TEMA 4.GESTIÓN DE LA INFORMACIÓN

Objetivos:

- *Conocer diversas tecnologías para almacenar la información en plataformas Web*
- *Ser capaz de interpretar documentos XML y valorar su uso en soluciones Web*
- *Conocer los aspectos básicos de la Web Semántica y su influencia en los sistemas de información basados en Web*

INTRODUCCIÓN

La Web es un entorno en el que hoy en día la mayoría de empresas y organizaciones han volcado sus esfuerzos operativos, bien ampliando o incluso migrando sus procesos cotidianos, tanto internos como externos.

Las aplicaciones que hacen uso de Internet pueden ser:

- B2B, *business-to-business*, aplicaciones en las que interactúan dos empresas entre sí.
- B2C, *business-to-customer*, sistemas que ponen en contacto a la empresa con su cliente final.
- G2B, *government-to-business*, aplicaciones que conectan a entes gubernamentales con empresas.
- G2C, *government-to-customer*, aplicaciones de interacción entre la ciudadanía y los entes del gobierno.
- Librerías digitales.

En todas estas aplicaciones es necesario procesar texto, introduciéndolo en formularios a tal efecto. Esta información (textual, numérica o de otro orden) sólo tiene sentido si es utilizada en los procesos de negocio, intercambiada y explotada en las transacciones comerciales habituales.

Ejercicio 32. *¿Puede enumerar un par de aplicaciones de cada uno de los grupos indicados (B2B, B2C, G2B, G2C)?*

Pero en la Web hay mucha más información además de la que intuitivamente podamos imaginar. Es normal que pensemos en conjuntos de información específicos: viajes, recetas de cocina, tipos de cambio monetarios, publicaciones científicas, rutas en bicicleta, etc. Toda esta información posiblemente esté ya almacenada en servidores web, cuyos contenidos alguien se encarga de mantener y actualizar. Pero en la WWW aún hay más información: aquella que se encuentra diseminada por las páginas web, mezclada entre los textos e imágenes.

Por un lado tenemos bases de datos razonablemente bien diseñadas, con sus diagramas relacionales o jerárquicos, estructurada, orientada a un único dominio de la aplicación… pero por otro tenemos información sin un diseño centralizado, sin un sistema gestor de bases de datos que lo respalde a la forma tradicional y cubriendo cualquier dominio de aplicación.

En definitiva podemos determinar que en la Web los datos se encuentran de forma desestructurada, semi-estructurada y estructurada:

- ***Datos desestructurados***: archivos de texto, video, e-mails, informes, presentaciones powerpoint, imágenes, etc. Estos documentos no se pueden clasificar ni almacenar bien en una base de datos clásica, salvo en un campo de tipo BLOB (*binary large object).* Además, al ser cada documento independiente, no es posible extraer una estructura o información semántica común.

- ***Datos semi-estructurados***. Son datos que tienen cierta estructura, pero no muy rígida, esto es, que permite ciertas variaciones. Un ejemplo podrían ser los CVs que encontramos en la red. Hay personas que mostrarán su experiencia profesional previa, otros no, pero aportan experiencia científica.
- ***Datos estructurados***. Todos los registros tienen la misma estructura, fuertemente tipada. Desde la aparición de las bases de datos relacionales es la forma tradicional de organizar la información, y de hecho los ERP, CRM y CMS más comunes hoy en día hacen uso de datos fuertemente estructurados para la gestión de la información.

Ejercicio 33. *¿Qué es un ERP?¿Y un CRM? Complete en sus apuntes al menos un par de páginas sobre estos sistemas software y comente al menos un ejemplo de ERP o CRM basado en Web.*

En este tema vamos a comentar en mayor o menor profundidad la gestión de la información en los Sistemas Web, partiendo de lo ya conocido (los datos estructurados) hasta adentrarnos ligeramente en la extracción de información desestructurada.

GESTIÓN DE DATOS ESTRUCTURADOS

La gestión de datos estructurados en los Sistemas de Información Web no difiere mucho de los programas que se ejecutan en entornos locales. Se suele utilizar un modelo relacional para la estructuración de la información, siguiendo una tradicional arquitectura cliente/servidor.

