3.1 INTRODUCCIÓN AL LENGUAJE XML

3.1.1 ORIGENES

XML (eXtensible Markup Language) comenzó a desarrollarse en septiembre de 1996 auspiciado por el W3C para diseñar un lenguaje de marcado optimizado y hoy es el estándar internacional para intercambio de información, multiplataforma y multi-entorno, combinando la simplicidad de HTML con la capacidad expresiva de su predecesor, **SGML**. En pocos años, XML se ha convertido en el lenguaje con mayor impacto en el desarrollo de aplicaciones, promovido por Microsoft, IBM, Sun, entre otras empresas de software. **[LIB014]**

3.1.2 DEFINICIÓN

XML es un Meta-Lenguaje abierto, que permite la definición de lenguajes de representación de documentos, para su uso en la WWW, permite describir el sentido o la semántica de los datos a diferencia del HTML, separa el contenido de la presentación.

XML permite crear una estructura de marca con absoluta libertad para estructurar información, la flexibilidad y la potencia de esa estructura permite definir información con una riqueza enorme (Objetos completos, imágenes, procesos complejos, entre otros.) Esa información se expresa en texto, entendible para cualquier procesador. **[WWW021]**

3.1.3 CARACTERÍSTICAS

- La posibilidad de marcado descriptivo, con un conjunto de marcas abierto.
  En XML las marcas diferencian los contenidos informativos de los documentos, frente al uso que se hace en HTML para visualizar contenidos.
- XML no especifica un conjunto válido de marcas, sino que nos ofrece reglas que nos permiten crear nuevos conjuntos de marcas.

- Riqueza informativa y estandarización.
- Hacer explícita su estructura y sus contenidos informativos.
- Crear documentos que puedan intercambiarse y procesarse con facilidad en aplicaciones heterogéneas.
- No es: una aplicación, un lenguaje o producto.**[WWW022]**

3.2 MARCADO, DATOS Y ESTRUCTURA DE DOCUMENTOS XML

3.2.1 MARCADO

Todos los documentos XML deben cumplir lo siguiente:

- Si no se utiliza una hoja de estilos, el documento debe comenzar con una declaración de documento **Standalone**.
- Todas las etiquetas (Tags) deben estar balanceadas: es decir, todos los elementos que contengan datos de tipo carácter deben tener etiquetas de principio (apertura) y fin (cierre) con el mismo nombre.
- Todos los valores de los atributos deben ir entre comillas (si el valor contiene caracteres comillas dobles, y viceversa utilizar ' y ")
- Cualquier elemento vacío (que no tienen etiqueta final como <IMG>, <HR>, y otros de HTML) deben terminar con '/>' o hacerlos no vacíos con una etiqueta de fin.
- No debe haber etiquetas aisladas (< ó &) en el texto (< y &), y la secuencia]]> debe darse como]]> si no ocurre esto como final de una sección marcada como CDATA.
- Los elementos deben estar correctamente anidados y no puede haber superposiciones de ningún tipo en ningún caso.
- Tener un solo elemento raíz el cual contiene a todos los demás.
- Los atributos no pueden repetirse dentro de un elemento.

XML es muy estricto en lo referido a su construcción, porque estamos hablando de estructuras confiables y estándares para conducir la información y no podemos permitir la falta de confiabilidad. **[LIB010]**

3.2.2 DATOS

Hemos visto algunos conceptos teóricos de XML, pero comprendamos su ventaja en el campo de las aplicaciones. XML puede definir esquemas de datos como campos, tablas, relaciones, clave primaria entre otras, de manera que se pueda trabajar con esquemas y no emplear la data.

Trayendo consigo ventajas para trabajar con la estructura de una manera fácil y transparente, las aplicaciones pueden intercambiar datos, conociendo la estructura de la información que se envía, y ya no serán simplemente una cadena de caracteres sín sentido que era muy difícil de interpretar por aplicaciones ajenas a la enviante. De esta forma optimizamos nuestras aplicaciones en la comunicación de datos. **[LIB011]**

3.2.3 ESTRUCTURA

Una forma de entender la estructura de un documento XML es con un ejemplo:

```
<?xml version="1.0" encoding="UTF-7" ?>
    <dato>
        <nombre>DAVID</nombre>
        <apellido>VASCONEZ</apellido>
        <titulo>Ingeniero</titulo>
        <edad>24</edad>
    </dato>
    <dato>
        <nombre>HIRALDA</nombre>
        <apellido>RECALDE</apellido>
    </dato>
```

Cada documento XML posee una estructura lógica y una física.

- La estructura lógica es una serie de declaraciones, elementos, comentarios, entre otros, que se indican en el documento mediante marcas explícitas.

