GUÍA PRÁCTICA PARA LA CREACIÓN DE SERVICIOS WEB BAJO LOS PRINCIPIOS DE INTEROPERABILIDAD WEB SEGURA

Título	Guía práctica para la creación de servicios web bajo los principios de interoperabilidad web segura.
Versión	1.0
Autor	Unidad Administrativa Especial de Catastro Distrital – Angel Lubiel Simbaqueba Ruiz.
Identificador	IPIG-01
Fecha de creación	2016-02-19
Descripción	El presente documento es una guía que permite llevar a cabo las actividades necesarias para crear servicios web bajo los principios de interoperabilidad web segura, como apoyo para las entidades del orden Distrital que hagan parte de IDECA, y complemento a las actividades en la materia dentro de IDE de Bogotá.
Publicador	Unidad Administrativa Especial de Catastro Distrital – UAECD.
Colaboradores	No aplica.
Tipo	Texto.
Formato	Microsoft Word (.doc)
Fuente	Lineamiento 8 sobre la implementación de servicios web en el marco de la definición de la política de gestión de información geoespacial para el Distrito Capital.
Idioma	Español.
Cobertura	Bogotá Distrito Capital
Derechos	Copyright.
Palabras claves	Servicios Web, IDE, WS-I, interoperabilidad, SOA, XML, WSDL, UDDI.

CONTROL DE VERSIONES

Fecha	Autor/ Modificado por	Versión	Cambio efectuado
2016-02-19	Angel Lubiel Simbaqueba Ruiz	1.0	Primera versión del documento. No hay cambios para registrar.

CONTENIDO

1. OBJETIVO Y ALCANCE

1.1 OBJETIVO

El presente instructivo tiene por objeto guiar de una manera práctica el proceso de creación de servicios web bajo los principios de interoperabilidad web segura, como mecanismo de comunicación estándar entre diferentes aplicaciones que interactúan entre sí, dentro de las entidades del orden Distrital, de forma tal que permitan presentar información dinámica al usuario.

1.2 ALCANCE

La presente guía pretende abarcar a todas las entidades del orden Distrital que forman parte de IDECA, y como complemento al desarrollo de las mismas actividades en la materia dentro de IDE de Bogotá.

2. DEFINICIONES, SIGLAS Y ABREVIATURAS

B	
Batch	Un archivo Batch (o Bat), es un simple archivo de texto (ASCII) en el que cada línea contiene comandos que pueden ser interpretados sucesivamente por DOS[i].
Browser	Un navegador web o explorador web (del inglés, navigator o browser), es una aplicación software que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web de todo el mundo a través de Internet[ii].
C	
Clase java	Las clases en Java son plantillas para la creación de objetos, en lo que se conoce como programación orientada a objetos, la cual es una de los principales paradigmas de desarrollo de software en la actualidad[iii].
E	
Eclipse	Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores[iv].
H	
HTTP	Hypertext Transfer Protocol - Protocolo de transferencia de hipertexto, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web[v].
I	
IDE	Un ambiente de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software[vi].
IDECA	Infraestructura de Datos Espaciales para el Distrito Capital.
Interoperabilidad	Se define interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada[vii].
Instructivo	Documento que detalla la forma de llevar a cabo una generalidad o una

	actividad de un proceso o un procedimiento[1].
J	
JAR	Un archivo JAR (por sus siglas en inglés, Java ARchive) es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java[viii].
JAVA	Es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible[ix].
JDK	Java Development Kit o (JDK), es un software que provee herramientas de desarrollo para la creación de programas en Java[x].
JSP	JavaServer Pages (JSP) es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos[xi].
JUnit	Es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java[xii].
L	
Localhost	En Hardware, en el contexto de redes TCP/IP, localhost es un nombre reservado que tienen todas las computadoras, ratón o dispositivo independientemente de que disponga o no de una tarjeta de red ethernet. El nombre localhost es traducido como la dirección IP de loopback 127.0.0.1 en IPv4, o como la dirección ::1 en IPv6[xiii].
M	
Maven	Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation[xiv].
MTOM	Mecanismo de Optimización de Transmisión de Mensajes para SOAP, con el objeto de optimizar el rendimiento de las aplicaciones basadas en Servicios Web; esta tecnología complementaria a SOAP agiliza el envío de los mensajes para SOAP.
P	
Parámetros	Los parámetros de una función son los valores que esta recibe por parte del código que la llama. Pueden ser tipos simples u objetos[xv].
Plugin	Se entiende como un elemento complemento, que es una aplicación (o

[1] 03-01-PR-01 Procedimiento Administración Documental – UAECD.

