Cómo crear un cliente SOAP en Java sin usar AXIS paso a paso

Introducción

Cuando nos encontramos ante la necesidad de desarrollar un cliente SOAP en Java, normalmente lo que se hace es generar las clases *stub* necesarias mediante **AXIS**, o también **CXF** o **Metro**. Puede que no siempre sea la opción más conveniente o quizá no sea posible el uso de alguna de estas herramientas, aunque eso sí, no cabe duda que facilitan muchísimo un desarrollo que sin ellas sería bastante arduo.

Y bien, ¿por qué motivo podría ser interesante complicarse la vida no usando estas librerías? De primeras se me ocurren varios:

1- **Si eres estudiante**, para aprender cómo funciona el protocolo SOAP internamente y probarlo de primera mano, con posibilidad de experimentar más allá de lo que te permitirían estas herramientas.

2- **Si eres profesor**, para impartir de una forma práctica el funcionamiento interno de SOAP.

3- **Si eres un desarrollador**, puede que te encuentres en la necesidad de desarrollar el código lo más eficiente posible eliminando partes innecesarias (por ejemplo, si desarrollas para dispositivos móviles, con limitaciones), o bien lo puedes necesitar por cuestiones de arquitectura o diseño técnico.

De cualquier forma, siempre será bueno para todo desarrollador enriquecer sus conocimientos y abrirse horizontes aprendiendo una forma “diferente” de hacer las cosas.

En este tutorial mostraré cómo desarrollar dicho cliente paso a paso, de esa forma no habitual, sin usar ninguna de las utilidades mencionadas, y además de una forma sencilla y clara. A partir de ahí, que cada uno concluya de qué manera pueden resultarle útiles estas técnicas.

Requisitos previos

Para sacar provecho a este tutorial recomiendo disponer de conocimientos de nivel medio de las siguientes tecnologías:

- Lenguaje Java
- XML y Xpath
- Protocolos HTTP y SOAP
- Maven

Antes de empezar...

¿Qué es soapUI?

Es una excelente herramienta para trabajar con servicios web. Nos permite probar y simular servicios, generar código SOAP para un servicio a partir de su definición WSDL, entre otras funciones, y la podemos encontrar en versión freeware y de pago, y como aplicación de escritorio o como plugin para varios IDES.

¿Qué es freemarker?

Traducido de su web, es un “motor de plantillas”; una herramienta genérica para generar textos (de cualquier tipo, desde HTML hasta código fuente autogenerado) basado en plantillas. Es un paquete Java, una librería de clases para programadores Java. No es una aplicación para usuario final en sí mismo, sino algo que los programadores pueden incluir en sus productos.

Manos a la obra

Lo que haremos en este tutorial es:

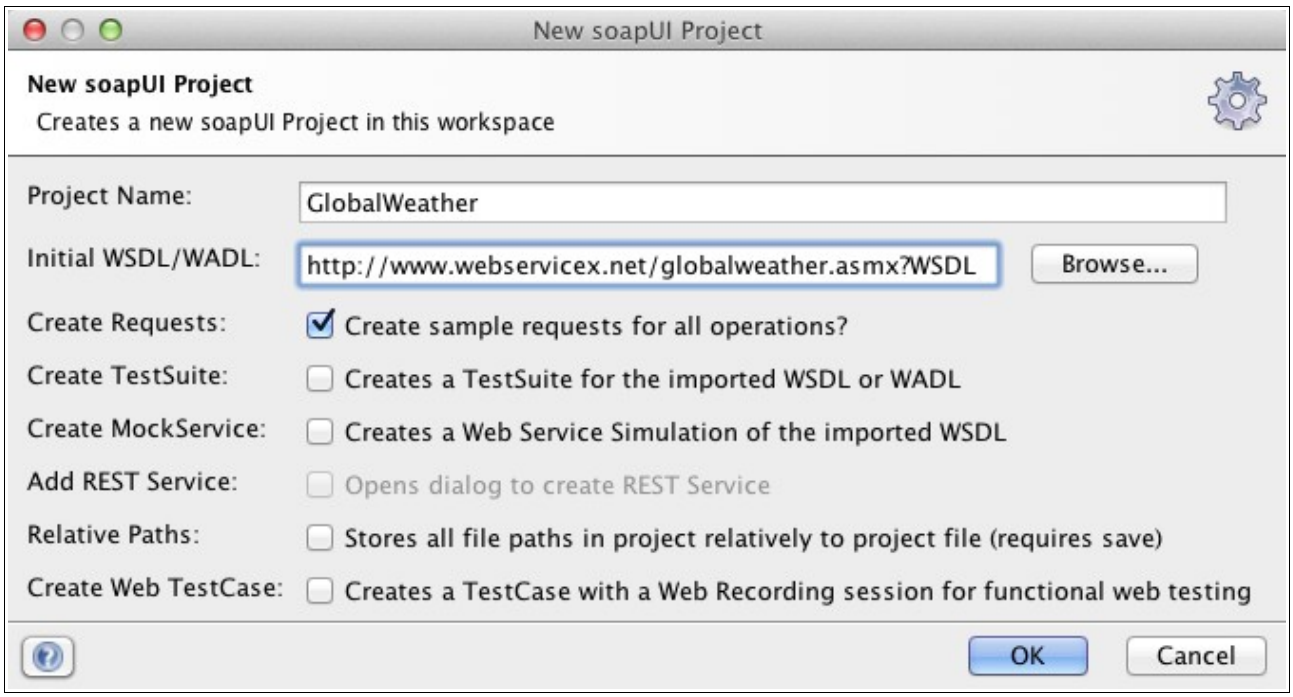
1. Construir manualmente la petición HTTP que se hará desde nuestro cliente Java.
   1. Emplearemos soapUI como ayuda para construir la petición HTTP.
   2. Después nos serviremos de Freemarker para construir la llamada HTTP+SOAP y personalizar los parámetros de la petición.
2. Ejecutar la llamada HTTP construida con la ayuda de Apache HttpClient.
3. Interpretar la respuesta del servicio web usando Xpath.

Necesitaremos un servicio web a donde conectar nuestro cliente SOAP. Usaremos para el ejemplo el servicio Global Weather, todo un clásico gracias al cual podremos conocer cómo está el tiempo en cualquier ciudad del mundo.

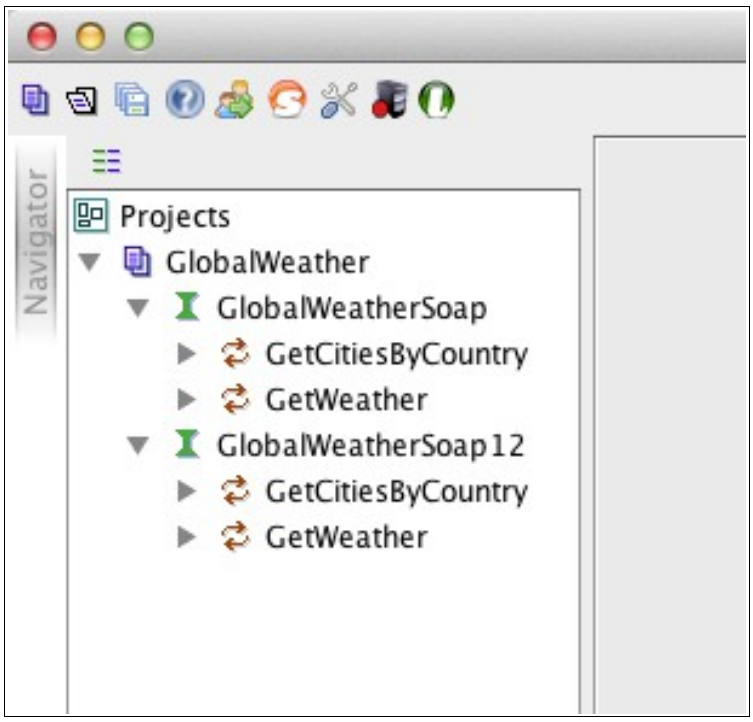
1.- Creación de proyecto GlobalWeather con soapUI

Primeramente, si no tienes instalada la herramienta soapUI, deberás descargarla e instalarla desde su sitio web: http://www.soapui.org/
Como indiqué antes, puedes conseguir una versión freeware en modo aplicación de escritorio o plugin para tu IDE favorito. Personalmente recomiendo la aplicación de escritorio.