- La estructura física del documento es una serie de unidades llamadas entidades, que indica los datos que contendrá el documento. Las estructuras lógica y física deben anidarse de forma correcta. **[WWW023]**

3.2.4 ESQUEMAS

Un esquema de XML describe la estructura, o apariencia de un documento XML, es una alternativa al **DTD** (complejidad y menor legibilidad), ya que son extensibles, más ricos y útiles con capacidad expresiva (sobre todo en cuanto a tipos se refiere). Un esquema nos permite definir los tipos de datos y las cualidades de los elementos que pueden aparecer un documento, actualamente existen varios esquemas pero analizaremos los más utilizados.

a. XSL

XSL (eXtensible Stylesheet Language), según el W3C, XSL es "un lenguaje para transformar documentos XML", asi como un vocabulario XML para especificar semántica de formateo de documentos. En definitiva, además del aspecto que ya incluía la presentación y estilo de los elementos del documento, añade una pequeña sintáxis de lenguaje de script para poder procesar los documentos XML de forma más cómoda.

Al igual que con HTML hasta ahora, se pueden especificar las hojas de estilo, sean CSS o XSL, dentro del propio documento XML o referenciandolas de forma externa. XSL se estructura en base al concepto de plantilla de trabajo, que básicamente es un conjunto de instrucciones modelo, entremezcladas con etiquetas XSL que repiten bloques de instrucciones para cada nodo seleccionado, de un documento XML. Existen etiquetas adicionales que sustituyen el valor de los nodos especificados en las instrucciones modelo; lo que se trata de hacer es acoplar los diferentes datos que existen en un documento XML, transformándo en un esquema que reconozca tanto la sintaxis de HTML como XML. **[LIB011]**

b. XSLT

XSLT (Extensible Stylesheet Language Transformation), permite que podamos darle un estilo de presentación a los documentos XML, es decir que los datos también pueden tener una capa de presentación, ajena a su funcionalidad como data misma. Un archivo XSLT nos permite transformar un documento XML de un formato de presentación a otro.

El modelo de procesamiento XSLT consiste en la técnica siguiente:

1. El proceso inicia con el recorrido del árbol del documento, buscando una coincidencia con una regla para cada nodo que se visita.
2. Una vez que se encuentra una regla el cuerpo de esta se convierte en un documento. Las instrucciones de procesamiento XSLT dentro de la regla usan al nodo que coincide a su contexto.
3. Si se desea procesamiento adicional, se debe incluir la instrucción xsl:apply–templates. Los nodos que habrán de procesarse se encuentran especificados mediante el atributo match. Si se omite éste, el comportamiento predeterminado consiste en procesar los elementos secundarios del nodo actual, ejemplo: **[LIB012]**

Documento XML

```
<?xml version="1.0" encoding="UTF-8" ?>
  <datos>
     <dato>
        <nombre>DAVID</nombre>
     </dato>
  </datos>
```

Documento XSLT

```
<?xml version="1.0" encoding="utf-8" ?>
   <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/datos">
     <dato>
        <xsl:apply-templates select="nombre" />
     </dato>
   </xsl:template>
</xsl:stylesheet>
```

3.3 COMPONENTES DE UN DOCUMENTO XML

3.3.1 ELEMENTOS

1. PRÓLOGO Y DECLARACIÓN STANDALONE

La primera línea del prólogo específica la versión de XML y la codificación de carácter (US-ASCII, UTF-8, BIG5, ISO-8850-7, para más información ver el Anexo 3.1) ejemplo:

```
<?xml version="1.0" encoding="UTF-7" ?>
<?xml-stylesheet type="text/xsl" href="choose.xsl"?>
  <nombre>DAVID</nombre>
```

La segunda línea, define que tipo de documento. En la parte del prólogo se puede incluir una declaración de documento standalone que controla que componentes son necesarios para completar el procesamiento del documento, ejemplo:

```
<?xml version="1.0" standalone='yes'?>
     <nombre>DAVID</nombre>
```

El valor "yes" indica que no existen declaraciones de marcas externas a la entidad documento. El valor "no" indica que existe o que puede haber dichas declaraciones de marcas.

2. ELEMENTOS Y CONTENIDO

Los elementos XML tienen contenido (elementos, caracteres o ambos) o elementos vacíos. Un elemento con contenido empieza con <etiqueta> que puede contener atributos o no, y termina con </etiqueta> que debe tener el mismo nombre ejemplo:

```
<nombre>DAVID</nombre>
```

HTML permite elementos sin contenido, XML también, pero la etiqueta debe ser de la siguiente forma: <elemento-sin-contenido/>, que puede contener atributos o no ejemplo:

```
<identificador DNI=”1002920252”/>
```

3. ATRIBUTOS

Los elementos pueden tener atributos, que son características o propiedades a los elementos de un documento, ejemplo:

```
<alumno nota="5" asistencia="Nula">Pozo</alumno>
<noticia titulo="Pase de nivel" autor="David "DV"" />
```

Los atributos tienen que estar delimitado con comillas dobles o comilla simple. Cuando se usa uno para delimitar el valor del atributo, el otro se puede usar dentro.

```
<dato titulo="Ingeniero">David</dato>
<dato titulo="Ingeniero" nombre="David" />
```