	programa informático) que se relaciona con otra para agregarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la interfaz de programación de aplicaciones[xvi].
Protocolo	En redes, un protocolo de comunicaciones o protocolo de red es la especificación de una serie de reglas para un tipo particular de comunicación. La red Internet se basa en el modelo de referencia TCP/IP (Transmission Control Protocol/Internet Protocol) que toma su nombre de los dos principales protocolos que regulan la comunicación a través de esta red[xvii].
Proxies	Un proxy (representante) es un agente o sustituto autorizado para actuar en nombre de otra máquina que lo autoriza a hacerlo y puede utilizarse en el contexto de servidor proxy, que hace de intermediario en las peticiones de recursos que realiza un cliente (A) a otro servidor (C)[xviii].
R	
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.
S	
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 Web[xix].
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 dispositivos[xx].
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.
T	

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 Sun Microsystems)[xxi].
U	
UDDI	UDDI (Universal Description, Discovery and Integration), protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.
URL	Se entiende por localizador de recursos uniforme (conocido por la sigla URL, del inglés Uniform Resource Locator) es un identificador de recursos uniforme (Uniform Resource Identifier, URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo. Están formados por una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que designa recursos en una red, como Internet[xxii].
W	
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[xxiii].
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[xxiv].
X	
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[xxv].
Z	
ZIP	Los archivos Zip (.zip o .zipx) son archivos individuales, algunas veces

llamados "ficheros", que contienen uno o más archivos comprimidos. Los archivos Zip facilitan agrupar archivos relacionados y transportarlos, enviarlos por correo electrónico, descargar y almacenar datos y software de forma más rápida y eficiente[xxvi].

3. GENERALIDADES

En la actualidad, los modelos de desarrollo de software están siendo orientados a servicios lo cual está posibilitando el surgimiento de arquitecturas de aplicación como es la Arquitectura Orientada a Servicios (SOA - *Service Oriented Architecture*) considerada como un paradigma de arquitectura de referencia para diseñar y desarrollar sistemas distribuidos brindando una forma bien definida de exposición e invocación de servicios (entre ellos los servicios web) y facilitando la interacción entre diferentes sistemas propios o de terceros, donde al aumento en el uso de los servicios web, ha popularizado el uso de esta arquitectura SOA.

En este capítulo de generalidades se busca entonces cubrir varios conceptos como SOA, interoperabilidad, servicios web, este último el cual incluye tecnologías como XML[2], SOAP, WSDL[3], UDDI[4], entre otras; que permitan construir soluciones de programación para resolución de problemas de integración entre aplicaciones, y permitir la ejecución de servicios web distribuidos en múltiples plataformas de software y arquitecturas de hardware.

3.1 ¿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 inter operar 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.

3.2 ¿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

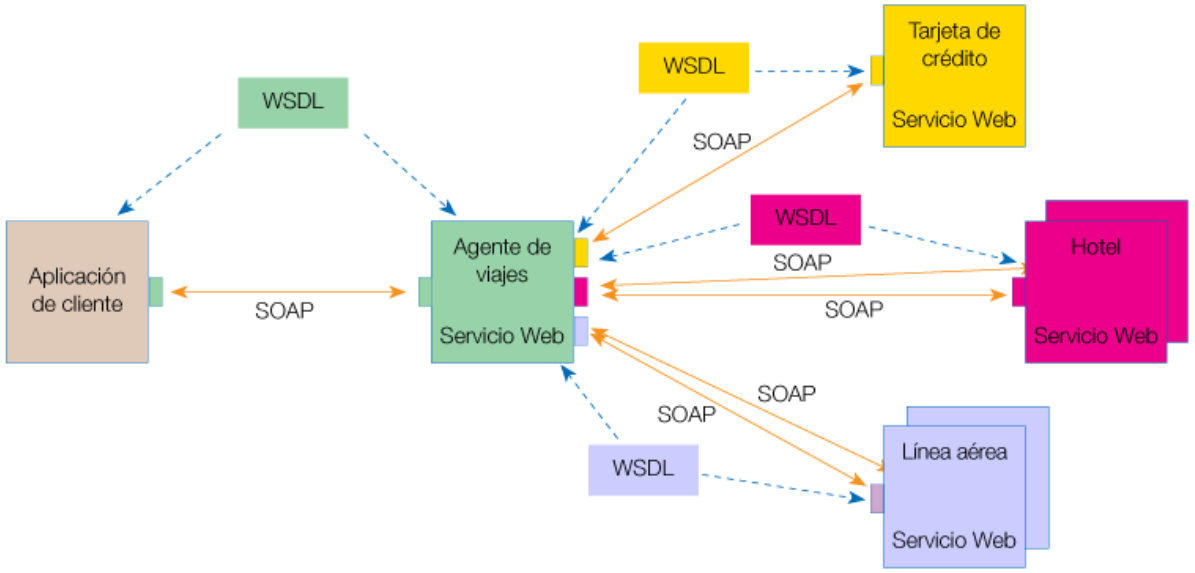
[2] XML (Extensible Markup Language): Es el formato estándar para los datos que se vayan a intercambiar.
[3] WSDL (Web Services Description Language): Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.
[4] UDDI (Universal Description, Discovery and Integration): Protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.

interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar complejas integraciones entre las mismas aplicaciones.

