# Servicios Web

Miguel Angel Astor Romero

Redes de Computadoras - Tarea 1 - Entrega 1

## Introducción

En el presente documento se presenta una introducción a los servicios Web vistos como uno de los mecanismos de comunicación más comunmente utilizados para transmitir información a travéz de la Web. El principal interés de esta introducción es de caracter técnico, y se enfoca principalmente en el estudio de la arquitectura y los protocolos utilizados para definir y ejecutar servicios Web.

El resto de este documento está estructurado en tres secciones más un anexo. La Sección 1 presenta la definición formal de servicio Web dada por la W3C. La Sección 2 describe el modo de funcionamiento general de un servicio Web, haciendo especial énfasis en detallar los entes que participan en la comunicación, y la secuencia de pasos que siguen estos para poder llevar a cabo la ejecución del servicio Web. En la Sección 3 se estudian a detalle los protocolos SOAP, XML-RPC y JSON-RPC, comunmente utilizados para definir servicios Web. Finalmente en el Anexo A se propone una posible solución para el ejercicio indicado como segunda parte de la asignación que compete a este documento.

## 1. Definición de servicio Web

Según la W3C (*World Wide Web Consortium* - Consorcio de la *World Wide Web*) un servicio Web se define como "*un sistema de* software *diseñado para soportar interacciones interoperables máquina-a-máquina sobre una red. Este posee una interfaz descrita en un formato procesable por máquinas (específicamente* WSDL*). Otros sistemas interactúan con el servicio Web de la forma prescrita por su descripción utilizando mensajes* SOAP*, típicamente entregados por* HTTP *con serialización en* XML *en conjunción con otros estándares relacionados a la Web.*" [1].

En otras palabras, un servicio Web es un mecanismo que utiliza los estándares de la Web para comunicar sistemas de cómputo. Los servicios Web utilizan fundamentalmente el protocolo SOAP como mecanismo de comunicación, el cual se apoya principalmente en HTTP (*Hypertext Transfer Protocol* - Protocolo de Transferencia de Hipertexto) como mecanismo de transporte y en el lenguaje XML (*Extensible Markup Language* - Lenguaje de Marcado Extendible) como formato de representación de los datos que se van a comunicar entre

los distintos sistemas. Sin embargo, como se verá más adelante, en la práctica un servicio Web es sencillamente un mecanismo de comunicación entre máquinas que se basa en las tecnologías de la Web, independientemente de si el protocolo de comunicación es SOAP o no, de si el transporte es HTTP u otro protocolo de capa de aplicación de Internet (como por ejemplo FTP (*File Transfer Protocol* - Protocolo de Transferencia de Archivos) o SMTP (*Simple Mail Transfer Protocol* - Protocolo Simple de Transferencia de Correo)) e incluso independientemente del lenguaje de representación de los datos.

# 2. Arquitectura de los servicios Web

Los servicios Web representan un mecanismo de comunicación complejo que involucran múltiples tecnologías, y como tales debe seguirse una serie de criterios para poder utilizarlos apropiadamente. Como con todo mecanismo de comunicación, primero hace falta establecer quienes serán los entes que llevarán a cabo dicha comunicación, seguido por los mensajes que intercambiarán dichos entes, así como también es necesario establecer cual será la semántica del intercambio de dichos mensajes. En esta sección se describen las definiciones y recomendaciones de la W3C con respecto a estos elementos de los servicios Web.

## 2.1. Agente solicitante y agente proveedor

En todo servicio Web deben existir por lo menos dos entes computacionales que son los que llevarán a cabo la comunicación. El primero de estos son los llamados agentes proveedores, encargados de proveer la descripción del servicio y de establecer la semántica del mismo, así como por supuesto el ejecutar el servicio en sí. El otro tipo de ente computacional involucrado con los servicios Web son los agentes solicitantes, los cuales se encargan de utilizar los servicios de los agentes proveedores. Estos agentes son utilizados por los usuarios humanos que serán los beneficiarios finales de la comunicación. En el caso de los agentes proveedores estos son utilizados por personas o empresas conocidas como proveedores del servicio, mientras que los agentes solicitantes son utilizados por personas o empresas conocidas como usuarios del servicio.

Ambos tipos de agentes deben ser configurados y/o utilizados por usuarios humanos. En el caso de los agentes proveedores, los administradores del sistema deberían definir uno o más documentos que en conjunto describan las interfaces del servicio y la semántica de uso de dichas interfaces. Los documentos de descripción del servicio y los lenguajes utilizados para redactarlos se detallan en la Sección 2.2.

Los agentes solicitantes son cualquier sistema de software que posean la capacidad de comunicarse con los agentes proveedores para poder realizar peticiones de servicio y obtener respuestas del mismo. El uso de protocolos y tecnologías Web permite que estos agentes sean capaces de comunicarse con los agentes proveedores de los servicios independientemente del sistema operativo sobre el que se ejecuten, o del lenguaje de programación en el cual estén implementados.