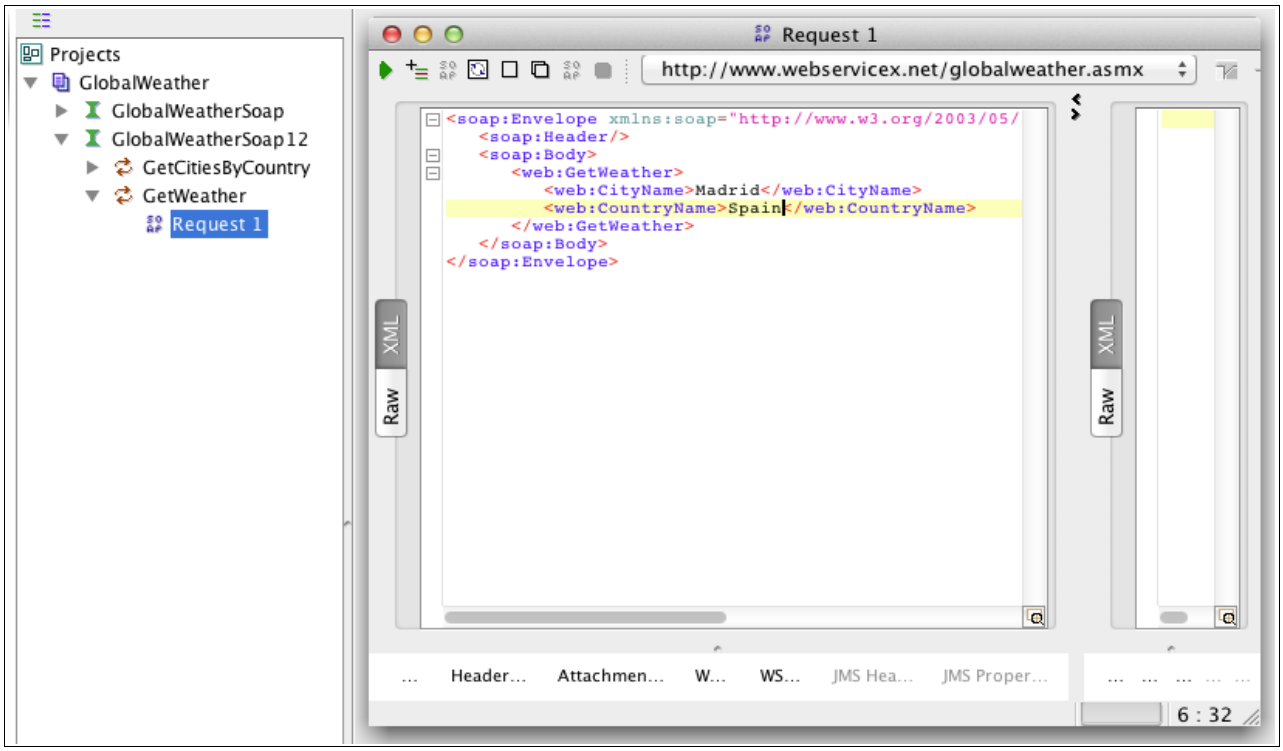
Si ya tienes instalado soapUI crearemos un nuevo proyecto usando como nombre de proyecto GlobalWeather y como WSDL inicial el correspondiente al servicio web GlobalWeather, ‘‘http://www.webservicex.net/globalweather.asmx?WSDL”, y marcamos la opción “Create sample requests for all operations?” como se ve en la captura:



Tras pulsar OK se creará la estructura de proyecto:



Para el ejemplo emplearemos la versión 1.2 de SOAP e intentaremos obtener los datos del tiempo de Madrid. Desplegamos la operación *GetWeather* de la versión 1.2 y rellenamos nuestra petición (Request 1) indicando que queremos obtener el tiempo de Madrid de la siguiente forma:



La operación se podrá ejecutar pulsando el botón verde en forma de “Play”:

Si todo ha funcionado correctamente el servicio nos devolverá una respuesta indicándonos la información meteorológica de Madrid en el momento de la ejecución. **Lo importante aquí es que ya tenemos casi hecha la petición HTTP que tendremos que enviar al servidor, y podremos ver la respuesta, y sabremos cómo interpretarla y explotarla posteriormente.**

2.- Creación de proyecto Java

Crearemos un proyecto Java de consola con nuestro **IDE** y herramientas habituales. Para este ejemplo usé como IDE **Eclipse**, y me construí un proyecto **Maven** simple, sin arquetipo, pero dejo esto a libre elección.