3.3 ¿CÓMO FUNCIONAN?

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

Figura 1. 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[5], SMTP[6], 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.

3.4 ¿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

[5] HTTP (Hypertext Transfer Protocol)
[6] SMTP (Simple Mail Transfer Protocol)

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.

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

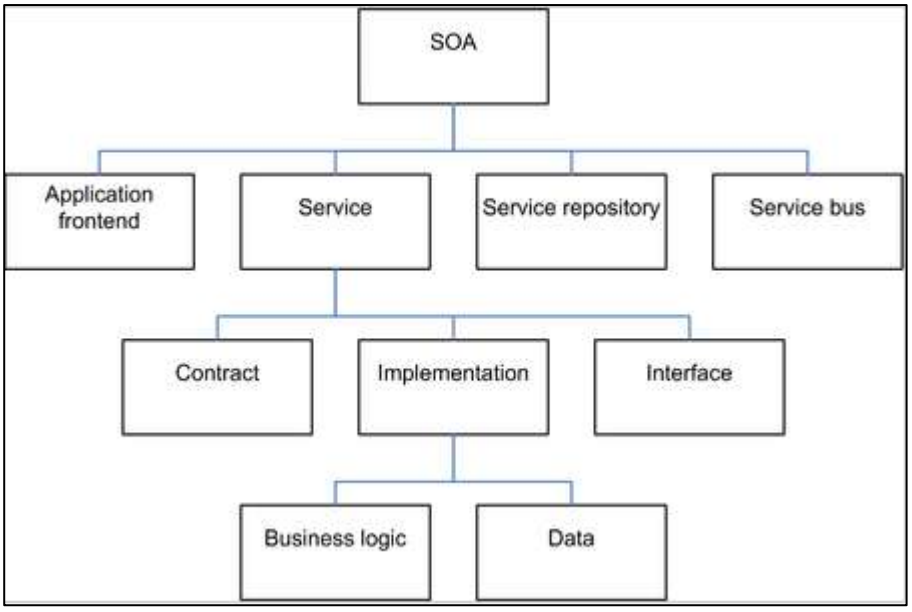
Para el diseño y desarrollo de aplicaciones SOA se aplica la metodología "análisis y diseño orientado a servicios" dentro de un marco de trabajo donde desarrolladores de software deben orientarse ellos mismos a la mentalidad de crear servicios comunes que son orquestados por clientes o middleware para implementar procesos de negocio. Para el desarrollo de dichos servicios web se apoyan en el uso de estándares como: XML, HTTP, SOAP, REST[7], 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.

Según Dirk Krafzig, Karl Banke, y Dirk Slama[xxvii], los siguientes son elementos de una arquitectura SOA:

[7] 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, etc) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

Figura 2. Elementos de una arquitectura SOA



Fuente: https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios#cite_ref-6

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.

3.6 ¿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.

3.7 ¿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.

4. INSTRUCCIÓN

En el proceso de creación de servicios web, resulta conveniente además de tener los conceptos claros mencionados en la sección de “Generalidades”, 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. CREE 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.

Figura 3. Clase Java Alumno

```
package ideca.alumnos.service;

public class Alumno {

	private String codID;
	private String nombre;
	private String apellidos;
	private String direccion;

	public String getID() {
		return codID;
	}
	public void setID(String codID) {
		this.codID = codID;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public String getDireccion() {
		return direccion;
	}
	public void setDireccion(String direccion) {
		this.direccion = direccion;
	}

}
```

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.

Figura 4. Clase Java AlumnosWS

```
package alumnos.service.ws;

import java.util.ArrayList;
import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebService;

import alumnos.service.Alumno;

@WebService(name="alumnosWS")
public class AlumnosWS {

        @WebMethod public @WebResult(name="alumnos") List<Alumno> obtenerTodosAlumnos(){

                List<Alumno> alumnos = new ArrayList<Alumno>();
                Alumno miguel = new Alumno();
                miguel.setID("44558883V");
                miguel.setNombre("Miguel");
                miguel.setApellidos("Garza Sada");
                miguel.setDireccion("Los Heroes");

                Alumno diego = new Alumno();
                diego.setID("44669988P");
                diego.setNombre("Diego");
                diego.setApellidos("Sanchez");
                diego.setDireccion("Los Martires");

                alumnos.add(miguel);
                alumnos.add(diego);
                return alumnos;
        }

}
```

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:

Figura 5. Archivo bat generar_ws

```
PATH=%PATH%;C:\Java\jdk1.7.0_02\bin;
SET CODEPATH="C:\soap\AlumnosService"
SET WSCLASSPATH="C:\soap\Alumnosservice\bin";

wsgen -keep -cp %WSCLASSPATH% -verbose -wsdl -r %CODEPATH%\wsdl -s %CODEPATH%\src -d %CODEPATH%\bin  alumnos.service.ws.AlumnosWS

pause
```

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 creara otra clase java llamada "AlumnosServer.java":

Figura 6. Clase Java AlumnosServer

```
package alumnos.server;

import javax.xml.ws.Endpoint;

import alumnos.service.ws.AlumnosWS;

public class AlumnosServer {

	/**
	 * @param args
	 */

	private static AlumnosWS alumnosWS = null;
	private static Endpoint endpointAlumnos = null;

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		AlumnosServer server = new AlumnosServer();
		alumnosWS = new AlumnosWS();
		startWebServices();
		Runtime.getRuntime().addShutdownHook(new Thread() {
	    @Override
	    public void run() {
	        stopWebServices();

	    }
		});

	}

	private static void startWebServices(){
		System.out.println("Iniciando Servicios");
		startAlumnosWS();

	}

	private static void startAlumnosWS(){
		String webServLocalHostConfig="http://localhost:9090/alumnos";
		endpointAlumnos = Endpoint.publish(webServLocalHostConfig, alumnosWS);
	}

	private static void stopWebServices(){
		System.out.println("Parando Servicios");
		stopAlumnosWS();
	}

	private static void stopAlumnosWS(){
		endpointAlumnos.stop();
	}
}
```

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:

Figura 7. Archivo bat importar_ws

```
PATH=%PATH%;C:\Java\jdk1.7.0_02\bin;
SET CODEPATH= "C:\soap\AlumnosServiceClient"

wsimport -d %CODEPATH%\build\classes -extension -p alumnos.service -s %CODEPATH%\src -verbose http://localhost:9090/alumnos?wsdl

pause
```

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”

Figura 8. Archivo JSP index

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import="alumnos.service.ws.AlumnosWSService" %>
<%@ page import="alumnos.service.Alumno" %>
<%@ page import="java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Prueba del Servicio Web</title>
</head>
<body>
	<%
		AlumnosWSService obtener = new AlumnosWSService();
		List<Alumno> alumnos = obtener.getAlumnosWSPort().obtenerTodosAlumnos();
		Alumno alumno;
	%>
	<table>
		<%
			for (int i=0; i<alumnos.size(); i++){
				alumno = alumnos.get(i);
				if(alumno != null){
		%>
				<tr>
					<td><%=alumno.getID()%></td>
					<td><%=alumno.getNombre()%></td>
					<td><%=alumno.getApellidos()%></td>
					<td><%=alumno.getDireccion()%></td>
				</tr>
				<%} %>
		<%}%>
	</table>
</body>
</html>
```

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[xxviii]".

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.

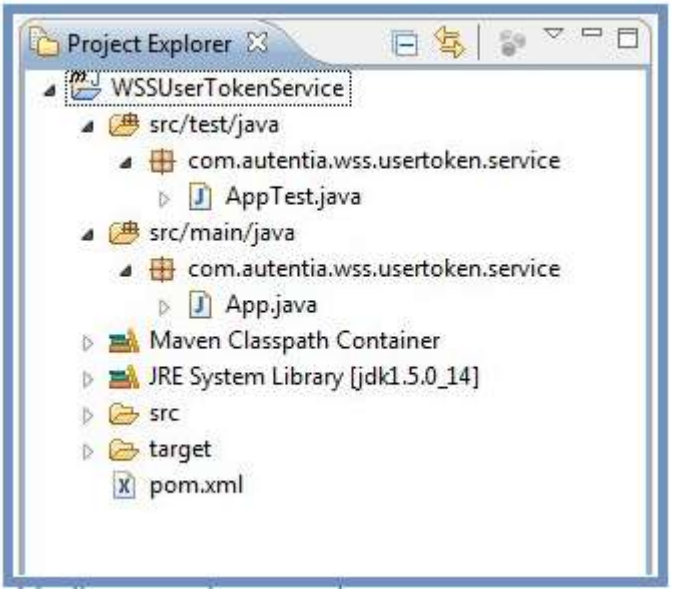
Creación del servicio web con soporte WS-Security:

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”).

Figura 9. Creación proyecto de Maven para el servicio web

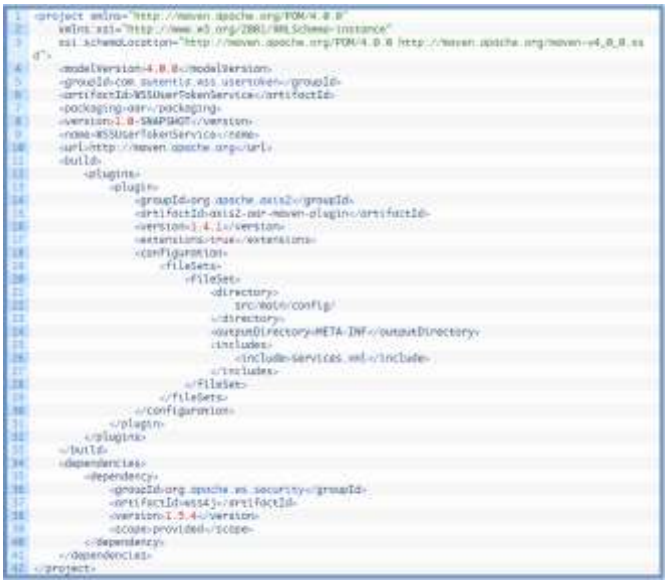
A partir de la siguiente estructura generada, se procede a eliminar las clases “App.java” y “AppTest.java”.

Figura 10. 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.

Figura 11. Modificación archivo pom.xml



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

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

Figura 12. Clase en Java que expone el servicio de seguridad