## 2.2. Descripción del servicio

Para poder llevar a cabo la comunicación entre los agentes mencionados es necesario que estos estén de acuerdo sobre la especificación de la comunicación, específicamente, deben estar de acuerdo sobre la interfaz de los servicios que el agente proveedor provee, valga la redundancia, y sobre la semántica que de uso que debe cumplir el agente solicitante al utilizar el servicio. Esta descripción se provee como un documento, usualmente escrito en el lenguaje WSDL (*Web Services Description Language* - Lenguaje de descripción de Servicios Web), basado en XML. En otras palabras, según la W3C [1], la descripción de un servicio es un documento hecho para ser procesado por máquinas que especifíca las capacidades de un servicio y su interfaz, incluyendo la especificación de todos los tipos de mensajes que se pueden intercambiar durante el uso del servicio y la semántica de su uso.

## 2.3. Semántica del servicio

Se entiende por semántica del servicio a las expectativas que deben tener los agentes solicitantes con respecto al comportamiento del proveedor del servicio. En otras palabras, es un contrato que establece el significado y el propósito de las interacciones entre los agentes involucrados en el uso del servicio [1]. Este contrato puede ser explícito (definido en la descripción del servicio) o implícito, así como también puede ser un contrato legal formal (vinculante) o un contrato informal sin base legal.

## 2.4. Arquitectura general del uso de un servicio Web

Para utilizar un servicio Web se suelen seguir los siguientes cuatro (4) pasos, algunos de los cuales pueden ser automáticos mientras otros pueden realizarse de forma manual por operadores humanos [1]:

1. El ente solicitante y el ente proveedor se descubren de alguna manera. Para esto se necesita seguir los siguientes pasos:
   a) El ente proveedor debe cargar la descripción del servicio en el sistema de descubrimiento de servicio.
   b) El ente solicitante identifica el servicio del ente proveedor utilizando un criterio de búsqueda en el sistema de descubrimiento de servicio.
   c) El sistema de descubrimiento de servicio envía la descripción del servicio al ente solicitante.
2. El solicitante y el proveedor establecen un acuerdo sobre la semántica del servicio.
3. La descripción y la semántica del servicio son procesada como entradas por el software ejecutado por el agente solicitante y el agente proveedor.

4. Ambos programas comienzan el intercambio de mensajes posiblemente utilizando el protocolo SOAP, aunque en este punto es posible utilizar otros protocolos como se verá en la Sección 3.

Este procedimiento puede verse en la figura 1.

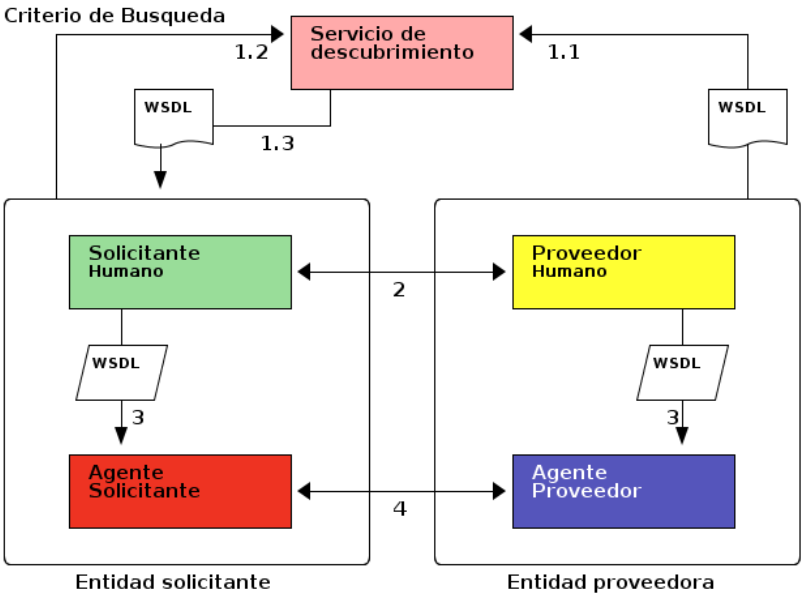


Figura 1: Arquitectura del uso de un servicio Web. Figura reconstruida de [1].

# 3. Protocolos para servicios Web

En esta Sección se describen tres de los protocolos más utilizados para definir e implementar servicios Web. Concretamente se describen y especifican los protocolos SOAP (promovido por la W3C), XML-RPC y JSON-RPC. Al final de la Sección se listan algunos protocolos adicionales.

## 3.1. SOAP

SOAP es un protocolo de comunicación basado en XML *Infoset* (*Information Set* - Conjunto de Información) y utilizado para intercambiar información estructurada entre pares en entornos distribuidos y descentralizados. Originalmente el nombre SOAP era un acrónimo de *Simple Object Access Protocol* (Protocolo Simple de Acceso a Objetos), sin embargo, desde la versión 1.2 del protocolo se utiliza SOAP directamente como un sustantivo sin un significado particular [2].

SOAP fue definido originalmente en 1998 en Microsoft por Dave Winer y Don Box [3] debido a problemas de ingeniería encontrados por los grupos de trabajo de dicha empresa encargados de portar el sistema de objetos distribuido DCOM (*Distributed Component Object Model* - Modelo de Objetos de Componentes Distribuidos) a sistemas operativos UNIX [4]. SOAP fue diseñando para proveer un mecanismo de intercambio de objetos basado en XML y utilizando un mecanismo de tipos fuerte, que pudiera funcionar de manera independiente del sistema operativo a través de Internet. Actualemente la especificación 1.2 del protocolo es una recomendación de la W3C, lo que hace a SOAP un estandar de la Web [2].

