir que las aplicaciones se conectaran mediante RPCs (Remote Procedure Calls), utilizando los estándares de comunicación [XML](https://es.wikipedia.org/wiki/XML) y [HTTP](https://es.wikipedia.org/wiki/HTTP).

SOAP fue diseñado como un protocolo de acceso a objetos en 1998 por Dave Winer, Don Box, Bob Atkinson y Mohsen Al-Ghosein por Microsoft, donde Atkinson y Al-Ghosein trabajaban en aquel entonces. La [especificación SOAP](https://es.wikipedia.org/w/index.php?title=Especificaci%C3%B3n_SOAP&action=edit&redlink=1) actualmente es mantenida por el XML Protocol Working Group del [World Wide Web Consortium](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium).

La versión SOAP 1.1 se presentó en el año 2000 e IBM participó en su creación. Esta participación resultó muy positiva ya que se produjeron cambios significativos y cruciales para su posterior uso: se diseñó de una forma más modular y escalable, eliminando los problemas derivados de una tecnología propietaria, en este caso de Microsoft. Además, IBM llevó a cabo una implementación de SOAP en Java y SOAP se integró en Web Services J2EE.

**SOAP** originalmente significaba "Simple Object Access Protocol", pero esta sigla se abandonó con la versión 1.2 de la norma. La versión 1.2 se convirtió en una recomendación del [W3C](https://es.wikipedia.org/wiki/W3C) el 24 de junio de 2003. El acrónimo se confunde a veces con [SOA](https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios), siglas de arquitectura orientada a servicios, pero las siglas no están relacionados.

Después que SOAP se introdujo por primera vez, se convirtió en la capa subyacente de un conjunto más complejo de los web services, basada en la [WSDL](https://es.wikipedia.org/wiki/WSDL) (Web Services Description Language) y [UDDI](https://es.wikipedia.org/wiki/UDDI) (Universal Description Discovery and Integration). Estos servicios, especialmente UDDI, han demostrado ser de mucho menos interés, pero una apreciación de ellos da una comprensión más completa del esperado rol de SOAP comparado a como los webs services están actualmente desarrollados.

Un mensaje SOAP es un documento XML ordinario con una estructura definida en la especificación del protocolo. Dicha estructura la conforman las siguientes partes:

* **Envelope (obligatoria)**: raíz que, de la estructura, es la parte que identifica al mensaje SOAP como tal.
* **Header**: esta parte es un mecanismo de extensión ya que permite enviar información relativa a cómo debe ser procesado el mensaje. Es una herramienta para que los mensajes puedan ser enviados de la forma más conveniente para las aplicaciones. El elemento "Header" se compone a su vez de **"Header Blocks"** que delimitan las unidades de información necesarias para el header.
* **Body (obligatoria)**: contiene la información relativa a la llamada y la respuesta.
* **Fault**: bloque que contiene información relativa a errores que se hayan producido durante el procesado del mensaje y el envío desde el "SOAP Sender" hasta el "Ultimate SOAP Receiver".

En los próximos apartados de este documento se podrá apreciar esta estructura con ejemplos concretos.

Modelo de procesado[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=4)]

El modelo de procesado de SOAP está definido como un sistema distribuido, en el que intervienen diferentes nodos. En un escenario básico, los nodos SOAP se comunican uno asumiendo el rol de **SOAP Sender** y **SOAP Receiver**. Aun así, la especificación define diferentes tipos de nodos en función del rol que asumen en el envío del mensaje:

* **SOAP Sender**: nodo que transmite un mensaje SOAP.
* **SOAP Receiver**: nodo que acepta un mensaje.
* **SOAP message path**: conjunto de nodos por los cuales debe pasar un mensaje SOAP, incluyendo el nodo inicial, cero o más nodos intermediarios y el SOAP Receiver definitivo.
* **Initial SOAP sender**: el "sender" que origina el mensaje y que es el punto de inicio del camino que seguirá el mensaje.
* **SOAP intermediary**: el intermediario actúa como SOAP receiver y como SOAP sender, ya que primero recibe el mensaje para después reenviarlo al siguiente nodo en el camino.
* **Ultimate SOAP receiver**: destino final del mensaje SOAP, es el responsable de procesarlo. Cabe destacar que el mensaje podría no llegar al receptor definitivo debido a que problemas en los intermediarios hagan que se pierda.