```java
package com.autentia.wss.usertoken.service;

/**
 * <p>
 * Echo.java <br/> Clase que implementa la logica de nuestro web service
 * </p>
 */
public class Echo {

    /**
     * Metodo que implementa la funcionalidad de saludo
     *
     * @param nombre Nombre de la persona que invoca el servicio
     * @return Cadena de saludo
     */
    public String saludar(String nombre) {

        return "Hola " + nombre;
    }

    /**
     * Metodo que implementa la funcionalidad de despedida
     *
     * @param nombre Nombre de la persona que invoca el servicio
     * @return Cadena de despedida
     */
    public String despedir(String nombre) {

        return "Adios " + nombre;
    }

}
```

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”.

Figura 13. Descriptor del servicio de seguridad

```
<service>
    <description>
        Web service que emite un saludo o una despedida con autenticación de usuario
    </description>
    <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
            class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
    </messageReceivers>
    <parameter name="ServiceClass">
        com.autentia.wss.usertoken.service.Echo
    </parameter>
    <operation name="saludar"
        mep="http://www.w3.org/2004/08/wsdl/in-out" />
    <operation name="despedir"
        mep="http://www.w3.org/2004/08/wsdl/in-out" />

    <module ref="rampart" />

    <wsp:Policy wsu:Id="UserToken"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:SignedSupportingTokens
                    xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                    <wsp:Policy>
                        <sp:UsernameToken
                            sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolic
y/IncludeToken/AlwaysToRecipient" />
                    </wsp:Policy>
                </sp:SignedSupportingTokens>
                <ramp:RampartConfig
                    xmlns:ramp="http://ws.apache.org/rampart/policy">
                    <ramp:passwordCallbackClass>
                        com.autentia.wss.usertoken.service.PWCBHandler
                    </ramp:passwordCallbackClass>
                </ramp:RampartConfig>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
</service>
```

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".

Figura 14. Clase comprobación de usuario y contraseña

```
package com.autentia.wss.usertoken.service;

import org.apache.ws.security.WSPasswordCallback;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import java.io.IOException;

public class PWCBHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            //Autenticación del usuario y contraseña
            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
            //comprobaciones de usuario y contraseña para acceso al servicio
            if(pwcb.getIdentifer().equals("autentia") && pwcb.getPassword().equals("password")) {
                //autenticación con éxito
                return;
            } else {
                throw new UnsupportedCallbackException(callbacks[i], "fallo de autenticación");
            }
        }
    }

}
```

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.

Figura 15. 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:

Figura 16. Modificación Cliente de Seguridad Parte 1 – pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.autentia.wss.usertoken</groupId>
    <artifactId>WSSUserTokenClient</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>WSSUserTokenClient</name>
    <url>http://maven.apache.org</url>
    <build>

        <plugins>
            <plugin>
            <!-- Genera automaticamente las clases clientes del servicio para ser utilizadas en nuestro proyecto -->
                <groupId>org.apache.axis2</groupId>
                <artifactId>axis2-wsdl2code-maven-plugin</artifactId>
                <version>1.4.1</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>wsdl2code</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <packageName>
                        com.autentia.wss.usertoken.client
                    </packageName>
                    <wsdlFile>
                        http://localhost:8080/axis2/services/WSSUserTokenService-1.0-SNAPSHOT?wsdl
                    </wsdlFile>
                </configuration>
            </plugin>
            <plugin>
            <!-- Genera un fichero .jar ejecutable incluyendo el classpath a las dependencias -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>
                                com.autentia.wss.usertoken.client.EchoClient
                            </mainClass>
                            <packageName>
                                com.autentia.wss.usertoken.client
                            </packageName>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                        </manifest>
                        <manifestEntries>
                            <mode>development</mode>
                            <url>${pom.url}</url>
                        </manifestEntries>
                    </archive>
                </configuration>
            </plugin>
            <plugin>
```