4. COMENTARIOS

Los comentarios pueden aparecer en cualquier parte del documento, fuera del resto de las marcas, tienen el mismo formato de HTML "<!--comentario-->".

```
<!—nombre del alumno EISIC -->
<nombre>DAVID</nombre>
```

5. ENTIDADES PREDEFINIDAS

En XML 1.0 se definen cinco entidades para representar caracteres especiales como se muestra en la siguiente tabla:

ENTIDAD	CARACTER
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Tabla 3.1 Entidades para caracteres especiales

6. SECCIONES CDATA

Las secciones CDATA también nos permiten especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como una marca XML. Las secciones CDATA empiezan con "<![CDATA[" y terminan con "]]>". No se pueden anidar secciones CDATA, ejemplo:

```
<?xml version="1.0" encoding="UTF-7" standalone="yes" ?>
<ejemplos>
   <ejemplo>
      <![CDATA[
         <HTML>
         <HEAD> <TITLE> Rock & Roll </TITLE></HEAD>
      ] ] >
  </ejemplo>
</ejemplos>
```

7. INSTRUCCIONES DE PROCESAMIENTO

Las intrucciones de procesamiento permiten a los documentos XML contener instrucciones para las aplicaciones, van entre **<?** y **?>**, ejemplo:

```
<?xml version="1.0" ?>
   <?cocoon-process type="xslt"?>
```

8. IDENTIFICACIÓN DEL LENGUAJE

Puede ser útil la identificación entre el lenguaje natural o formal, en el que está escrito el contenido. Un atributo denominado **xml:lang** puede ser insertado en los documentos para especificar el lenguaje,ejemplo:

```
<?xml version="1.0" encoding="UTF-7" standalone="yes" ?>
<ejemplos>
    <p xml:lang="en">El poder.</p>
    <p xml:lang="en-US">El poder?</p>
    <sp who="David" desc="bajo" xml:lang="de">
        <l>Bienvenidos</l>
        <l>Mi trabajo,</l>
     </sp>
 </ejemplos>
```

Al utilizar espacios, tabuladores y líneas en blanco (en código fuente) se utiliza el atributo **xml:space** que puede tomar los valores **preserve** o **default**.**[WWW023] [LIB014]**

```
<?xml version="1.0" ?>
<textos>
    <cita>A quien madruga Dios le ayuda ?</cita>
    <poema xml:space="preserve">El monte es verde, el agua </poema>
</textos>
```

3.3.2 PARSER XML

Un parser es un módulo, biblioteca o programa que se ocupa de transformar un archivo de texto en una representación interna. En el caso de XML, como el formato siempre es el mismo, no necesitamos crear un parser cada vez que hacemos un programa.

El W3C ha especificado dos mecanismos o normas que indican la manera de acceder a los documentos. Estas incluyen una jerarquía de objetos que tienen métodos y atributos con los que se trabaja, simplificando las tareas de recorrido y acceso a las partes del documento, estos dos mecanismos son SAX y DOM tendremos más información sobre este último ya que se lo utiliza con AJAX. **[WWW024]**

a) **SAX**

Significa Simple **API** for XML, se utiliza para hacer un recorrido secuencial de los elementos del documento XLM, tiene una interfaz del estilo:

saxParse(documento, f_inicio_elemento, f_fin_elemento, f_texto)

En que los principales argumentos son punteros a funciones. Estas funciones serán ejecutadas por el parser cuando él se encuentre con un elemento inicial, con uno final, o con texto, ejemplo:

```
<p>Hola   <b>mundo</b></p>
```

Si ejecuto saxParse(documento, fstart, fend, ftext), se produce la siguiente secuencia de invocaciones:

```
fstart(http://www.w3.org/TR/REC-xml-
names/"p")
ftext("Hola")
fstart("b")
ftext("mundo")
fend("b")
fend("p")
```

b) DOM

El DOM se originó como una especificación para permitir que los programas Java y los scripts de JavaScript fueran portables entre los navegadores Web. El "HTML Dinámico" fue el ascendiente inmediato de DOM. **[WWW023]**

DOM es un API basada en árbol para XML. Su enfoque principal no es sólo para analizar XML, sino para representar ese código en objetos que pueden modificarse y accederse directamente.

i. DEFINICIÓN

DOM es la interfaz que permite manipular y acceder, mediante programación, a los contenidos de un documento. Proporciona una representación estructurada, orientada a objetos, de los elementos individuales y el contenido de una página, con métodos para recuperar y fijar las propiedades de dichos objetos.
Hay que notar que DOM no es JavaScript ya que de hecho se lo implementa en otros lenguajes como Java. Para los navegadores Web, sin embargo, el DOM se lo ha implementado usando ECMAScript y es una parte fundamental del JavaScript.
DOM, proporciona métodos para agregar y eliminar dichos objetos, permitiéndo crear contenido dinámico, para trabajar con eventos, para capturar y responder a las acciones del usuario o del navegador, a continuación se representa el DOM de la siguiente figura:

```
<TABLE><TBODY>
  <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
  </TR>
  <TR>
      <TD>Over the River,
            Charlie</TD>
      <TD>Dorian</TD>
  </TR>
</TBODY></TABLE>
```

