CAPÍTULO 6: SOAP

Las diferentes entidades que componen nuestro proyecto necesitan poder comunicarse mediante ***SOAP*** *(Simple Object Access Protocol).* Por este motivo incluimos este capítulo donde trataremos de introducir nociones generales del tema que nos ocupa.

6.1.- Introducción

Actualmente un sin fin de empresas se han decantado por el desarrollo de aplicaciones que puedan trabajar sobre ***Internet*** porque permite la distribución global de la información. Las tecnologías más usadas para el desarrollo de estas aplicaciones, han sido hasta hace poco ***CORBA***, ***COM*** y ***EJB.*** Cada una proporciona un marco de trabajo basado en la activación de objetos remotos mediante la solicitud de ejecución de servicios de aplicación a un servidor de aplicaciones.

Estas tecnologías han demostrado ser muy efectivas para el establecimiento de sitios Web, sin embargo presentan una serie de desventajas, como son: total incompatibilidad e interoperabilidad entre ellas y dependencia de la máquina servidora sobre la que corren, así como del lenguaje de programación.

Esto ha llevado a la necesidad de considerar un nuevo modelo de computación distribuida de objetos que no sea dependiente de plataformas, modelos de desarrollo ni lenguajes de programación. Por todos estos motivos surge el concepto de ***SOAP*** *(Simple Object Access Protocol).*

6.2.- Concepto de SOAP

La funcionalidad que aporta **SOAP** es la de proporcionar un mecanismo simple y ligero de intercambio de información entre dos puntos usando el lenguaje ***XML.*** **SOAP** no es más que un mecanismo sencillo de expresar la información mediante un modelo de empaquetado de datos modular y una serie de mecanismos de codificación de datos. Esto permite que **SOAP** sea utilizado en un amplio rango de servidores de aplicaciones que trabajen mediante el modelo de comunicación ***RPC*** *(Remote Procedure Call).*

SOAP consta de tres partes:

• El ***SOAP envelope*** que define el marco de trabajo que determina **qué** se puede introducir en un mensaje, **quién** debería hacerlo y si esa operación es **opcional u obligatoria.**

• Las ***reglas de codificación SOAP*** que definen el **mecanismo de serialización** que será usado para encapsular en los mensajes los distintos tipos de datos.

• La representación **SOAP RPC** que define un modo de funcionamiento a la hora de realizar llamadas a procedimientos remotos y la obtención de sus resultados.

6.3.- Objetivos de SOAP

A la hora de realizar el diseño de **SOAP** se han tenido en cuenta una serie de consideraciones con el fin de cumplir una serie de objetivos claros, objetivos que le darán el potencial que reside en **SOAP** y que le harán tan atractivo. Éstos son:

- Establecer un protocolo estándar de invocación a servicios remotos que esté basado en protocolos estándares de uso frecuente en ***Internet***, como son ***HTTP*** *(Hiper Text Transport Protocol)* para la transmisión y ***XML*** *(eXtensible Markup Language)* para la codificación de los datos.

- Independencia de plataforma hardware, lenguaje de programación e implementación del servicio Web.

El logro de estos objetivos ha hecho de **SOAP** un protocolo extremadamente útil, ya que el protocolo de comunicación ***HTTP*** es el empleado para la conexión sobre ***Internet,*** por lo que se garantiza que cualquier cliente con un navegador estándar pueda conectarse con un servidor remoto. Además, los datos en la transmisión se empaquetan o ***serializan*** con el lenguaje ***XML***, que se ha convertido en algo imprescindible en el intercambio de datos ya que es capaz de salvar las incompatibilidades que existían en el resto de protocolos de representación de datos de la red.

Por otra parte, los servidores Web pueden procesar las peticiones de usuario empleando tecnologías tales como ***Servlets***, páginas ***ASP*** *(Active Server Pages)*, páginas ***JSP*** *(Java Server Pages)* o sencillamente un servidor de aplicaciones con invocación de objetos mediante ***CORBA, COM o EJB***.