Figura 17. Modificación Cliente de Seguridad Parte 2 – pom.xml

```
            <!-- Copia las dependencias al directorio "lib" para tenerlas ya disponible para la ejecucion del empaquetado final-->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                        <configuration>
                            <outputDirectory>
                                ${project.build.directory}/lib
                            </outputDirectory>
                            <includeScope>runtime</includeScope>
                            <overWriteIfNewer>true</overWriteIfNewer>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
            <!-- copia los recursos necesarios al directorio destino para que se pueda ejecutar el .jar generado directamente -->
                <artifactId>maven-resources-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy-resources</id>
                        <phase>validate</phase>
                        <goals>
                            <goal>copy-resources</goal>
                        </goals>
                        <configuration>
                            <outputDirectory>
                                ${project.build.directory}/modules
                            </outputDirectory>
                            <resources>
                                <resource>
                                    <directory>
                                        modules
                                    </directory>
                                </resource>
                            </resources>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.rampart</groupId>
            <artifactId>rampart-core</artifactId>
            <version>1.4</version>
        </dependency>
        <dependency>
            <groupId>org.apache.rampart</groupId>
            <artifactId>rampart-trust</artifactId>
            <version>1.4</version>
        </dependency>
        <dependency>
            <groupId>org.apache.rampart</groupId>
            <artifactId>rampart-policy</artifactId>
            <version>1.4</version>
        </dependency>
    </dependencies>
</project>
```

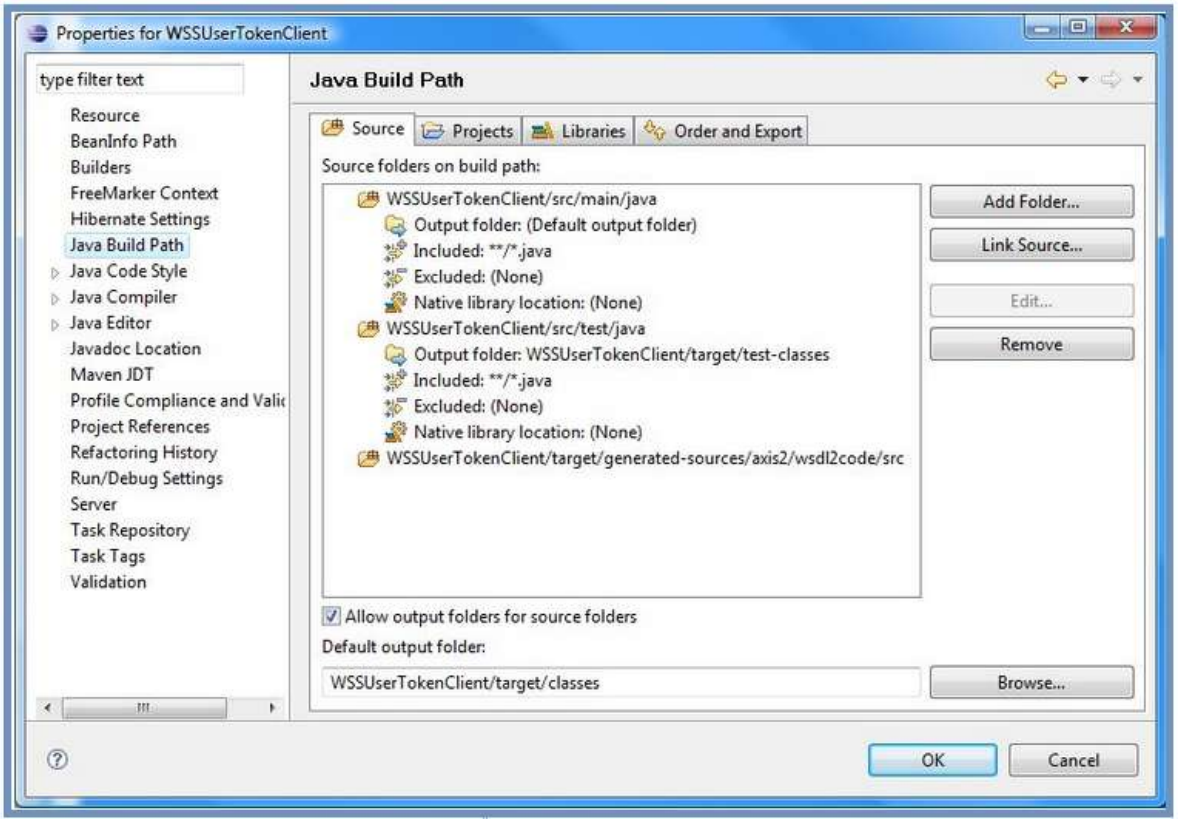
Creación de la clase Cliente

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:

Figura 18. Vista propiedades para WSSUserTokenClient



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

Figura 19. Clase en Java del Cliente de Seguridad - Parte 1

```java
package com.autentia.wss.usertoken.client;

import java.rmi.RemoteException;

import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.context.ConfigurationContext;
import org.apache.axis2.context.ConfigurationContextFactory;

import com.autentia.wss.usertoken.client.WSSUserTokenService10SNAPSHOTStub.Despedir;
import com.autentia.wss.usertoken.client.WSSUserTokenService10SNAPSHOTStub.DespedirResponse;
import com.autentia.wss.usertoken.client.WSSUserTokenService10SNAPSHOTStub.Saludar;
import com.autentia.wss.usertoken.client.WSSUserTokenService10SNAPSHOTStub.SaludarResponse;

/**
 * <p>
 * EchoClient.java <br/> Clase que prueba la invocacion a nuestro web service de
 * echo
 * </p>
 */

public class EchoClient {

    private String user;
    private String password;
    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public EchoClient(String user, String password){
        this.setUser(user);
        this.setPassword(password);
    }

    /**
     * Metodo principal de la clase
     *
     * @param args
     */
    public static void main(String[] args) {

        EchoClient client = new EchoClient("autentia","password");
        try {
            System.out.println(client.callServiceSaludar("Borja"));
            System.out.println(client.callServiceDespedir("Borja"));
        } catch (AxisFault e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

```

Figura 20. Clase en Java del Cliente de Seguridad - Parte 2

```
68      public String callServiceSaludar(String name) throws AxisFault,RemoteException  {
69          /*
70           * Utilizamos el stub generado a partir del wsdl que logran establecer
71           * la conexion con el web service proveedor.
72           */
73
74              WSSUserTokenService10SNAPSHOTStub stub =getStub();
75
76              Saludar saludar = new Saludar();
77              saludar.setNombre(name);
78              SaludarResponse result = stub.saludar(saludar);
79
80              return result.get_return();
81
82      }
83      public String callServiceDespedir(String name) throws AxisFault,RemoteException{
84          /*
85           * Utilizamos el stub generado a partir del wsdl que logran establecer
86           * la conexion con el web service proveedor.
87           */
88
89              WSSUserTokenService10SNAPSHOTStub stub =getStub();
90
91              Despedir despedir = new Despedir();
92              despedir.setNombre(name);
93              DespedirResponse result = stub.despedir(despedir);
94
95              return result.get_return();
96
97      }
98
99      private WSSUserTokenService10SNAPSHOTStub getStub() throws AxisFault {
100         ConfigurationContext ctx;
101         //el directorio que le pasamos a la configuración debe contener un directorio
102         //con nombre "modules" donde estarán los módulos de rampart "rampart-1.4" y "rahas-1.4"
103         ctx = ConfigurationContextFactory
104                 .createConfigurationContextFromFileSystem(".", null);
105
106         //indicamos la URL de punto de entrada a nuestro servicio
107         WSSUserTokenService10SNAPSHOTStub stub = new WSSUserTokenService10SNAPSHOTStub(
108                 ctx,
109                 "http://localhost:8080/axis2/services/WSSUserTokenService-1.0-SNAPSHOT");
110
111         ServiceClient sc = stub._getServiceClient();
112         //vinculamos el módulo de rampart
113         sc.engageModule("rampart");
114         Options options = sc.getOptions();
115         //indicamos usuario y contraseña
116         options.setUserName(this.getUser());
117         options.setPassword(this.getPassword());
118         return stub;
119
120     }
121 }
```

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:

Figura 21. Clase de prueba unitaria al cliente del servicio web de seguridad – Parte 1