SOAP utiliza un paradigma de comunicación tipo solicitud-respuesta sin estados, el cual puede extenderse para formar conversaciones. En una conversación SOAP una solicitud proveniente de un nodo $A$ a un nodo $B$ producirá una respuesta del nodo $B$ la cual puede ser seguida por otra respuesta proveniente de $A$ a la cual $B$ responderá también, y así sucesivamente. De igual forma, una solicitud puede recivir múltiples respuestas sucecivas. SOAP tambien puede ser utilizado como un mecanismo de RPC (*Remote Procedure Call* - Llamadas a Procedimientos Remotos).

### 3.1.1. Entes involucrados en la comunicación

Todo ente que participa en una comunicación utilizando el protocolo SOAP es conocido como un nodo SOAP. Se distinguen los siguientes tipos de nodos SOAP:

1. Un SOAP *Receiver* (receptor SOAP) es cualquier nodo capaz de recibir mensajes.
2. Un SOAP *sender* (emisor SOAP) es cualquier nodo capaz de enviar mensajes a receptores SOAP.
3. El *initial* SOAP *sender* (emisor inicial SOAP) es un emisor SOAP que inicia una conversación SOAP particular.
4. El *ultimate SOAP receiver* (receptor SOAP último) es un receptor SOAP que actúa como destinatario final de una conversación
5. Un SOAP *intermediary* (intermediario SOAP) es un nodo que actua simultaneamente como receptor y emisor SOAP entre un par de nodos emisor inicial y receptor último.

Se define como el SOAP *message path* al conjunto de todos los nodos por los que pasa un mensaje SOAP particular entre el emisor inicial y el receptor último.

Adicional al tipo, todo nodo SOAP posee adicionalmente un rol que define como se comporta este al recibir mensajes SOAP.

#### Roles de los nodos SOAP

SOAP 1.2 define 3 roles que pueden verse en la Tabla 1. Los nodos que funcionen en el rol `next` actuan como nodos intermediarios entre el nodo iniciador y el nodo receptor, el cual debe actuar en el rol `ultimateReceiver`. Todo nodo SOAP debe estar en capacidad

Tabla 1: Roles de SOAP 1.2.

| Rol | URL |
|---|---|
| next | "http://www.w3.org/2003/05/soap-envelope/role/next" |
| none | "http://www.w3.org/2003/05/soap-envelope/role/none" |
| ultimateReceiver | "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver" |

de actuar en el rol `next`. El rol `none` se utiliza para marcar un bloque de cabecera como no procesable, dado que ningún nodo SOAP debe actuar en este rol.

Fuera de estos atributos, la especificación de SOAP no define bloques de cabecera en particular, ni los espacios de nombre que deben usar estos. Esto da flexibilidad absoluta a las aplicaciones basadas en SOAP para definir y utilizar los bloques de cabecera que necesiten. Igualemente la especificación 1.2 de SOAP no define métodos específicos para establecer el rol de los nodos SOAP o como cambiar dichos roles, detalles que son de libre implementación por las aplicaciones basadas en SOAP.

### 3.1.2. Estructura de los mensajes SOAP

Un mensaje SOAP se define como un XML *Infoset* basado en el espacio de nombres SOAP-ENV, el cual posee la siguiente estructura general:

```
<?xml version="1.0"?>
<env:Envelope
 xmlns:env="http://www.w3.org/2001/12/soap-envelope"
 env:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

   <env:Header>
      ...
   </env:Header>

   <env:Body>
      ...
      <env:Fault>
         ...
      </env:Fault>
      ...
   </env:Body>

</env:Envelope>
```

Como se puede observar, todo mensaje SOAP se compone de los siguientes elementos:

***Envelope*** **(envoltorio)**

Define los límites del mensaje, el espacio de nombres del mismo y la codificación utilizada.

### *Header* (cabecera)

Contiene atributos y propiedades generales del mensaje para ser utilizadas por los nodos involucrados en la comunicación, sean nodos finales o nodos intermedos. Este elemento es opcional. Un mensaje SOAP puede contener a lo sumo un elemento de cabecera, y este debe aparecer siempre antes que el cuerpo del mensaje.

Internamente el elemento cabecera puede contener cero o más bloques de cabecera, los cuales representan agrupaciones lógicas de propiedades, las cuales pueden estar dirigidas a nodos específicos de la comunicación que no necesariamente tienen que ser los nodos finales.

Un ejemplo de una cabecera SOAP es la siguiente (ejemplo tomado de [2]).

```
<env:Header>
  <m:reservation
       xmlns:m="http://travelcompany.example.org/reservation"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger
       xmlns:n="http://mycompany.example.com/employees"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       env:mustUnderstand="true">
    <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
</env:Header>
```

Este ejemplo representa las propiedades utilizadas por un sistema hipotético de reservaciones de viajes. Incluye dos bloques de cabecera (`m:reservation` y `n:passenger` respectivamente. Como se puede observar, estos bloques utilizan dos atributos definidos en el espacio de nombres de SOAP, los cuales son opcionales según la especificación:

1. `role` (rol) especifica con un URL (*Uniform Resource Locator* - Localizador Uniforme de Recursos) el rol en el cual deben operar los nodos que deben procesar el bloque de cabecera respectivo. Los posibles roles se definen en la Tabla. Un nodo que no este operando en el rol especificado debe ignorar el contenido del bloque de cabecera. Los valores válidos para este atributo pueden verse en la columna URL de la Tabla 1.