Lo importante serán las dependencias que vamos a necesitar, que si usamos Maven serán las siguientes:

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.2.5</version>
</dependency>
<dependency>
    <groupId>freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.9</version>
</dependency>
```

Si no usas Maven tendrás que descargar las librerías y sus correspondientes dependencias. Puedes conseguir las librerías necesarias de estos enlaces:

- Freemarker: http://freemarker.sourceforge.net/
- Apache HttpClient: http://hc.apache.org/httpcomponents-client-ga/index.html

Necesitaremos un lugar donde guardar la plantilla de Freemarker. En mi caso he seguido el estándar Maven y me he creado la plantilla (*template.ftl*) en la siguiente ruta:

“src/main/resources/templates/template.ftl”

3.- Creación de la plantilla freemarker

La plantilla de *freemarker* nos servirá para construir la llamada que haremos al servicio web. Para ello haremos uso de la plantilla que ya nos ha creado automáticamente soapUI en el primer paso de este tutorial. Crearemos la plantilla y mediante un simple copy-paste de soapUI a la plantilla ya casi la tenemos.

En la carpeta *templates* creamos el archivo *template.ftl* y en él incluimos el contenido generado mediante soapUI. La ciudad y el país serán datos variables que pasaremos a *freemarker* para que él construya la petición. Indicaremos esto a *freemarker* usando ${...} quedando como resultado algo así:

template.ftl

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:web="http://www.webserviceX.NET">
   <soap:Header/>
   <soap:Body>
      <web:GetWeather>
         <web:CityName>${ciudad}</web:CityName>
         <web:CountryName>${pais}</web:CountryName>
      </web:GetWeather>
   </soap:Body>
</soap:Envelope>
```

4.- Creación de la petición HTTP-SOAP mediante freemarker

Para trabajar con la plantilla necesitaremos crear primero una configuración de freemarker. Después definir y cargar la plantilla, crear la información variable que vamos a usar (en este caso, ciudad y país), y por último construir con todo esto la que será nuestra petición HTTP-SOAP.

```
// Configuración Freemarker
Configuration cfg = new Configuration();

// Cargar plantilla
Template template = cfg.getTemplate("src/main/resources/templates/template.ftl");

// Modelo de datos
Map<String, Object> data = new HashMap<String, Object>();
data.put("ciudad", "Madrid");
data.put("pais", "Spain");

// Crear mensaje SOAP HTTP
StringWriter out = new StringWriter();
template.process(data, out);
String strRequest = out.getBuffer().toString();
```

Mediante este código conseguiremos crear la que será la petición SOAP y para comprobar que todo fue bien podríamos mostrar el contenido de la variable *strRequest* por consola. Debería aparecer el mismo contenido que teníamos originalmente, pero esta vez creado dinámicamente:

strRequest

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:web="http://www.webserviceX.NET">
   <soap:Header/>
   <soap:Body>
      <web:GetWeather>
         <web:CityName>Madrid</web:CityName>
         <web:CountryName>Spain</web:CountryName>
      </web:GetWeather>
   </soap:Body>
</soap:Envelope>
```

5.- Llamada al webservice mediante HttpClient

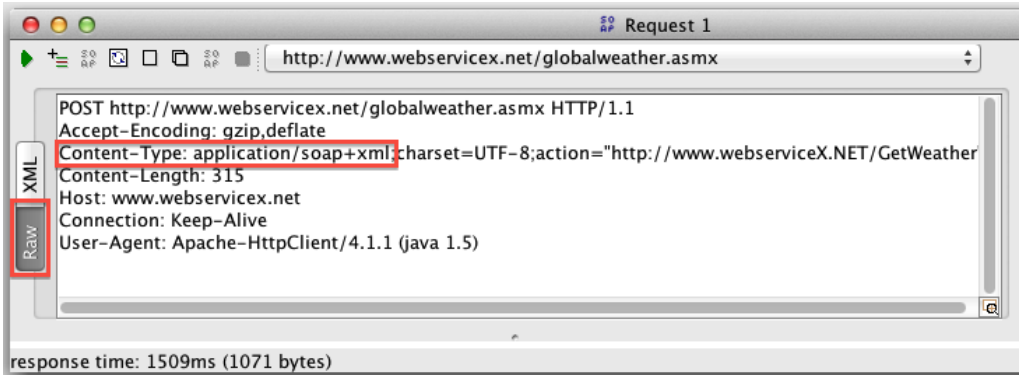
Una vez que ya tenemos la petición SOAP creada, el siguiente paso será establecer una comunicación vía HTTP con el servidor y obtener su respuesta.

Para ello nos creamos un cliente HTTP mediante el constructor de *DefaultHttpClient* y una petición POST instancia de la clase *HttpPost*. Para crear el *body* de la petición usaremos la clase *StringEntity*:

```
// Crear la llamada al servidor
httpClient = new DefaultHttpClient();
HttpPost postRequest = new HttpPost("http://www.webservicex.net/globalweather.asmx");
StringEntity input = new StringEntity(strRequest);
input.setContentType("application/soap+xml");
postRequest.setEntity(input);

// Tratar respuesta del servidor
HttpResponse response = httpClient.execute(postRequest);
if (response.getStatusLine().getStatusCode() != 200) {
       throw new RuntimeException("Error : Código de error HTTP : " +
       response.getStatusLine().getStatusCode());
}
```

Una posible pregunta que puede surgir tras ver este trozo de código es: ¿cómo sé cuál es el *content-type*? Esta información, originalmente está contenida en la definición del servicio, esto es, en su archivo WSDL, pero podremos averiguarlo muy fácil y rápidamente gracias a soapUI. **Tras haber ejecutado la llamada al servidor (importante este detalle)**, podremos pulsar en el botón “Raw” de la petición y ver los detalles de la llamada:



Como se puede ver en la captura, de esta forma no sólo podemos conocer el *content-type* sino otra información que también nos puede ser de utilidad como el *encoding* y el parámetro *action* por ejemplo.

6.- Interpretar los datos mediante Xpath

Si todo fue bien, el servidor nos devolverá una respuesta conforme a la definición de la misma, incluida en el WSDL, y su estructura por tanto variará entre diferentes servicios web. En este caso, como se puede ver a continuación, consistirá en un XML embebido dentro de la respuesta SOAP. Existen múltiples posibilidades para tratar dicha respuesta, como pueden ser DOM, XPath, XQuery, ... En este ejemplo usaremos XPath para extraer este XML. Posteriormente, si quisiéramos explotar dicho XML, obviamente también XPath sería de gran ayuda. La respuesta del servicio web será algo así:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetWeatherResponse xmlns="http://www.webserviceX.NET">
      <GetWeatherResult><![CDATA[<?xml version="1.0" encoding="utf-16"?>
<CurrentWeather>
 <Location>Madrid / Cuatro Vientos, Spain (LEVS) 40-23N 003-47W 687M</Location>
 <Time>Xxx nn, nnnn - nn:nn AM EDT / nnnn.nn.nn nnnn UTC</Time>
 <Wind> from the W (270 degrees) at 26 MPH (23 KT):0</Wind>
 <Visibility> 1 mile(s):0</Visibility>
 <SkyConditions> mostly cloudy</SkyConditions>
 <Temperature> 44 F (7 C)</Temperature>
 <DewPoint> 41 F (5 C)</DewPoint>
 <RelativeHumidity> 87%</RelativeHumidity>
 <Pressure> 29.80 in. Hg (1009 hPa)</Pressure>
 <Status>Success</Status>
</CurrentWeather>]]></GetWeatherResult>
    </GetWeatherResponse>
  </soap:Body>
</soap:Envelope>
```

Por tanto, usaremos XPath para obtener el contenido del nodo XML *GetWeatherResult*. Tras parsear la respuesta del servicio web, la consulta XPath más simple para esto sería "*//GetWeatherResult*":

```
//Obtener información de la respuesta
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
Document XMLDoc = factory.newDocumentBuilder().parse(response.getEntity().getContent());
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression expr = xpath.compile("//GetWeatherResult");
String result = String.class.cast(expr.evaluate(XMLDoc, XPathConstants.STRING));
```

En la variable de tipo *String "result"* tendremos el XML con la información meteorológica que hemos solicitado. Ahora podríamos explotarlo según nuestras necesidades. Lo más sencillo sería mostrarlo íntegro por pantalla, pero también podríamos usar de nuevo XPath para extraer la información que más nos interese y mostrarla, registrarla o enviarla a donde necesitemos.