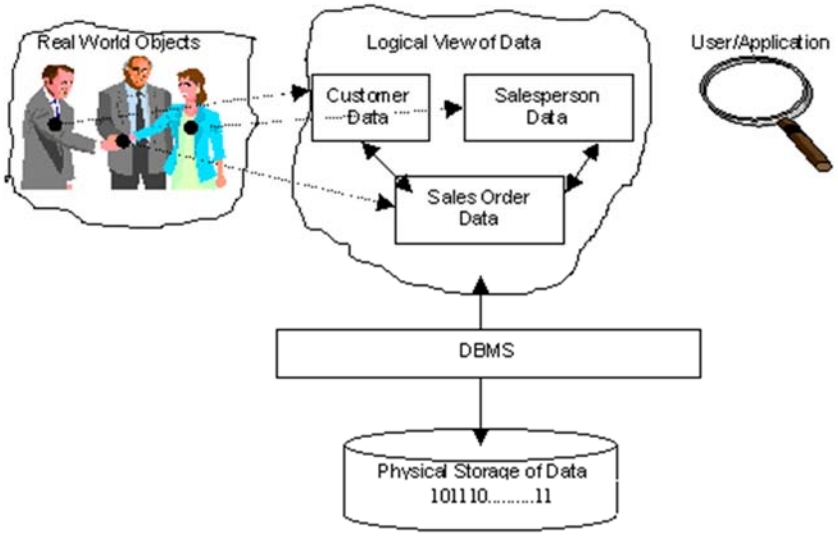

Ilustración 30: Típica aplicación basada en datos y archivos.[Eaglestone01]

De hecho, se suele hablar de una arquitectura de tres capas a la hora de describir los SIBW que usan bases de datos relacionales:

- Una **capa cliente**, el navegador web, que apenas tiene lógica y se encarga de mostrar la información al usuario final y recopilar de éste los datos para remitirlos al servidor HTTP
- Una **capa controlador** en el servidor, normalmente implementada con tecnologías de scripting (PHP, ASP.NET, etc.) o de servicios (*servlets, Web Services)*, que realiza tareas como la coordinación entre la capa cliente y los datos alojados en el servidor, la autenticación y securización de la información, el control de estados de la aplicación, etc.
- Una **capa de gestión de datos**. Esta capa se encarga de controlar la integridad de los datos, gestionar la concurrencia de accesos, proporcionar seguridad, ocultar si fuese necesario la estructura interna de la información, etc.

Afortunadamente, la mayoría de los Sistemas Gestores de Bases de Datos (SGBD) implementan la mayor parte de la funcionalidad requerida para la capa de gestión de datos. Recordemos que, de manera práctica, una base de datos no es más que un conjunto de archivos convenientemente indexados. Puede haber aplicaciones en las que usar un SGBD sea *"matar moscas a cañonazos"* pero serán las menos, en general, toda aplicación Web tendrá alguno de estos requisitos:

- más de un usuario pueden querer acceder a los datos de forma concurrente
- hay cierto volumen de información (p.ej. más de 100 clientes)
- los datos están relacionados entre sí (p.ej. nº Seg. Social y nómina)
- hay más de un tipo de datos
- hay restricciones en cuanto a formatos, tipos de datos, etc.
- se deben realizar búsquedas rápidas sobre los datos
- hay una jerarquía de roles
- se realizan continuas modificaciones sobre los datos almacenados

En todos estos casos, es imperativo utilizar un SGBD en el desarrollo de nuestro sistema Web. Ya será en función de la naturaleza de la información y los servicios que proveamos lo que determinará qué SGBD utilizar.

Al haber cursado a estas alturas de la carrera diversas asignaturas orientadas al diseño e implantación de bases de datos relacionales, vamos a centrarnos exclusivamente en una de ellas y en ciertos aspectos de su operativa con PHP, el lenguaje de scripting de servidor utilizado en las prácticas de la asignatura.

MySQL

MySQL es un sistema gestor de bases de datos eficiente y libre que se está convirtiendo en el estándar *de facto* en el almacenamiento de datos estructurados en la Web. De hecho, empresas como ticketmaster.com pasaron de Microsoft SQL Server a MySQL y mejoraron su eficiencia un 400%, o Wikipedia, que tiene montados 20 servidores de bases de datos sobre MySQL que soportan más de 350 millones de visitas mensuales.

> *En http://www.mysql.com/customers/ puede encontrar una lista de grandes empresas que usan MySQL como motor de sus bases de datos.*

Hay que dejar bien claro que MySQL no es gratis total. Es propiedad de Oracle, y es gratis sólo para aplicaciones liberadas como *open source*. El resto de aplicaciones que se distribuyan con MySQL deben pagar por el SGBD, o al menos eso dice la licencia.

MySQL se puede utilizar en aplicaciones de escritorio, sin conexión a la red, o en aplicaciones de servidor como las que nos ocupa.

En el caso de un servidor de base de datos, se ejecutará un *demonio* `mysqld` que estará atento por defecto al puerto 3306 TCP, escuchando las posibles peticiones que le entren. Para la gestión de la información almacenada se podrá ejecutar el programa `mysql` que abre un entorno tipo Shell para ejecutar sentencias SQL clásicas.