2. `mustUnderstand` es un atributo booleano que indica que el nodo que reciba esta cabecera debe ser capaz de interpretarla y procesarla de forma obligatoria. En caso de que el nodo que este procesando la cabecera no pueda procesar algún bloque marcado como `mustUnderstand`, entonces dicho nodo debe dejar de procesar el mensaje y enviar un mensaje de error al remitente.

Si un bloque de cabecera no posee el atributo rol, se asume que este va dirigido al nodo que asuma el rol de `ultimateReceiver`.

### *Body* (cuerpo)

Este elemento contiene todos los datos del mensaje y debe aparecer de forma obligatoria dentro del envoltorio, siempre después de la cabecera si es que esta está presente. Este campo es procesado únicamente por el nodo que se encuentre actuando en el rol `ultimateReceiver` definido en la Tabla 1, o por algún nodo intermediario arbitrario que decida asumir dicho rol después de procesar la cabecera del mensaje.

El cuerpo de un mensaje SOAP se compone de cero o más elementos hijos los cuales pueden ser de libre implementación. El único requisito particular de la especificación 1.2 de SOAP con respecto a los hijos del cuerpo del mensaje es que estos deben pertenecer a un espacio de nombres calificado de algún XML *infoset*. Un ejemplo de un cuerpo de mensaje SOAP es el siguiente (igualmente tomado de [2]):

```
<env:Body>
  <p:itineraryClarification
    xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>
        <p:airportChoices>
          JFK LGA EWR
        </p:airportChoices>
      </p:departing>
    </p:departure>
    <p:return>
      <p:arriving>
        <p:airportChoices>
          JFK LGA EWR
        </p:airportChoices>
      </p:arriving>
    </p:return>
  </p:itineraryClarification>
</env:Body>
```

### *Fault* (falla)

El elemento de falla es el único elemento hijo del cuerpo del mensaje especificado formalmente en el protocolo SOAP. Este elemento se utiliza para reportar errores en el procesamiento de los mensajes de forma estandarizada.

Los elementos de falla pueden poseer los siguientes nodos hijos:

1. `Code` denota el error ocurrido utilizando valores definidos por la especificación de SOAP. Este elemento es obligatorio en toda falla.

2. `Reason` incluye mensajes de descripción del error para ser interpretados por operadores humanos del sistema. El campo `Reason` puede incluir múltiples traducciones de la descripción en distintos idiomas. Este elemento es obligatorio en toda falla.

3. `Detail` se utiliza para incluir información detallada de los errores ocurridos. Este campo es de libre implementación y es opcional.

4. `Node` identifica mediante un URI (*Uniform Resource Identifier* - Identificador Uniforme de Recursos) cual nodo SOAP en el tránsito del mensaje produjo el error. Este campo es opcional.

5. `Role` indica en que rol de los definidos en la Tabla 1 se encontraba el nodo SOAP que produjo la falla al momento de ocurrirse esta.

Un ejemplo de un campo de falla es el siguiente (tomado de [2]):

```
<env:Fault>
  <env:Code>
    <env:Value>env:Sender</env:Value>
    <env:Subcode>
      <env:Value>rpc:BadArguments</env:Value>
    </env:Subcode>
  </env:Code>
  <env:Reason>
    <env:Text xml:lang="en-US">Processing error</env:Text>
    <env:Text xml:lang="cs">Chyba zpracování</env:Text>
  </env:Reason>
  <env:Detail>
    <e:myFaultDetails
     xmlns:e="http://travelcompany.example.org/faults">
      <e:message>Name does not match card number</e:message>
      <e:errorcode>999</e:errorcode>
    </e:myFaultDetails>
  </env:Detail>
</env:Fault>
```

### 3.1.3. Procesamiento de los mensajes SOAP

Todo nodo SOAP debe seguir los siguientes pasos al recibir un mensaje:

1. Determinar los roles que debe asumir el nodo. Se puede inspeccionar todo el contenido del envoltorio del mensaje para poder apoyar la toma de esta decisión.

2. Identificar todos los bloques de cabecera obligatorios que se deben procesar según el rol seleccionado.

3. Si el nodo no puede interpretar aunque sea uno de los bloques de cabecera obligatorios destinados a él, se debe enviar una respuesta de falla.

4. Procesar todos los bloques de cabecera obligatorios. En el caso del nodo `ultimateReceiver`, se debe procesar el cuerpo del mensaje en esta etapa. También se pueden procesar los bloques de cabecera no obligatorios en esta etapa. Si ocurre algún error en el procesamiento se deben enviar los mensajes de error correpondientes.

5. Si el nodo es intermediario entonces se debe redirigir el resultado del procesamiento (si no ocurrió una falla) al siguiente nodo en el camino hacia el nodo `ultimateReceiver`.

La ejecución de estos pasos puede implementarse de forma concurrente o paralela, siempre y cuando se garantize que el resultado de la ejecución sea equivalente a ejecutarlos en el orden dado.