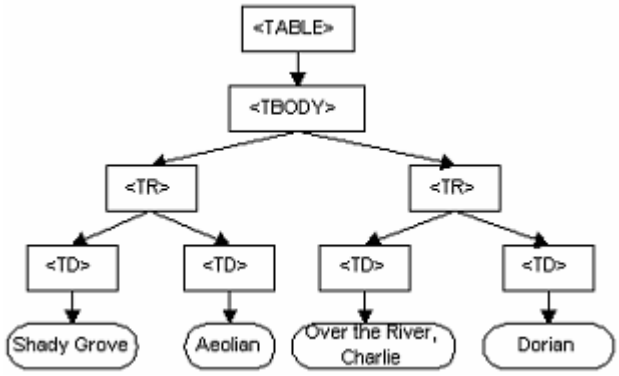

Figura 3.1 Representación del DOM

En DOM, los documentos tienen una estructura lógica como un árbol, pero es más bien como un "bosque", que puede contener más de un árbol. Sin embargo puede implementarse de cualquier manera que sea conveniente.

Una propiedad importante de los modelos de estructura del DOM es su isomorfismo estructural: si dos implementaciones cualesquiera del DOM se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

Los documentos se modelizan usando objetos, y el modelo comprende no solamente la estructura de un documento, sino también el comportamiento de un documento y de los objetos de los cuales se compone. En otras palabras, los nodos del diagrama anterior no representan una estructura de datos, sino que representan objetos, los cuales pueden tener funciones e identidad, DOM identifica:

- las interfaces y objetos usados para representar y manipular un documento
- la semántica de estas interfaces y objetos, incluyendo comportamiento y atributos
- las relaciones y colaboraciones entre estas interfaces y objetos **[LIB013]**

Al hablar sobre los árboles DOM se está hablando sobre una jerarquía de nodos donde se define la interfaz del nodo como un número grande de tipos de nodo para representar los siguientes aspectos múltiples del código XML:

- **Document,** El nodo de nivel mayor donde los otros nodos se atan
- **DocumentType**, Representa una referencia al DTD que usa la sintaxis <! DOCTYPE HTML “-//W3C//DTD HTML PÚBLICO 4.0 Transitional//EN”>. no puede contener nodos hijos.
- **DocumentFragment**, puede usarse como Document para sostener otros nodos
- **Element**, Representa el contenido de una etiqueta XHTML Este tipo del nodo es el único que puede contener los atributos como nodos hijo.

- **Attr**, Representa un atributo par (nombre-valor). Este tipo de nodo no puede tener nodos hijos.
- **Text**, Representa texto plano en un documento XML contenido dentro de la una Sección de CData. Este tipo del nodo no puede tener nodos hijos.
- **CDataSection**, Se representa por <! [CDATA []]>. Este tipo del nodo sólo puede tener nodos de texto como los nodos hijo.
- **Entity**, Representa una definición de entidad. Esto tipo de nodo no puede tener nodos hijo.
- **EntityReference**, Representa una referencia de la entidad, como ". Este tipo del nodo no tiene nodos hijo.
- **ProcessingInstruction**, Representa a PI. Este tipo del nodo no tiene nodos hijo.
- **Comment**, Representa un comentario XML. Este tipo del nodo no tiene nodos hijo.

ii. SOPORTE Y NIVELES

La mayor parte de los navegadores usaban un array de objetos por ejemplo Image para representa todas las etiquetas IMG en una página. Entonces, podía accederse a éstas y manipularlas vía JavaScript. Estos modelos iniciales estaban limitados, sólo proporcionaban acceso a unos pocos tipos de elementos y atributos, como imágenes, links o formularios.

Ventajosamente, la mayoría de navegadores empezaron a adoptar el estándar DOM fijado por el 3WC, los navegadores actuales soportan el estándar DOM2. Además, continúan soportando algunas características de los anteriores niveles del DOM, o sus propias extensiones propietarias, por lo que son compatibles con páginas Web antiguas.

La norma se define en niveles en lugar de versiones:

- **Nivel 1**: se volvió una recomendación del W3C en octubre de 1998. Consistió en dos módulos: el DOM CORE que proporcionó una manera de trazar la estructura de un documento XML para permitir el acceso y manipulación de

cualquier parte de un documento, y el DOM HTML que extendió el DOM Core agregando métodos para objetos HTML.