MySQL ofrece también librerías para distribuirse de forma embebida en aplicaciones de escritorio o portables, si bien su entorno más común es el acceso mediante APIs que proporcionan interfaces de acceso a los datos mediante el uso de SQL.

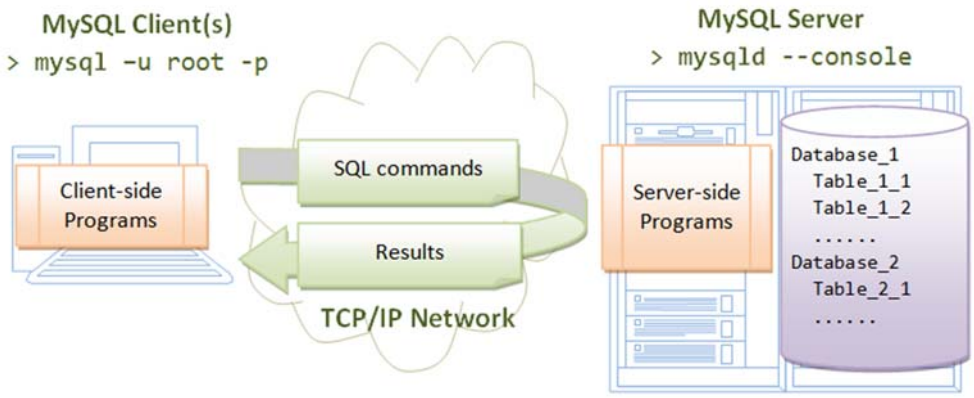

Ilustración 31: Cliente MySQL atacando un servidor MySQL.

Uso de MySQL y PHP

La extensión `mysqli` (mysql improved, o MySQL mejorada) permite acceder a la funcionalidad proporcionada por MySQL 4.1 y versiones posteriores, y se encuentra incluida en las versiones PHP 5 y posteriores. Algunas de las características que presenta respecto a la extensión `mysql` son:

- Interfaz orientada a objetos
- Soporte para Declaraciones Preparadas
- Soporte para Múltiples Declaraciones
- Soporte para Transacciones
- Mejoras en las opciones de depuración
- Soporte para servidor empotrado

La extensión mysql ofrece una interfaz dual, esto es, permite el uso de un paradigma procedural o uno orientado a objetos.

La tarea de obtener registros de una base de datos utilizando PHP se basa en cuatro pasos:

- Conectar con el servidor y seleccionar la base de datos
- Ejecutar una instrucción de SQL
- Procesar la información
- Cerrar la conexión con el servidor

Paso 1: Conectar con el servidor y seleccionar la base de datos MySQL

Dado que en un mismo servidor pueden coexistir varias bases de datos, es necesario además de conectar con el servidor MySQL decir a qué BD nos vamos a conectar en concreto. En el antiguo interfaz mysql esto se hacía con las instrucciones mysql_connect y mysql_select_db:

Sintaxis:

```
mysql_connect (servidor, usuario, clave);
```

Si la conexión tiene éxito devuelve un identificador, en otro caso devuelve false.

```
mysql_select_db (baseDeDatos, identificadorDeConexión);
```

Si la selección tiene éxito devuelve true, en otro caso devuelve false

Ejemplo:

```
$conexion = mysql_connect ("localhost", "usuario", "clave");
$abreBD = mysql_select_db ("SIBW_bd", $conexion);
```

Si bien en mysqli ya se puede realizar la conexión del tirón e incluso utilizar atributos de la conexión para evaluar el estado:

```
$conexion = new mysqli("localhost", "usuario", "clave", "SIBW_bd");
if ($conexion ->connect_errno) {
   echo "Fallo al contenctar a MySQL: " . $conexion ->connect_error;
}
```

Paso 2: Ejecutar una instrucción de SQL

Sintaxis mysql:

```
mysql_query (instrucciónSQL, identificadorDeConexión);
```

Si la instrucción se ejecuta correctamente devuelve true o las filas afectadas por la instrucción, dependiendo de la instrucción ejecutada, en otro caso devuelve false

Sintaxis mysqli:

```
mysqli::query (instruccionSQL);
```

Ejemplo:

```
$seleccion = "SELECT  *  FROM  productos";
$resultado = $conexión->mysql_query ($seleccion);
```

Paso 3: Procesar la información

Los contenidos de la variable $resultado son de lo más diverso, pues depende del tipo de orden SQL que se haya ejecutado y de si ésta devuelve una única fila o varias. Por ello hay varias funciones asociadas.