### 3.1.4. Serialización y transporte de mensajes SOAP

Por serialización se entiende el procedimiento realizado para convertir un cierto dato a ser transmitido dado en alguna representación particular a una forma canónica que puede ser transmitida por la red y que es entendida por los entes involucrados en la comunicación. Para el caso particular de SOAP, forma canónica utilizada no está definida como parte de la especificación del protocolo. Usualmente se utiliza el lenguaje XML como lenguaje de representación para dicha forma canónica [1], aunque es posible utilizar otras según las necesidad de las aplicaciones, como por ejemplo CSV (*Comma Separated Values* - Valores Separados por Comas) o JSON (*Javascript Object Notation* - Notación de Objetos de Javascript) aunque esto es poco común [2].

SOAP es independiente del protocolo de transporte utilizado para enviar los mensajes, siendo los protocolos más comunes HTTP, SMTP, FTP, TCP (*Transmission Control Protocol* - Protocolo de Control de Transmisiones), UDP (*User Datagram Protocol* - Protocolo de Datagramas de Usuario) y BEEP (*Blocks Extensible Exchange Protocol* - Protocolo de Intercambio Extendible por Bloques) [2].

## 3.2. XML-RPC y JSON-RPC

XML-RPC [1] y JSON-RPC[2] son protocolos para realizar comunicación entre procesos según el paradigma RPC, utilizando XML o JSON respectivamente como lenguajes de representación de las solicitudes y respuestas. Estos protocolos permiten a un proceso el solicitar la ejecución de un procedimiento implementado en otro proceso distinto, independientemente de los lenguajes de programación con los cuales hayan sido implementados ambos procesos, o incluso independientemente de los sistemas operativos en los cuales se estén ejecutando. Esto contrasta con mecanismos similares como es el caso de RMI (*Remote Method Invocation*

[1] http://xmlrpc.scripting.com/
[2] http://json-rpc.org/

- Invocación de Métodos Remotos) del lenguaje de programación Java, el cual necesita que todos los entes involucrados en la comunicación hayan sido desarrollados en el lenguaje Java con versiones compatibles de RMI.

### 3.2.1. XML-RPC

XML-RPC es un protocolo que permite realizar llamadas a procedimientos remotos entre procesos escritos en lenguajes heterogéneos y ejecutándose sobre sistemas heterogéneos utilizando el lenguaje XML como el mecanismo de representación de datos y HTTP como protocolo de transporte [5]. Fue diseñado en 1999 por Dave Winer debido a la reticencia de Microsoft en apoyar la estandarización abierta del protocolo SOAP [3]. XML-RPC es un subconjunto del sistema de tipos de SOAP, el cual utiliza un mecanismo de mensajes solicitud-respuesta caracterizado por su brevedad de mensajes en comparación con SOAP.

Los mensajes de XML-RPC son enviados por los clientes en solicitudes HTTP POST. Los servidores responden a la solicitud enviando el valor de retorno y el estado de la ejecución dentro de un documento XML. Este protocolo no utiliza un lenguaje ni documentos de descripción del servicio.

Un ejemplo de una solicitud XML-RPC es la siguiente:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181



<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Al comienzo del ejemplo se puede observar la cabecera de HTTP utilizada. El URI de la solicitud se indica como /RPC2 para indicar al servidor que la solicitud se debe resolver utilizando el mecanismo de XML-RPC, en lugar del resolvedor normal utilizado para páginas web. Este mecanismo es opcional. Puede observarse también que el contenido de la solicitud es de tipo `text/xml`. Después de la cabecera HTML esta el cuerpo de la solicitud. Esta comienza con el elemento raíz `methodCall` el cual contiene un sub-elemento `methodName`

el cual contiene como valor al nombre del método a ejecutar. Posteriormente está un sub-elemento `params` el cual contiene múltiples elementos `param`, uno por cada parámetro recibido por el método a invocar. Los tipos de los parámetros pueden ser valores escalares, cadenas, fechas y estructuras de datos complejas como registros y listas de elementos, entre otros. El servidor tiene libertad de definir como debe ser interpretado el nombre del método a ejecutar. La Tabla 2 lista todos los tipos de parámetros válidos en XML-RPC. Si un parámetro no está contenido en un elemento de tipo, se asume que el tipo de dicho parámetro es `string`.

Tabla 2: Tipos elementales en XML-RPC. Tabla recuperada de [5]

| **Etiqueta** | **Tipo** | **Ejemplo** |
|---|---|---|
| <i4>o <int> | Entero con signo de 4 bytes | -12 |
| <boolean> | 0 (falso) ó 1 (verdadero) | 1 |
| <string> | Cadena de caracteres UNICODE [3] | Hola, mundo! |
| <double> | Punto flotante de doble precisión [4] | -12.214 |
| <dateTime.iso8601> | Fecha y hora | 19980717T14:08:55 |
| <base64> | Binario codificado en base64 | SGVsbG8sIGJhc2U2NCE= |

Es posible definir registros y arreglos como tipos de datos utilizando la siguiente sintaxis. Los tipos de los valores contenidos en los registros y arreglos pueden ser cualquiera de los tipos elementales de la Tabla 2, o incluso otros registros o arreglos. Los arreglos pueden contener datos de tipos heterogéneos.

```
<struct>
  <member>
    <name>lowerBound</name>
    <value><i4>18</i4></value>
  </member>
  <member>
    <name>upperBound</name>
    <value><i4>139</i4></value>
  </member>
</struct>

<array>
  <data>
    <value><i4>12</i4></value>
    <value><string>Egypt</string></value>
    <value><boolean>0</boolean></value>
    <value><i4>-31</i4></value>
```

[3]La especificación de XML-RPC no define una codificación de caracteres específica para las cadenas de caracteres.