- **Nivel 2**: DOM Level 2 es una extensión al DOM nivel 1, a más de exponer la estructura de un documento se agrega el soporte para el ratón y eventos de interfaz del usuario (soportado por DHTML), rangos de tiempo, y soporte para CSS. DOM Level 2 incorpora nuevos módulos para trabajar con la interfaz como:
    - DOM Views (Vistas): describe varios estilos CSS de vistas en una interfaz.
    - DOM Events (Eventos): describe los eventos para la interfaz.
    - DOM Style (Style): describe los estilos CSS para la interfaz.
    - DOM Traversal, Range (Transversal y Rango): describe la interfaz para manipular un documento árbol.
- **Nivel 3**: extiende a DOM introduciendo métodos para cargar y grabar documentos de manera uniforme (nuevo módulo llamado DOM Load and Save) así como los métodos para validar el documento (DOM Validation). En DOM Nivel 3, el DOM Core se extiende soportando todo el formato XML 1.0, incluyendo XML Infoset, XPath, y XML Base. **[LIB015]**

iii. ESTRUCTURA DEL MODELO

1. El Árbol del Documento

Cuando un navegador carga una página, crea una representación jerárquica de sus contenidos que representa, aproximadamente, su estructura HTML. Esto desemboca en una organización parecida a un árbol de nodos, cada uno representando un elemento, un atributo, contenido o algún otro objeto.

2. Nodos

Cada uno de estos tipos diferentes de objetos tendrá sus propios y únicos métodos y propiedades. Pero cada tipo implementará también la interfaz Node. Esta interfaz es un conjunto común de métodos y propiedades relacionadas con la estructura de árbol

del documento. La interfaz Node proporciona métodos para añadir, actualizar y eliminar nodos dinámicamente, como:

- insertBefore(), inserta un nuevo nodo hijo antes del nodo referencia.
- replaceChild(), elimina el nodo hijo de referencia.
- removeChild(), elimina el nodo hijo de referencia.
- appendChild(), añade el nuevo nodo al final de los nodos hijos
- cloneNode(), hace una copia del nodo.

3. La Raíz (Root) Document y sus métodos

El objeto document sirve de raíz del árbol, este tiene nodos hijos pero no tendrá nodo padre ni nodos de su mismo nivel, dado que él es el nodo inicial. Además de ser un Node, también implementa la interfaz Document.

Cabe aclarar que DOM tiene varios métodos que no soportan todos los navegadores, la siguiente tabla muestra los métodos para crear nodos, incluidos en DOM Level1 que los navegadores soportan.

Método	Descripción	IE	MOZ	OP	SAF
createAttribute(name)	Crea un nodo del atributo con el nombre dado	X	X	X	-
createCDATASection(text)	Crea una Sección CDATA con un nodo hijo que contiene el texto	-	X	-	-
createComment(text)	Crea un nodo del comentario que contiene el texto	X	X	X	X
createDocumentFragment()	Crea un fragmento del documento del nodo	X	X	X	X
createElement(tagname)	Crea un elemento con una etiqueta de nombre tagname	X	X	X	X
createEntityReference(name)	Crea un nodo con entidad de referencia obtenida de name	-	X	-	-
createProcessingInstruction(target, data)	Crea un nodo PI	-	X	-	-
CreateTextNode (text)	Crea un nodo de texto	X	X	X	X

IE = Internet Explorer 6, MOZ = Mozilla 1.5, OP = Opera 7.5, SAF = Safari 1.2 / MacOS.

Tabla 3.2 Métodos de DOM Level 1 y soporte en los navegadores

A continuación se describen los más utilizados en el desarrollo:

Para crear y manipular nodos:

- createElement(),createTextNode(), appendChild(), por lo general se usan juntos estos métodos. Ejemplo:

  Vamos a crear mediante DOM Hola

  Primero creamos el elemento , luego creamos el nodo texto y lo agregar al elemento usando appendChild () este existe en cada nodo, finalmente el elemento debe ser atado al elemento document.body o a un nodo hijo, usando appendChild () como se muestra a continuación:

```
…
<head>
<script type=”text/javascript”>
        function createMessage() {
                var oP = document.createElement(“p”);
                var oText = document.createTextNode(“Hola “);
                oP.appendChild(oText);
                document.body.appendChild(oP);
        }
</script>
</head>
<body onload=”createMessage()”>
…
```

- removeChild(), replaceChild() e insertBefore(), estos métodos se usan naturalmente, si se quiere quitar, remover o insertar un nodo respectivamente. Usando el ejemplo anterior se eliminará el mensaje “Hola” de la siguiente manera:

```
…
<head>
<script type=”text/javascript”>
        function removeMessage() {
                var oP = document.body.getElementsByTagName(“p”)[0];
                document.body. removeChild (oP);
        }
</script>
</head>
<body onload=”removeMessage()”>
…
```