Un ejemplo típico de diseño de un **servicio Web** utilizando las ventajas de **SOAP** podría ser el siguiente:

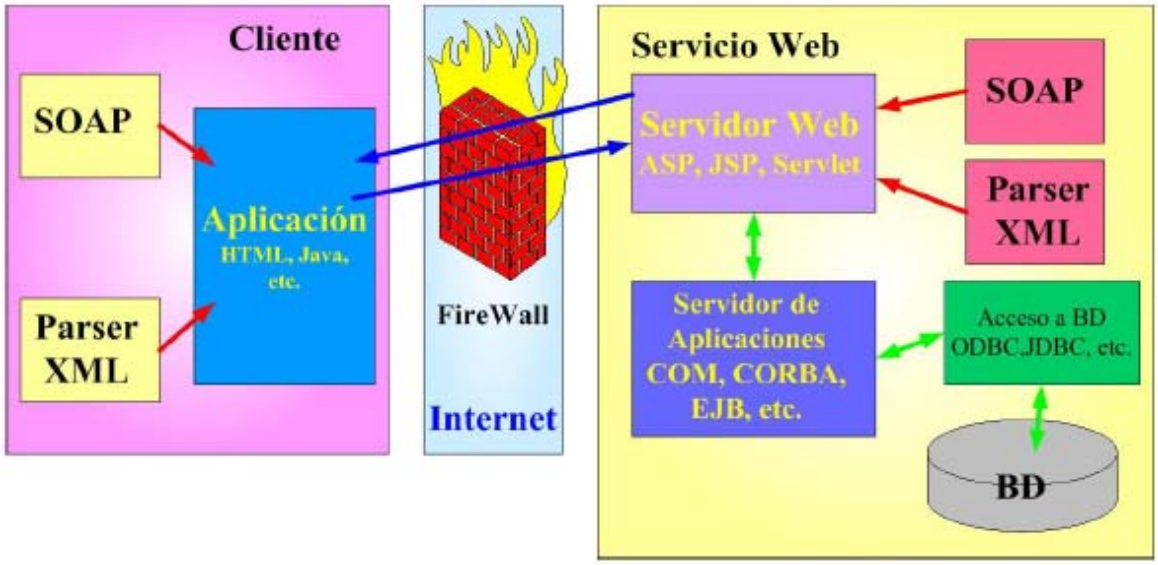

Figura 6. 1: Ejemplo de uso de SOAP

La ***especificación SOAP (SOAP Specification 1.1)*** indica que las aplicaciones deben ser independientes del lenguaje de desarrollo, por lo que las aplicaciones cliente y servidor pueden estar escritas con ***HTML, DHTML, Java, Visual Basic*** o cualquier otra herramienta o lenguaje disponibles.

6.4.- Un ejemplo sencillo de mensajes SOAP

Supongamos que tenemos un servicio Web el cual contiene una relación de productos junto con una serie de características de estos, y supongamos que le queremos hacer una consulta acerca del precio de uno de los productos cuyo código es **RHAT**. El código relativo a la petición de consulta en lenguaje **SOAP** sería:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
                    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
          SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

   <SOAP-ENV:Body>

      <getQuote xmlns="http://namespaces.cafeconleche.org/xmljava/ch2/">
            <symbol>RHAT</symbol>
      </getQuote>

   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6. 2: Ejemplo de mensaje SOAP

Como respuesta a esta petición vendría de vuelta el siguiente mensaje en el que se le dice que el precio del producto pedido es *4.12*:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
          SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

   <SOAP-ENV:Body>

        <Quote xmlns="http://namespaces.cafeconleche.org/xmljava/ch2/">
                <Price>4.12</Price>
        </Quote>

   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6. 3: Respuesta del ejemplo de mensaje SOAP

6.4.1.- Partes de un mensaje SOAP

Un mensaje **SOAP** no es más que un documento en formato **XML** que está constituido por tres partes bien definidas que son: el ***SOAP envelope,*** el ***SOAP header*** de carácter opcional y el ***SOAP body.*** Cada uno de estos elementos contiene lo siguiente:

• El ***envelope*** es el elemento más importante y de mayor jerarquía dentro del documento **XML** y representa al mensaje que lleva almacenado dicho documento.

• El ***header*** es un mecanismo genérico que se utiliza para añadir características adicionales al mensaje **SOAP**. El modo en la que se añadan cada uno de los campos dependerá exclusivamente del servicio implementado entre cliente y servidor, de forma que cliente y servidor deberán estar de acuerdo con la jerarquía con la que se hayan añadido los distintos campos. De esta forma será sencillo separar entre sí los distintos datos a transmitir dentro del mensaje.

• El ***body*** es un contenedor de información en el cual se almacenarán los datos que se quieran transmitir de lado a lado de la comunicación. Dentro de este campo, **SOAP** define un elemento de uso opcional denominado ***Fault*** utilizado en los mensajes de respuesta para indicar al cliente algún error ocurrido en el servidor.