[4]No incluye representaciones para el infinito positivo o negativo, o para el valor NaN.

```
  </data>
</array>
```

A cada solicitud le debe seguir una respuesta que debe seguir el formato visible en el siguiente ejemplo.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT



<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

El código de respuesta de HTTP siempre debe ser `200 OK`. Toda respuesta debe estar contenida dentro de un elemento raíz `methodResponse` el cual debe contener un sub-elemento `params` que a su vez contendrá únicamente un elemento `param`. Este `param` contendrá el valor de retorno del método invocado, el cual sigue las mismas reglas con respecto al tipo de dato utilizadas en la solicitud de ejecución. Alternativamente, una respuesta puede contener un elemento `fault` en lugar del elemento `params` que indica al solicitante que ocurrió un error en la invocación del método. Un ejemplo de una respuesta tipo `fault` se ve en el siguiente ejemplo.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT



<?xml version="1.0"?>
```

```
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

De la misma forma que en el caso de las respuestas normales, las respuestas de error deben utilizar el código de retorno `200 OK` de HTTP. Toda respuesta de error contendrá un elemento `fault` como contenido del `methodResponse`. Este elemento `fault` a su vez contendrá un único valor de tipo registro, el cual contendrá dos campos: el primero es un entero que contendrá un código que indique el error ocurrido, y el segundo campo será una cadena de caracteres que describa al error.

### 3.2.2. JSON-RPC

JSON-RPC esta basado directamente en XML-RPC y por lo tanto es muy similar en su diseño y funcionamiento. Se distinguen dos versiones del protocolo, la 1.0 y la 2.0.

#### JSON-RPC 1.0

Este protocolo define tres tipos de mensaje [6]:

**Solicitud** Este mensaje es enviado por el agente solicitante al agente proveedor para iniciar la invocación de un método. Consiste en un único objeto JSON que posee las siguientes propiedades:

***method*** Un *string* que contiene el nombre del método a ejecutar.

***params*** Un arreglo de objetos que deben ser pasados como parámetros del método a invocar.

***id*** Un identificador de la llamada. Puede ser cualquier valor de cualquier tipo.

**Respuesta** Este mensaje contiene el resultado de la invocación de un método y es enviado por el agente proveedor al agente solicitante. Este mensaje se envía únicamente como respuesta a un mensaje de solicitud. Consiste igualmente en un único objeto JSON con las siguientes propiedades:

***result*** El valor de retorno del método como un objeto. Debe ser nulo si hubo un error al invocar el método.

***error*** Un objeto que represente un error en la invocación del método. Debe ser nulo si la invocación fue exitosa.

***id*** Exactamente el mismo identificador del mensaje de solicitud correspondiente.

**Notificación** Funciona de la misma manera que un mensaje de solicitud pero en este caso el agente solicitante no esperará una respuesta del agente proveedor. La única diferencia con el mensaje de solicitud es que en este caso el campo de *id* debe ser nulo.

Los tipos de los parámetros pueden ser cualquier tipo de dato elemental soportado por JSON, o arreglos de tipos elementales. Para dar soporte a tipos de datos complejos, JSON-RPC define una extensión al lenguaje que permite definir clases anidadas dentro de los objetos JSON utilizando la siguiente propiedad, la cual es compatible con la sintaxis estándar de JSON:

```
{ "__jsonclass__": ["constructor", [param1,...]],
  "prop1": ... }
```

JSON-RPC puede ser utilizado directamente sobre flujos TCP o con HTTP como mecanismos de transporte. En el caso de utilizar HTTP, las solicitudes o notificaciones deben realizarse mediante una solicitud HTTP POST, y la respuesta debe enviarse en el cuerpo de la respuesta HTTP. Adicionalmente, utilizando HTTP es posible agrupar más de una solicitud o notificación JSON-RPC en una misma solicitud HTTP POST, en cuyo caso el agente proveedor debe enviar la respuesta a todas las solicitudes agrupadas en el cuerpo de la correspondiente respuesta HTTP. El par cliente puede utilizar solicitudes POST vacías para permitir que el servidor realice notificaciones al cliente.

La especificación del protocolo define que el cerrar una conexión entre pares debe provocar una excepción por cada mensaje de solicitud que aun no haya sido respondido. En el caso de TCP como mecanismo de transporte, las excepciones deben levantarse tanto el agente solicitante como en el agente proveedor. Si el mecanismo de transporte es HTTP, solo hace falta levantar las excepciones en el agente solicitante.

Para mostrar un ejemplo del funcionamiento de JSON-RPC utilizaremos el siguiente sistema de *chat* hipotético, tal como es descrito en la especificación del protocolo [6]. La arquitectura general del sistema puede apreciarse en la Figura 2.