Ahora vamos a reemplazar el contenido del nodo:

```
…
<head>
<script type=”text/javascript”>
        function replaceMessage() {
                var oNewP = document.createElement(“p”);
                var oText = document.createTextNode(“Chao “);
                oNewP.appendChild(oText);
                var oOldP = document.body.getElementsByTagName(“p”)[0];
                oOldP.parentNode.replaceChild(oNewP, oOldP);
        }
</script>
</head>
<body onload=”replaceMessage()”>
…
```

Ahora vamos a insertar el contenido del nodo:

```
…
<head>
<script type=”text/javascript”>
        function insertMessage() {
                var oNewP = document.createElement(“p”);
                var oText = document.createTextNode(“Chao!! “);
                oNewP.appendChild(oText);
                var oOldP = document.getElementsByTagName(“p”)[0];
                document.body.insertBefore(oNewP, oOldP);
        }
</script>
</head>
<body onload=”insertMessage()”>
…
```

- createDocumentFragment(), permite insertar una gran cantidad de nodos al documento, en el ejemplo siguiente se añade 5 parrafos:

```
var arrText = [“uno”, “dos”, “tres”, “cuatro”, “cinco”];
var oFragment = document.createDocumentFragment();
for (var i=0; i < arrText.length; i++) {
        var oP = document.createElement(“p”);
        var oText = document.createTextNode(arrText[i]);
        oP.appendChild(oText);
        oFragment.appendChild(oP);
}
document.body.appendChild(oFragment);
```

- getElementsByTagName(), devuelve un objeto NodeList con el nombre de su argumento más la lista de sus descendientes. Ejemplos:

```
var oPs = document.getElementsByTagname("p");
var oImgsInP = oPs[0].getElementsByTagName("img");
var oAllElements = document.getElementsByTagName("*");
```

- getElementsByName(), recupera todos los elementos que tienen su nombre del atributo con un valor específico. Ejemplo:

```
…
<input type="radio" name="radColor" value="rojo" />
Red<br />
<input type="radio" name="radColor" value="verde" />
Green<br />
<input type="submit" value="Submit" />
…


var oRadios = document.getElementsByName("radColor");
Ahora se puede manipular los radios:
alert(oRadios[0].getAttribute("value")); // "rojo"
```

- getElementById (), recupera más rápido un solo elemento específico del árbol del documento mediante un id. En HTML, el id del atributo es único no se puede repetir. Ejemplo:

```
<p>Hello World!</p>
<div id="div1">This is my first layer</div>
…
var oDiv1 = document.getElementById("div1");
```

4. Recorriendo el árbol del Documento

Técnicamente, el objeto Document tiene sólo un elemento hijo, dado por document.documentElement, para páginas web, éste representa la etiqueta exterior HTML, y ésta actúa como el elemento raíz del árbol del documento, teniendo los elementos hijos HEAD y BODY que tendrán a su vez otros elementos hijos.

Utilizando esto, y los métodos de la interfaz Node, se puede recorrer el árbol del documento para acceder a los nodos individuales contenidos en dicho árbol, ejemplo:

```
…
<body><p>Este es un ejemplo</p></body>
…
```

```
alert(document.documentElement.lastChild.firstChild.tagName);
```

5. Tipos de Nodos

Existen varios tipos de nodos definidos en DOM, pero los más utilizados en páginas web son elemento, texto y atributo.

- **Elemento**, corresponden a las etiquetas individuales o pares de éstas en el código HTML. Éstas pueden tener nodos hijos (children), que pueden ser otros elementos o nodos de texto.
- **Texto** representan contenido, o datos de carácter, tienen un nodo padre (parent) y, posiblemente, nodos del mismo nivel (siblings), pero no pueden tener nodos hijos.
- **Atributo** son un caso especial. No están considerados una parte del árbol, posibilitan el aceso a los atributos de un nodo elemento. **[LIB016]**

  Aunque la interfaz del Nodo tiene un método de los atributos que se hereda por todos los tipos del nodo. La propiedad de los atributos para un nodo del Elemento es un NamedNodeMap que mantiene varios métodos accediendo y manipulando su contenido como:
  - getNamedItem(name): devuelve el nombre del nodo
  - removeNamedItem(name): quita el nombre nodo de la lista
  - setNamedItem(node): agrega el nodo en la lista, poniéndolo un índice
  - item(pos) como NodeList, retorna el nodo en una posición numérica

```
Por ejemplo:
        <p id=”p1”>Hola mundo!</p>
Para acceder al valor del  atributo seria:
        var sId = oP.attributes.getNamedItem(“id”).nodeValue;
o
        var sId = oP.attributes.item(1).nodeValue;
o cambiar el valor de atributo:
        oP.attributes.getNamedItem(“id”).nodeValue = “newId”;
```

DOM también define tres métodos para la asignación de atributos:

- getAttribute(name): como attributes.getNamedItem(name).value
- setAttribute(name,newvalue):como

  attributes.getNamedItem(name).value = newvalue

- removeAttribute(name): como attributes.removeNamedItem(name)