Un ejemplo de uso del ***header***, donde se indica un cierto parámetro útil para el servicio Web que le indica como debe procesar el mensaje sería:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP- ENV:encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/">

        <SOAP-ENV:Header>
                <t:Transaction   xmlns:t="some-URI"   SOAP-ENV:mustUnderstand="1">
                        5
                </t:Transaction>
        </SOAP-ENV:Header>

        <SOAP-ENV:Body>
                <getQuote xmlns= "http://namespaces.cafeconleche.org/xmljava/ch2/">
                        <symbol>RHAT</symbol>
                </getQuote>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6. 4: Ejemplo del uso de **header** dentro de un mensaje SOAP

Un ejemplo de uso del nuevo elemento ***body*** sería el siguiente, en el que se ha detectado un error en la aplicación que corre sobre el servidor que provoca que no opere convenientemente, por lo que se le indica al cliente:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/">

        <SOAP-ENV:Body>

                <SOAP-ENV:Fault>
                        <faultcode>SOAP-ENV:Server</faultcode>

                        <faultstring>Server Error</faultstring>

                </SOAP-ENV:Fault>

        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6. 5: Ejemplo de **body** que contiene elemento falta **Fault**

El atributo *encodingStyle* se utiliza para indicar las reglas de *serialización* utilizadas en el mensaje **SOAP**. No existe un formato de codificación por defecto, sino que existen una serie de posibles formatos a utilizar. El valor de este atributo es una lista ordenada de una o más ***URIs*** que identifican la regla o reglas de serialización que pueden ser utilizadas en el mensaje, en el orden en el que se han de aplicar. De entre todas las posibles, las más utilizadas son:

http://schemas.xmlsoap.org/soap/encoding/ y
http://my.host/encoding/restricted http://my.host/encoding/.

Todo mensaje **SOAP** debe tener un elemento ***Envelope*** asociado a la dirección de red *http://schemas.xmlsoap.org/soap/envelope/*. Si un mensaje recibido

por una aplicación **SOAP** contiene en este elemento un valor distinto al anterior la aplicación trataría dicho mensaje como erróneo.

6.5.- Serialización

A la hora de introducir los datos en un mensaje **SOAP** existen una serie de normas a tener en cuenta. De esta forma **SOAP** define una serie de tipos básicos que serán empaquetados de forma directa y una serie de mecanismos para empaquetar tipos complejos y estructurados formados por elementos simples.

En un inicio, **SOAP** consideró un conjunto de tipos como tipos simples con el fin de realizar un *mapeo* directo entre el propio documento **SOAP** y tipos básicos de **Java.** Ahora bien, actualmente este conjunto ha aumentado notablemente de forma que existe un número bastante grande de tipos que gozan de esta situación. De entre todos ellos destacan:

Tipo SOAP	Tipo Java
SOAP-ENC:int	java.lang.Integer
SOAP-ENC:long	java.lang.Long
SOAP-ENC:short	java.lang.Short
SOAP-ENC:string	java.lang.String
SOAP-ENC:boolean	java.lang.Boolean
SOAP-ENC:float	java.lang.Float
SOAP-ENC:double	java.lang.Double
SOAP-ENC:byte	java.lang.Byte

Figura 6. 6: Mapeo entre tipos SOAP y JAVA

Además de todos estos tipos sencillos, **SOAP** implementa una serie de mecanismos para *serializar* tipos complejos con elementos simples en su interior. De esta forma tenemos:

- ***Structs***

Un ***struct*** no es más que un elemento que contiene un conjunto de elementos *hijos* almacenados cada uno de ellos en un campo propio. Un ejemplo podría ser:

```
<Quote xmlns="http://namespaces.cafeconleche.org/xmljava/ch2/">
        <Symbol>RHAT</Symbol>
        <Price>4.12</Price>
</Quote>
```

Figura 6. 7: Realización de **Structs** mediante SOAP

En términos de **Java** tendríamos un objeto de tipo ***Quote*** dentro del cual hay dos elementos sencillos: uno de tipo *String* llamado ***Symbol*** y el otro de tipo *double* y de nombre ***Price.*** Es decir:

```
public class Quote {

        public String Symbol();
        public double Price();
}
```

Figura 6. 8: Realización de **Structs** mediante JAVA

• ***Arrays***

Un ***array*** en un mensaje **SOAP** es representado mediante un elemento cuyo tipo es ***SOAP-ENC:Array.***

Aunque en Java sea obligatorio que dentro de un array haya únicamente elementos del mismo tipo, SOAP no presenta esta restricción, sino que es posible albergar elementos de distintos tipos, así un ejemplo sería:

```
<Bid xsi:type="SOAP-ENC:Array">
        <Symbol xsi:type="xsd:token">RHAT</Symbol>
        <Price xsi:type="xsd:double">4.12</Price>
        <Account xsi:type="xsd:string">777-7777</Account>
</Bid>
```