Un nodo SOAP puede actuar con uno o varios roles, cada uno de los cuales se encuentra definido mediante una URI conocida como el nombre de rol. Los roles asumidos por un nodo son invariantes durante el envío de un mensaje, teniendo en cuenta la especificación el procesado individual de mensajes. Tal y como se ha comentado, una aplicación puede crear protocolos de comunicación más complejos como capas superiores sobre SOAP, pudiendo definir sus propios roles para poder cumplir con sus necesidades.

La especificación de SOAP define unas normas sobre cómo deben ser procesados, definiendo una serie de pasos que deben cumplir las implementaciones del protocolo. Estos pasos se pueden encontrar en la sección [2.6 de la especificación de W3C](http://www.w3.org/TR/2007/REC-soap12-part1-20070427/#procsoapmsgs).

SOAP sobre correo electrónico[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=5)]

Los desarrolladores de aplicaciones hoy en día, pueden utilizar la infraestructura de correo electrónico de Internet para transmitir mensajes SOAP ya sea como mensajes de correo electrónico de texto o como adjuntos. Los ejemplos que se muestran a continuación muestran un modo de transmitir mensajes SOAP, y deben ser tomados como el modo estándar de hacerlo. Las especificaciones SOAP Versión 1.2 no especifican tal vínculo. Sin embargo, existe una Nota W3C no-normativa [SOAP Email Binding] que describe un vínculo de SOAP con el correo electrónico. Su propósito principal es comenzar a demostrar la aplicación de la Infraestructura general de Vínculos con el Protocolo SOAP.

El ejemplo muestra el mensaje de petición de reserva de viaje del Ejemplo 1 transportado como un mensaje de correo electrónico entre un agente de usuario remitente y un agente de usuario destinatario. Está implícito que el nodo destinatario tiene capacidad para entender SOAP, por lo que se envía el mensaje de correo electrónico para su procesamiento. (Se asume que también el nodo remitente puede manejar errores SOAP que pudiera recibir en la respuesta o correlacionar cualesquiera mensajes SOAP recibidos en respuesta a éste).

**Ejemplo**

De: a.oyvind@miempresa.example.com

A: reservas@empresaviajes.example.org

Asunto: Viaje a LA

Fecha: Thu, 29 Nov 2001 13:20:00 EST

Message-Id: <EE492E16A090090276D208424960C0C@miempresa.example.com>

Content-Type: application/soap+xml

<?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>**Mensaje SOAP del Ejemplo 1 transportado como un mensaje SMTP

El encabezado del Ejemplo tiene la forma estándar de [[RFC 2822](https://tools.ietf.org/html/rfc2822)] para mensajes de [correo electrónico](https://es.wikipedia.org/wiki/Correo_electr%C3%B3nico).

Aunque el correo electrónico es un intercambio de mensajes en un solo sentido, y no se da ninguna garantía de entrega, infraestructuras como la de la especificación Simple Mail Transport Protocol ([SMTP](https://es.wikipedia.org/wiki/SMTP)) ofrecen un mecanismo de notificación de entrega que, en el caso de SMTP, se denominan Delivery Status Notification (DSN) [Notificación de Estado de Entrega] y Message Disposition Notification (MDN) [Notificación de Disposición de Mensaje]. Estas notificaciones toman la forma de mensajes de correo electrónico enviados a la dirección de correo electrónico especificada en el encabezamiento del mensaje de correo. Las aplicaciones, así como los usuarios finales del correo, pueden utilizar estos mecanismos para proporcionar el estado de una transmisión de correo electrónico, pero estos, si existiesen, serían notificaciones al nivel SMTP. El desarrollador de aplicaciones debe comprender completamente las capacidades y limitaciones de estas notificaciones de entrega o el riesgo de asumir que haya existido una entrega del mensaje con éxito cuando podría no haberse producido.