Estos métodos son útiles ya que tratan directamente los valores del atributo y esconden los nodos de Attr. Usando el ejemplo anterior:

```
var sId = oP.getAttribute("id");
oP.setAttribute("id", "newId");
```

iv. GESTIÓN DE MEMORIA

Los APIs del Núcleo del DOM están diseñados para ser compatibles con un amplio espectro de lenguajes. Así, los APIs de DOM necesitan trabajar bajo distintas filosofías de gestión de memoria, desde plataformas de lenguaje que no exponen al usuario a la gestión de memoria en absoluto y aquellas (especialmente Java) que proporcionan constructores explícitos para reclamar memoria libre, pasando por aquellas que generalmente exigen que se reserve explícitamente la memoria para el objeto. Para asegurar que el API es consistente en todas estas plataformas, DOM no contempla en absoluto las cuestiones de gestión de memoria, sino que deja éstas a cada implementación. **[LIB016]**

v. INTERFACES DOM

El DOM especifica interfaces que pueden utilizarse para manipular documentos XML o HTML, constituyen un medio de especificar una forma de acceder y manipular la representación interna que una aplicación hace de un documento. Las interfaces no implican una implementación concreta en particular. Cada aplicación DOM es libre de mantener los documentos según una representación conveniente cualquiera, siempre y cuando soporte las interfaces mostradas en esta especificación. Algunas implementaciones del DOM serán programas existentes que usen las interfaces del DOM para acceder a programas escritos mucho antes de que existiera la especificación del DOM. Por tanto, el DOM se ha diseñado para evitar dependencias de la implementación:

1. Los atributos definidos no implican objetos concretos que deban tener miembros de datos específicos.

2. Las aplicaciones DOM pueden proporcionar interfaces adicionales y objetos que no se encuentren en esta especificación y seguir siendo considerardas como objetos DOM.
3. Debido a que especificamos interfaces, y no los objetos reales que deben ser creados. En general, los usuarios del DOM llamarán a métodos de la clase Document para crear estructuras del documento y las implementaciones del DOM crearán sus propias representaciones internas.**[LIB016]**

3.4 SERVICIOS WEB

Los servicios Web son aplicaciones Web auto contenidas, creadas con el propósito de intercambiar información entre distintas aplicaciones en la Web. Ellos permiten lograr una alta interoperabilidad, con eficiente integración entre las aplicaciones involucradas, representan servicios entre sistemas distribuidos heterogéneos.

Los Servicios Web son por definición recursos Web, y como tales están identificados por una URI, son asequibles usando los protocolos de la Web y además tienen el potencial de contar con descripciones (metadatos), para encontrarlos, clasificarlos, compararlos, componerlos e inferir propiedades sobre ellos. De este modo, el conjunto de Servicios Web en Internet es una WWW paralela, de carácter no humano, sino cibernético. **[WWW025]**

3.4.1 ARQUITECTURA

En la figura 3.2, podemos deducir que un servicio web se registra en un repositorio de servicios, el cliente busca en el repositorio el servicio que necesita y luego lo invoca. De manera más detallada "La arquitectura de los servicios Web es una meta-arquitectura que permite que ciertos servicios de red sean dinámicamente descritos, publicados, descubiertos e invocados en un ambiente de cómputo distribuido".
Los servicios Web pueden ser:

- Descritos mediante un lenguaje de descripción de servicio, como el lenguaje WSDL (Web Service Description Language)
- Publicadas al someter las descripciones y políticas de uso en algún Registro bien conocido, utilizando el método de registro UDDI (Universal Description, Discovery and Integration).
- Encontradas al enviar peticiones al Registro y recibir detalles de ligamiento (binding) del servicio que se ajusta a los parámetros de la búsqueda.
- Asociadas al utilizar la información contenida en la descripción del servicio para crear una instancia de servicio disponible o proxy.
- Invocadas sobre la red al utilizar la información contenida en los detalles de ligamento de la descripción del servicio.
- Compuestas con otros servicios para integrar servicios web y aplicaciones nuevas." **[LIB017]**

Bajo esta definición más precisa de la arquitectura podemos analizar la siguiente figura:

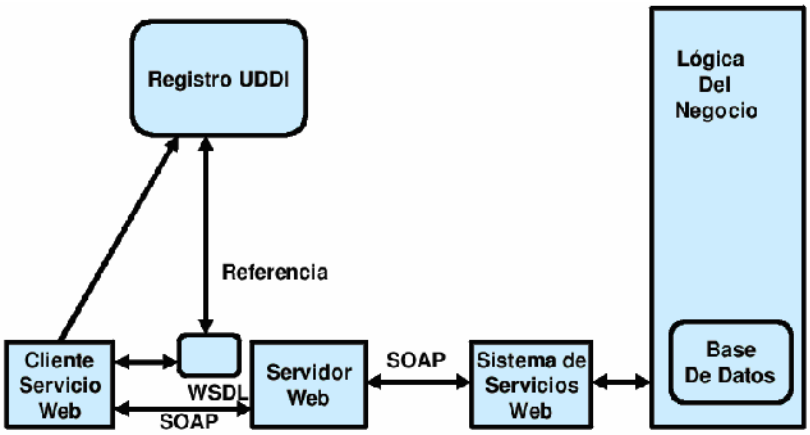

Figura 3.2 Arquitectura Servicios Web

Ahora podemos dar una descripción más real del funcionamiento de los servicios web. El servicio web es construido y luego descrito por medio de WSDL y registrado bajo el estándar UDDI, el cliente busca en el registro UDDI (como si fuese un motor de búsqueda) y obtiene el descriptor WSDL del servicio que necesita, lo invoca haciendo uso de SOAP el cual también es utilizado para comunicar la petición entre los diferentes componentes del servidor que aloja el servicio WEB, para entregar una

respuesta utilizando nuevamente SOAP (más información se encuentra en el Anexo 3.2).

3.4.2 FUNCIONAMIENTO

En la figura 3.3, un usuario a través de una aplicación, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes. La agencia ofrecerá a su cliente (usuario) la información requerida. Para proporcionar al cliente la información que necesita, esta agencia solicita a su vez información a otros recursos (otros Servicios Web) en relación con la línea aérea. La agencia obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros Servicios Web que le van a proporcionar la información solicitada sobre la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia que servirá de intermediario entre el usuario y el servicio Web que gestionará el pago.
En todo este proceso intervienen una serie de tecnologías que hacen posible esta circulación de información. **[LIB017]**

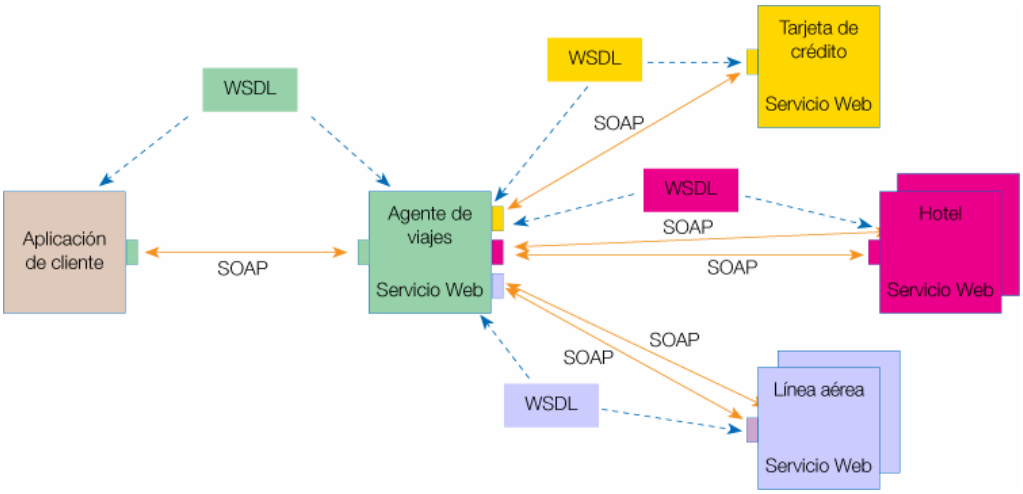

Figura 3.3 Los Servicios Web en funcionamiento

Por un lado, estaría SOAP (Protocolo Simple de Acceso a Objetos). Se trata de un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP. SOAP especifica el formato de los mensajes.

Por otro lado, WSDL (Lenguaje de Descripción de Servicios Web), 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, a través de un documento procesable por dispositivos. WSDL representa una especie de contrato entre el proveedor y el que solicita. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes. **[LIB017]**

3.4.3 PROTOCOLOS Y ESTÁNDARES

XML. Este es un lenguaje de marcado al estilo de HTML que pretende dar las pautas generales para la estructuración de información. De XML se pueden generar dialectos los cuales se adaptan a un problema particular de intercambio de información.

SOAP (Simple Object Access Protocol). Es un dialecto de XML el cual permite a las aplicaciones invocar métodos de objetos remotos, así como recibir las respuestas de los mismos.

WSDL (Web Service Description Language) Es al igual que SOAP, un dialecto de XML que contiene información acerca de la interfaz, semántica y administración de una llamada a un servicio Web.

UDDI (Universal, Description, Discovery, and Integration) Es un estándar para describir los componentes disponibles de servicios Web. Este permite a las empresas registrarse en un tipo de directorio sección amarilla de Internet que les ayuda anunciar sus servicios, de tal forma que que las compañías se puedan encontrarse unas a otras y realizar transacciones en la Web.
Lo más importante es que UDDI contiene información sobre las interfaces técnicas de los servicios de una empresa. A través de un conjunto de llamadas a API XML basadas en SOAP, de este modo, UDDI es una especificación para un registro distribuido de información sobre servicios Web. **[WWW025]**