En este ejemplo todos los entes actúan como agentes proveedores. Los clientes exponen por lo menos dos métodos llamados `handleMessage(String usr, String msg)`

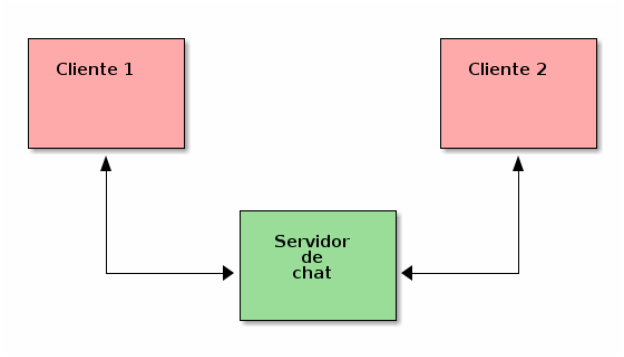


Figura 2: Un sistema de chat sencillo.

el cual se encarga de desplegar por pantalla un mensaje identificado por su remitente, y `userLeft(String usr)` el cual indica que uno de los participantes de la comunicación ha dejado la sala de *chat*. Por su parte el servidor expone un método llamado `postMessage(string msg)` el cual distribuye un mensaje a todos los usuarios conectados con el servidor. Una posible interacción entre estos agentes es la siguiente:

```
...
--> {"method": "postMessage",
     "params": ["Hello all!"],
     "id": 99}
<-- {"result": 1,
     "error": null,
     "id": 99}
<-- {"method": "handleMessage",
     "params": ["user1", "we were just talking"],
     "id": null}
<-- {"method": "handleMessage",
     "params": ["user3", "sorry, gotta go now, ttyl"],
     "id": null}
--> {"method": "postMessage",
     "params": ["I have a question:"],
     "id": 101}
<-- {"method": "userLeft",
     "params": ["user3"],
     "id": null}
<-- {"result": 1,
     "error": null,
     "id": 101}
...
```

En este ejemplo los mensajes identificados con el símbolo `-->` viajan del cliente al servidor, y los mensajes marcados con `<--` viajan del servidor al cliente. Podemos observar en el ejemplo como el cliente envía un mensaje de solicitud al servidor para colocar el mensaje *"Hello all!"* en el sistema. El servidor responde indicando la ejecución exitosa del método y luego envía dos notificaciones al cliente (caracterizadas por su *id* nulo) indicando que se deben mostrar sus respectivos mensajes. Posteriormente el cliente envía otro mensaje al servidor, el cual primero envía una notificación de tipo `userLeft` y posteriormente responde exitosamente al mensaje enviado por el cliente.

### JSON-RPC 2.0

Esta versión del protocolo es muy similar a la versión 1.0, siendo las siguientes las principales diferencias entre ambos [7]:

- El modelo de comunicación en JSON-RPC 2.0 es estrictamente cliente-servidor, a diferencia de JSON-RCP 1.0 donde el modelo es *peer-to-peer*.
- JSON-RPC 2.0 es completamente independiente del mecanismo de transporte, y no levanta excepciones ante conexiones rotas o mensajes no válidos.
- Los mensajes de solicitud y respuesta incluyen una propiedad adicional llamada `jsonrpc` cuyo valor es siempre "2.0".
- Los parámetros en los mensajes de solicitud y notificación pueden omitirse.
- Las propiedad `id` de los mensajes de solicitud y respuesta deben ser valores de tipo numérico entero, cadena de caracteres o nulo, y este campo siempre debe estar presente.
- Los mensajes de notificación no deben contener la propiedad de `id`.
- La especificacion de objetos `__jsonclass__` fue eliminada de la especificación base y se agrega como una extensión opcional al protocolo.
- Se puede agregar múltiples objetos de tipo solicitud dentro de un arreglo JSON para solicitar al servidor la ejecución por lotes de múltiples procedimientos. El servidor debe responder con un arreglo de objetos de tipo respuesta con los resultados de todas las invocaciones.

JSON-RPC 2.0 define adicionalmente una serie de códigos de error estándar para ser enviados por el servidor en los mensajes de respuesta. Estos códigos pueden verse en la Tabla 3. Las implementaciones tienen libertad de definir códigos de error adicionales siempre y cuando estén fuera del rango -32768 a -32000. Las respuestas de error pueden incluir un campo de datos opcional de libre implementación.

Tabla 3: Códigos de error de JSON-RPC 2.0. Tabla recuperada de [7].

| Código | Error |
|---|---|
| -32700 | JSON no válido |
| -32600 | Solicitud no válida |
| -32601 | Método no encontrado |
| -32602 | Parámetros no válidos |
| -32603 | Error interno de JSON-RPC |
| -32099 a -32000 | Error interno del servidor |

### 3.3. Otros protocolos

Además de los tres protocolos listados en esta Sección, se pueden distinguir los siguientes protocolos para implementar servicios Web.

**JSON-WSP** Este protocolo se define como una extensión de JSON-RPC la cual añade un tipo de mensaje utilizado para realizar la descripción del servicio. Este protocolo es utilizado principalmente por la herramienta Ladon[5], y no posee una especificación formal.

**SOAPjr** Se planteó como un protocolo que combinara las características de SOAP con la sencillez de JSON-RPC. Durante algún tiempo se intentó colocar a este protocolo como un Estándar de Internet con la IETF (Internet *Engineering Task Force* - Fuerza de Trabajo de Ingeniería de Internet), sin embargo este esfuerzo nunca dio frutos. Actualmente el mantenimiento del protocolo se encuentra abandonado.

**XMPP** Es un Estándar de Internet, definido en los RFC 6120[6] y 7622[7]. Es la base del sistema de mensajería Jabber[8]. Este protocolo no fue diseñado para implementar servicios Web aunque se caracteriza por ser tan versátil que en la práctica se ha demostrado posible utilizarlo para este fin.