Los mensajes de estado de entrega SMTP son separados del procesamiento del mensaje en la capa SOAP. Las respuestas SOAP resultantes a los datos SOAP serán devueltas a través de un mensaje de correo electrónico nuevo que podría tener o no un enlace con el mensaje de la petición original al nivel SMTP. El uso del encabezado In-reply-to: [En-respuesta-a] según [[RFC 2822](https://tools.ietf.org/html/rfc2822)] puede conseguir una correlación al nivel SMTP, pero no implica necesariamente una correlación al nivel SOAP.

Ejemplos de mensajes SOAP[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=6)]

Como ejemplo se muestra la forma en que un cliente solicitaría información de un producto a un proveedor de servicios Web:

**<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>**

Y esta sería la respuesta del proveedor:

**<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.**</description>**

**<price>**96.50**</price>**

**<inStock>**true**</inStock>**

**</getProductDetailsResult>**

**</getProductDetailsResponse>**

**</soap:Body>**

**</soap:Envelope>**

A pesar de no ser la única opción posible, HTTP fue elegido como protocolo de transporte por sus ventajas, para lidiar con cortafuegos, por ejemplo. Otros protocolos como GIOP/IIOP o DCOM (utilizados en CORBA, RMI y DCOM) suelen ser repelidos por estos cortafuegos.

Ventajas y desventajas[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=7)]

**Ventajas**[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=8)]

1. Debido al uso de XML permite invocar procedimientos remotos de muchos lenguajes, por lo tanto, presenta una gran interoperabilidad.
2. Al utilizar una comunicación vía [HTTP](https://es.wikipedia.org/wiki/HTTP) es fácilmente escalable, además de ser casi siempre permitido por los cortafuegos.
3. Puede ser implementado utilizando cualquier lenguaje y ejecutado en cualquier plataforma.
4. Es posible utilizarlo mediante usuario anónimo y mediante autentificación.
5. Es posible transmitirlo mediante cualquier protocolo de transporte capaz de transmitir texto, típicamente [HTTP](https://es.wikipedia.org/wiki/HTTP) o [SMTP](https://es.wikipedia.org/wiki/SMTP).

**Desventajas**[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=9)]

* Debido al uso de XML para el paso de mensajes, SOAP es considerablemente más lento que otros middleware como [CORBA](https://es.wikipedia.org/wiki/CORBA) ya que los datos binarios se codifican como texto. Para contrarrestar este punto débil en el caso de XML con código binario incrustado se desarrolló un método optimizado de transmisión de mensajes.
* Depende del [WSDL](https://es.wikipedia.org/wiki/WSDL) (Web Services Description Language).
* Al contrario que Java, PHP o Python ciertos lenguajes no ofrecen un apoyo adecuado para su uso ya sea a nivel de integración o de soporte [IDE](https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado).

Casos de uso[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=10)]

La forma más habitual de utilizar el protocolo SOAP es mediante el patrón **petición-respuesta** con remitente SOAP y destinatario final SOAP, el cual es utilizado cuando los mensajes SOAP están predefinidos y únicamente se desea enviar una petición y consultar su valor de retorno.

No obstante, muchas veces este patrón no es suficiente, y es necesario establecer un intercambio múltiple de mensajes entre los nodos. La W3C define dos tipos de intercambios de mensajes SOAP para formar una conversación:

* **Intercambio de mensajes Conversacionales**: permite redefinir la información de la petición. Estos intercambios pueden acabar comportándose como un patrón de mensajes de ida y vuelta.
* **Llamadas a Procedimientos Remotos**: permite encapsular la funcionalidad de procedimientos remotos utilizando las ventajas de XML de extensibilidad y funcionalidad, por este motivo se ha definido en la especificación una representación uniforme para realizar invocaciones y respuestas RPC mediante mensajes SOAP.