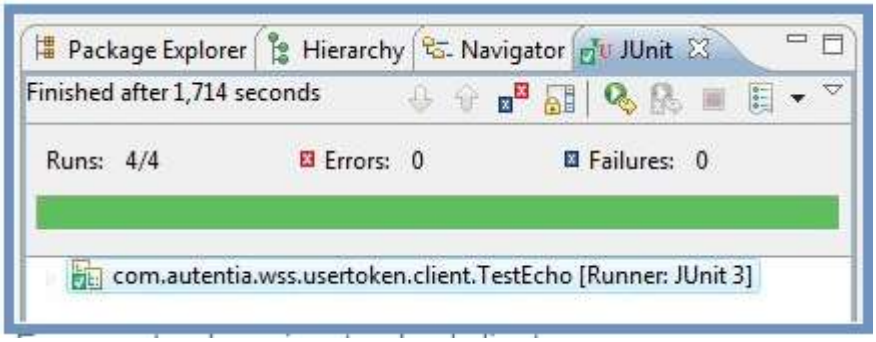
```
1  package com.autentia.wss.usertoken.client;
2
3  import java.rmi.RemoteException;
4
5  import junit.framework.Assert;
6  import junit.framework.TestCase;
7
8  import org.apache.axis2.AxisFault;
9
10 /**
11  * <p>
12  * TestEcho.java <br/> Clase que prueba la invocacion a nuestro web service de
13  * echo
14  * </p>
15  */
16
17 public class TestEcho extends TestCase {
18
19     /**
20      * Probamos el servicio saludar
21      */
22     public static void testEchoSaludar() {
23         EchoClient client = new EchoClient("autentia", "password");
24         try {
25             Assert.assertEquals("Hola Borja", client
26                     .callServiceSaludar("Borja"));
27         } catch (AxisFault e) {
28             Assert.fail(e.toString());
29         } catch (RemoteException e) {
30             Assert.fail(e.toString());
31         }
32     }
33
34     /**
35      * Probamos el servicio despedir
36      */
37     public static void testEchoDespedir() {
38         EchoClient client = new EchoClient("autentia", "password");
39         try {
40             Assert.assertEquals("Adios Borja", client
41                     .callServiceDespedir("Borja"));
42         } catch (AxisFault e) {
43             Assert.fail(e.toString());
44         } catch (RemoteException e) {
45             Assert.fail(e.toString());
46         }
47     }
```

Figura 22. Clase de prueba unitaria al cliente del servicio web de seguridad – Parte 2

```
48
49      /**
50       * Probamos el servicio saludar con un usuario incorrecto
51       */
52      public static void testEchoBadUserSaludar() {
53          EchoClient client = new EchoClient("otro", "password");
54          try {
55              client.callServiceSaludar("Borja");
56          } catch (AxisFault e) {
57              // si es porque no se a podido autenticar el usuario el test es
58              // correcto
59              Assert
60                      .assertEquals(e.getMessage(),
61                              "The security token could not be authenticated or authorized");
62          } catch (RemoteException e) {
63              Assert.fail(e.getMessage());
64          }
65      }
66
67      /**
68       * Probamos el servicio despedir con un usuario incorrecto
69       */
70      public static void testEchoBadUserDespedir() {
71
72          EchoClient client = new EchoClient("otro", "password");
73          try {
74              client.callServiceDespedir("Borja");
75          } catch (AxisFault e) {
76              // si es porque no se a podido autenticar el usuario el test es
77              // correcto
78              Assert
79                      .assertEquals(e.getMessage(),
80                              "The security token could not be authenticated or authorized");
81          } catch (RemoteException e) {
82              Assert.fail(e.getMessage());
83          }
84      }
85  }
```

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

Figura 23. 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

5. REFERENCIAS

[i] < http://es.ccm.net/faq/2057-que-es-un-archivo-batch>
[ii] < https://seguinfo.wordpress.com/2007/06/26/%C2%BFque-es-un-web-browser-o-navegador-3/>
[iii] < http://panamahitek.com/que-son-las-clases-en-java/>
[iv] < https://es.wikipedia.org/wiki/Eclipse_(software)>
[v] < https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol>
[vi] < https://es.wikipedia.org/wiki/Ambiente_de_desarrollo_integrado>
[vii] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
[viii] < https://es.wikipedia.org/wiki/Java_Archive>
[ix] < https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)>
[x] < https://es.wikipedia.org/wiki/Java_Development_Kit>
[xi] < https://es.wikipedia.org/wiki/JavaServer_Pages>
[xii] < https://es.wikipedia.org/wiki/JUnit>
[xiii] < https://es.wikipedia.org/wiki/Localhost>
[xiv] < https://es.wikipedia.org/wiki/Maven>
[xv] < https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Par%C3%A1metros_de_una_funci%C3%B3n>
[xvi] < https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)>
[xvii] < https://es.wikibooks.org/wiki/Tecnolog%C3%ADas_de_Internet/Protocolos>
[xviii] < https://es.wikipedia.org/wiki/Servidor_proxy>
[xix] < http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb>
[xx] < https://es.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol>
[xxi] < https://es.wikipedia.org/wiki/Tomcat>
[xxii] < https://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme>
[xxiii] < https://es.wikipedia.org/wiki/WAR_(archivo)>
[xxiv] < https://es.wikipedia.org/wiki/WS-Security>
[xxv] < https://es.wikipedia.org/wiki/XML_Schema>
[xxvi] < http://www.winzip.com/es/aboutzip.html>
[xxvii] Enterprise SOA. Prentice Hall, 2005
[xxviii] Web Services con Axis2. Configuración y ejemplo. Consultado en <http://www.adictosaltrabajo.com/tutoriales/web-services-axis-2/>