## Conclusiones

Se presentó la definición formal de la W3C de lo que es un servicio Web, haciendo especial énfasis en la relación que tienen estos con el protocolo SOAP y otras estructuras computacionales definidas igualmente por la W3C. Posteriormente se presentó la arquitectura general de un servicio Web, notando que este se compone de dos entidades llamadas solicitante y proveedora. La entidad solicitante se compone de una organización o incluso un solo usuario humano que desean utilizar uno o más de los servicios que expone la entidad proveedora a

[5] http://ladonize.org/index.php/Common_jsonwsp
[6] https://www.rfc-editor.org/rfc/rfc6120.txt
[7] https://www.rfc-editor.org/rfc/rfc7622.txt
[8] http://www.jabber.org/

travéz de la Web. Ambas entidades se comunican mediante agentes de software los cuales están configurados con un documento llamado "descripción del servicio".

Finalmente se estudiaron a detalle tres protocolos de capa de aplicación comunmente utilizados para definir e implementar servicios Web. El primero de estos es el protoclo SOAP, el cual es el estándar más difundido para el uso de servicios Web y es especificado y promovido directamente por la W3C. Los otros protocolos estudiados son XML-RPC y JSON-RPC los cuales se presentan como alternativas más sencillas a SOAP, especializadas en realizar comunicaciones mediante el paradigma de llamadas a procedimientos remotos (RPC).

# Anexos

# A. Propuesta de solución para el ejercicio: Centro de Monitoreo CEMON

Se propone desarrollar un sistema basado en el protocolo JSON-RPC 2.0 utilizando el método POST de HTTP como protocolo de transporte. JSON-RPC no posee un lenguaje o mecanismo para establecer la definición del servicio, por lo tanto se propone que el servidor exponga un procedimiento como el siguiente, cuya firma es dada en lenguaje C:

```
typedef struct SERVICE_RET_TYPE {
  double disponibility;
  char * service_name;
} service_ret_t;

service_ret_t *
get_disponibility(const char * start_date, const char * end_date);
```

Los parámetros de fecha son cadenas de caracteres ascii dadas en el formato "AAAA-MM-DD", por ejemplo "2016-07-29". Esto es meramente ilustrativo, puesto que dadas las condiciones del ejercicio el servidor en realidad no tiene porque procesar o validar las fechas.

Un cliente utilizaría el método anterior enviando mensajes de solicitud de JSON-RPC 2.0 a través de HTTP POST. Un ejemplo de un mensaje de solicitud válido sería el siguiente, utilizando argumentos posicionales según la sintaxis de JSON-RPC:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "get_disponibility",
  "params": ["2015-01-01", "2015-12-31"]
}
```

Una posible respuesta sería:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
              "disponibility": 0.44377905554551544,
              "service_name": "DATABASE"
            }
}
```

De igual forma es posible hacer la solicitud al servidor utilizando argumentos nombrados de la siguiente manera:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "get_disponibility",
  "params": {
              "start_date": "2015-01-01",
              "end_date": "2015-12-31"
            }
}
```

En este caso la respuesta es similar a la del ejemplo anterior.

El servidor no tiene necesidad de implementar el mecanismo de notificaciones ni de solicitudes por lotes de JSON-RPC 2.0.

En el repositorio [8] se encuentra una implementación de referencia de esta especificación con un servidor escrito en el lenguaje de programación Python utilizando el framework Web.py[9], junto a un cliente escrito en el lenguaje C desarrollado con las bibliotecas libcurl[10] y Jansson[11]. Dicho repositorio incluye instrucciones detalladas sobre como instalar y ejecutar tanto el servidor como el cliente.

# Referencias


[1] D. Booth et al., *Web Services Architecture*, 11 de frebrero de 2004, https://www.w3.org/TR/ws-arch/, consultado el 13 de julio de 2016.

[2] M. Gudgin et al., *SOAP Version 1.2 Part 0: Primer (Second Edition)*, 27 de abril de 2007, http://www.w3.org/TR/soap12/, consultado el 26 de julio de 2016.

[3] D. Box, A Brief History of SOAP, 4 de abril de 2001, http://www.xml.com/pub/a/ws/2001/04/04/soap.html, consultado el 26 de julio de 2016.


[9] http://webpy.org/

[10] https://curl.haxx.se/

[11] http://www.digip.org/jansson/

[4] D. Ferguson, *Exclusive .NET Developer's Journal "Indigo" Interview with Microsoft's Don Box*, 10 de agosto de 2004, http://dotnet.sys-con.com/node/45908, consultado el 26 de julio de 2016.

[5] D. Winer, *XML-RPC Specification*, 15 de junio de 1999, http://xmlrpc.scripting.com/spec.html, cosultado el 13 de julio de 2016.

[6] JSON-RPC.ORG, *JSON-RPC 1.0 Specifications*, 8 de julio de 2012, http://json-rpc.org/wiki/specification, cosultado el 13 de julio de 2016.

[7] JSON-RPC Working Group, *JSON-RPC 2.0 Specification*, 1 de abril de 2013, http://www.jsonrpc.org/specification, cosultado el 13 de julio de 2016.

[8] M. Astor, *JRPC-CEMON*, 13 de julio de 2016, https://github.com/miky-kr5/JRPC-CEMON.