Figura 6. 9: Realización de **Array** de tipos variados mediante SOAP

Si quisiésemos restringir explícitamente el tipo de los elementos almacenados en el ***array*** podríamos hacerlo de la forma:

```
<Bid xsi:type="SOAP-ENC:Array"      SOAP-ENC:arrayType="xsd:double[3]">
        <Price>4.52</Price>
        <Price>0.35</Price>
        <Price>34.68</Price>
</Bid>
```

Figura 6. 10: Realización de **Array** del mismo tipo mediante SOAP

En **SOAP** también es posible implementar ***arrays*** bidimensionales, incluso ***arrays*** que almacenan *strings.* Un ejemplo de esto sería:

```
<SOAP-ENC:Array     SOAP-ENC:arrayType="xsd:double[3,2]">
        <SOAP-ENC:double>1.1</SOAP-ENC:double>
        <SOAP-ENC:double>1.2</SOAP-ENC:double>
        <SOAP-ENC:double>2.1</SOAP-ENC:double>
        <SOAP-ENC:double>2.2</SOAP-ENC:double>
        <SOAP-ENC:double>3.1</SOAP-ENC:double>
        <SOAP-ENC:double>3.2</SOAP-ENC:double>
</SOAP-ENC:Array>
```

Figura 6. 11: Realización de Array bidimensional mediante SOAP

Este mismo Array que hemos explicado su realización en **SOAP**, en tecnología **Java** sería de la manera:

```
double[][] array = {
        {1.1, 1.2},
        {2.1, 2.2},
        {3.1, 3.2}
}
```

Figura 6. 12: Realización de un Array bidimensional mediante Java

O un ejemplo algo más complicado sería un A*rray* que contiene a su vez dos *arrays* cada uno de los cuales contiene a su vez un A*rray* de S*trings:*

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[][2]">
        <item href="#array-1"/>
        <item href="#array-2"/>
</SOAP-ENC:Array>

<SOAP-ENC:Array id="array-1" SOAP-ENC:arrayType="xsd:string[2]">
        <item>r1c1</item>
        <item>r1c2</item>
        <item>r1c3</item>
</SOAP-ENC:Array>

<SOAP-ENC:Array id="array-2"    SOAP-ENC:arrayType="xsd:string[2]">
        <item>r2c1</item>
        <item>r2c2</item>
</SOAP-ENC:Array>
```

Figura 6. 13: Realización de un **Array** de **Arrays**

6.6.- Protocolo HTTP mediante SOAP

Una de las funcionalidades con las que cuenta **SOAP** y que lo hace extremadamente atractivo es el envío de mensajes **SOAP** vía protocolo **HTTP,** todo ello realizado de forma directa por medio de las librerías de **SOAP.**

Este hecho provoca que existen ciertas diferencias entre un mensaje **SOAP** normal y uno enviado haciendo uso del protocolo **HTTP.** De esta forma deben cumplirse dos aspectos:

• El ***HTTP header*** del mensaje de petición al servicio Web debe contener un campo ***SOAPAction.*** El campo ***SOAPAction*** alerta a los *servidores Web* y *firewalls* por los que pase de que el mensaje contiene documento **SOAP,** esto facilita a dichos *firewalls* el filtrado de las peticiones **SOAP** sin necesidad de tener que explorar el contenido del ***body*** del mensaje.

• Si la petición **SOAP** al servicio Web falla, el servidor debe devolver en el mensaje de respuesta un error del tipo **HTTP 500 Internal Server Error** en vez de un **200 OK** que se envía cuando todo ha ido bien.

Un ejemplo de todo esto sería el siguiente extracto de código en el que lanza una petición **HTTP** de tipo **POST** contra un ***Servlet*** que corre en la dirección Web *www.ibiblio.org* bajo el control del usuario de nombre *elharo:*

```
POST /xml/cgi-bin/SOAPHandler HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: 267
SOAPAction: "http://www.ibiblio.org/#elharo"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >

        <SOAP-ENV:Body>
                <getQuote xmlns="http://namespaces.cafeconleche.org/xmljava/ch2/">
                        <symbol>RHAT</symbol>
                </getQuote>
        </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Figura 6. 14: Ejemplo de petición **SOAP sobre HTTP**

El servidor envía la respuesta de vuelta al cliente y después cierra la conexión. Como cualquier otra respuesta **HTTP** el mensaje **SOAP** empieza con el **código de retorno HTTP.** Suponiendo que la petición tuvo éxito y no ocurrió ningún error en el servidor, el mensaje de respuesta sería:

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 260

<?xml version="1.0"?>
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

        <SOAP-ENV:Body>
                <Quote xmlns="http://namespaces.cafeconleche.org/xmljava/ch2/">
                        <Price>4.12</Price>
                </Quote>
        </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Figura 6. 15: Ejemplo de respuesta **SOAP sobre HTTP**