En ocasiones, es necesario el uso de intermediarios en las comunicaciones SOAP, la especificación SOAP 1.2 define dos tipos:

* **Intermediario redirector**: se trata de un nodo SOAP, el cual redirige el mensaje SOAP a otro nodo SOAP según lo establecido en un bloque de encabezado que ha recibido el nodo destino o según el patrón de mensajes en uso.
* **Intermediario activo**: realiza un procesamiento adicional del mensaje SOAP antes de redirigirlo, sin utilizar criterios descritos en el encabezado del mensaje o del patrón de mensajes en uso.

Implementación de un servicio web SOAP[[editar](https://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&action=edit&section=11)]

Todos los lenguajes de uso mayoritario en el desarrollo de sistemas web implementan o incluyen algún tipo de soporte para la implementación tanto de web services SOAP como de los clientes que los consumen. Además de librerías que implementan el protocolo a nivel básico, encontramos otras que implementan diferentes escenarios de uso y establecen interfaces más sencillas simplificando la programación.

Estas librerías, utilizadas en conjunto con frameworks de desarrollo de sistemas web agilizan el proceso de desarrollo tanto del web service como de sus clientes, en especial si se genera un fichero WSDL que comunique a los clientes las características del servicio.

* **JAVA**: dentro de su librería estándar se encuentran implementaciones concretas a las que se da **soporte oficial**. También podemos encontrar librerías de terceros que, tal y como se ha comentado, ayudan al desarrollador simplificando las interfaces e implementando los casos de uso más habituales. Cabe destacar que los **IDEs más utilizados ofrecen soporte para la creación de servicios web SOAP** que, entre otras cosas, **generan automáticamente el fichero WSDL** y permiten diseñar de forma visual el API y las llamadas que contendrá. En cuanto el servidor a utilizar, se pueden considerar las opciones típicas en Java: Tomcat, Glassfish, etc. Aun así, la elección del servidor puede suponer algunas ventajas, por ejemplo, Glassfish genera una sencilla interfaz web para probar las diferentes llamadas del servicio. Además, la mayoría de herramientas permiten la [generación del cliente del servicio automáticamente a partir de su fichero WSDL](https://netbeans.org/kb/docs/websvc/jax-ws.html).
* **PHP**: **ofrece soporte y unas librerías de apoyo habilitando la** [**extensión SOAP**](http://www.php.net/manual/es/book.soap.php) en el servidor. Se ha desarrollado un [gran número de librerías de terceros](https://packagist.org/search/?q=SOAP), que combinadas con el uso de frameworks MVC, simplifican las interfaces e implementan los escenarios de uso más habituales. También son habituales las implementaciones de clientes para servicios web públicos concretos.
* **Python**: no ofrece un soporte en sus librerías estandar, sin embargo, existe un gran número de paquetes de terceros que permiten la implementación de servicios web SOAP y sus clientes. En el ámbito del desarrollo de servicios web en Python, predomina la utilización del [**Framework Django**](https://www.djangoproject.com/) que se puede combinar con cualquiera de las implementaciones de SOAP.

**.NET**: dentro del Framework se ofrecen herramientas similares a las de Java para el diseño visual del servicio y la creación automática de WSDL . También da soporte para la creación de los clientes a partir del fichero de definición del servicio. En el caso de .NET, el IDE destacado es Visual Studio. En cuanto a librerías encontramos que el ecosistema .NET ofrece múltiples opciones en varios lenguajes, aunque la apuesta actual de Microsoft para el desarrollo web es su [Framework .NET MVC](https://es.wikipedia.org/wiki/ASP.NET_MVC_Framework). Se debe tener en cuenta, que Microsoft creó el formato [**Windows Communication Foundation**](http://msdn.microsoft.com/es-es/vstudio/aa663324.aspx) que es un modelo para la creación de sistemas orientados a servicios, similar y complementario al WSDL.