7.- Últimos retoques

Para acabar, necesitaremos cerrar la conexión HTTP. Un ejemplo de código para esto sería:

```
// Cierre de la conexión
if (httpClient != null) httpClient.getConnectionManager().shutdown();
```

Y eso sería todo. Este es un ejemplo muy sencillo para hacer una primera aproximación al tema. En otros casos pueden surgir elementos más avanzados relativos a las tecnologías involucradas que requerirán un conocimiento más amplio de las mismas. Al final incluyo un listado de referencias para ampliar información sobre las tecnologías, estándares y herramientas citadas en el tutorial.

El código final para la clase podría ser el siguiente:

```
package com.dherrerabits.soapclientexample;

import java.io.StringWriter;
import java.util.HashMap;
import java.util.Map;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.w3c.dom.Document;

import freemarker.template.Configuration;
import freemarker.template.Template;
```

```
public class GlobalWeatherClient {

	public static void main(String[] args) {

		HttpClient httpClient = null;

		try {

			// Configuración Freemarker
			Configuration cfg = new Configuration();

			// Cargar plantilla
			Template template =
			cfg.getTemplate("src/main/resources/templates/template.ftl");

			// Modelo de datos
			Map<String, Object> data = new HashMap<String, Object>();
			data.put("ciudad", "Madrid");
			data.put("pais", "Spain");

			// Crear mensaje SOAP HTTP
			StringWriter out = new StringWriter();
			template.process(data, out);
			String strRequest = out.getBuffer().toString();

			// Crear la llamada al servidor
			httpClient = new DefaultHttpClient();
			HttpPost postRequest = new
			HttpPost("http://www.webservicex.net/globalweather.asmx");
			StringEntity input = new StringEntity(strRequest);
			input.setContentType("application/soap+xml");
			postRequest.setEntity(input);

			// Tratar respuesta del servidor
			HttpResponse response = httpClient.execute(postRequest);
			if (response.getStatusLine().getStatusCode() != 200) {
				throw new RuntimeException("Error : Código de error HTTP : " +
				response.getStatusLine().getStatusCode());
			}

			//Obtener información de la respuesta
			DocumentBuilderFactory factory =
			DocumentBuilderFactory.newInstance();
			Document XMLDoc =
			factory.newDocumentBuilder().parse(
			response.getEntity().getContent());
			XPath xpath = XPathFactory.newInstance().newXPath();
			XPathExpression expr = xpath.compile("//GetWeatherResult");
			String result = String.class.cast(expr.evaluate(XMLDoc,
			XPathConstants.STRING));

			System.out.println(result);

		} catch (Exception e) {
```

```
                e.printStackTrace();
            } finally {
                // Cierre de la conexión
                if (httpClient != null) httpClient.getConnectionManager().shutdown();
            }
        }
}
```

8.- Referencias

Para ampliar información sobre las tecnologías citadas, es posible consultar el estado actual de los estándares nombrados en la web del W3C a través de los siguientes enlaces:

- DOM: http://www.w3.org/standards/techs/dom#w3c_all
- HTML: http://www.w3.org/standards/techs/html#w3c_all
- HTTP: http://www.w3.org/standards/techs/http#w3c_all
- SOAP: http://www.w3.org/standards/techs/soap#w3c_all
- WSDL: http://www.w3.org/standards/techs/wsdl#w3c_all
- XML: http://www.w3.org/standards/techs/xml#w3c_all
- XPath: http://www.w3.org/standards/techs/xpath#w3c_all

De igual forma, es posible ampliar información sobre las herramientas citadas en las correspondientes webs oficiales a través de los siguientes enlaces:

- Apache HttpComponents: http://hc.apache.org/
- AXIS: http://axis.apache.org/
- CXF: http://cxf.apache.org/
- Eclipse: http://www.eclipse.org/
- Freemarker: http://freemarker.sourceforge.net/
- Java: http://www.oracle.com/technetwork/java/index.html
- Maven: http://maven.apache.org/
- Metro: http://metro.java.net/
- SoapUI: http://www.soapui.org/