Sintaxis mysql:

```
mysql_num_rows ($resultado);
# Devuelve el número de filas afectadas por la instrucción ejecutada anteriormente
mysql_fetch_array ($resultado);
# Cada vez que se ejecuta esta instrucción extrae una fila del resultado como un array asociativo, o como un array numérico o como ambos;
mysql_fetch_assoc ($resultado);
# Cada vez que se ejecuta esta instrucción extrae una fila del resultado como un array asociativo
mysql_fetch_row ($resultado);
# Cada vez que se ejecuta esta instrucción extrae una fila del resultado como un array numérico
mysql_fetch_object ($resultado);
# Cada vez que se ejecuta esta instrucción extrae una fila del resultado como un objeto
mysql_data_seek ($resultado, númeroDeFila);
# Mueve el puntero de filas interno del resultado de MySQL asociado con el identificador de resultado especificado para apuntar al número de fila especificada mediante un valor o una variable; la siguiente llamada a una función de obtención de MySQL, tal como mysql_fetch_array() , devolverá esa fila
```

Ejemplo utilizando como índices valores numéricos en lugar de como array asociativo:

```
$resultado = $mysqli->query("SELECT id, etiqueta FROM test WHERE id = 1");
$fila = $resultado->fetch_assoc();

printf("id = %s (%s)\n", $fila['id'], gettype($fila['id']));
printf("etiqueta = %s (%s)\n", $fila['etiqueta'], gettype($fila['etiqueta']));
```

Paso 4: Cerrar la conexión con el servidor

La extensión mysqli soporta conexiones persistentes a bases de datos, las cuales son un tipo especial de conexiones almacenadas en caché. Por defecto, cada conexión a una base de datos abierta por un script es cerrada explícitamente por el usuario durante el tiempo de ejecución o liberada automáticamente al finalizar el script. Una conexión persistente no. En su lugar, se coloca en una caché para su reutilización posterior, si una conexión es abierta al mismo servidor usando el mismo nombre de usuario, contraseña, socket, puerto y base de datos predeterminada. La reutilización ahorra gastos de conexión.

Sintaxis:

```
mysql_close (idDeConexion);
```

o en mysqli

```
bool mysqli::close ( void )
```

Ejemplo:

```
mysql_close ($conexion);
$conexion->close();
```

Conexiones Persistentes a Bases de Datos

Las conexiones persistentes no se cierran cuando la ejecución del script termina. Cuando una conexión persistente es solicitada, PHP chequea si ya existe una conexión persistente idéntica (que fuera abierta antes) - y si existe, la usa. Si no existe, crea el enlace. Una conexión "Idéntica" es una conexión que fue abierta por el mismo host, con el mismo usuario y el mismo password (donde sea aplicable).

Las conexiones persistentes no dan otra funcionalidad que no fuera posible hacerse con sus hermanas no-persistentes.

¿Por qué existen entonces?

Esto tiene que ver con la forma en que los web servers trabajan. Lo normal es ejecutar PHP como un módulo en un servidor multiproceso, Apache, que tiene un proceso padre que coordina un grupo de procesos hijos. Cuando una solicitud viene desde el cliente, es manejada por uno de los hijos libres. Esto significa que cuando el mismo cliente hace una segunda solicitud al servidor, podría ser servido por un proceso hijo diferente a la primera vez. Cuando se abre una conexión persistente, ésta es reutilizada entre los distintos procesos hijos de Apache.

GESTÍON DE DATOS SEMIESTRUCTURADOS.

XML

Algunos autores consideran el *eXtensible Markup Language* (XML) como una de las tecnologías más importantes que se han desarrollado para la WWW. XML consiste en un conjunto de tecnologías interrelacionadas, especificadas todas mediante recomendaciones del W3C que ya hoy día han supuesto un cambio sustancial en el almacenamienot y procesamiento de los datos.

Ejercicio 34. *Documente en sus apuntes en qué consisten los ataques SQL Injection y cómo prevenirlos.*

Un metalenguaje de marcas es un lenguaje para definir lenguajes de marcas, y SGML (*Standard Generalized Markup Language)* es un metalenguaje de marcas. En 1986, SGML se aprobó como estándar, y en 1990 fue utilizado como base para HTML. En 1998 fue publicada la primera versión de XML.

¿Por qué aparece XML, existiendo ya HTML? HTML es un lenguaje para describir la estructura de un documento web, y por tanto restringe el conjunto de etiquetas y atributos definidos. Por otro lado, HTML tan sólo describe la estructura, y no aporta nada sobre la semántica de la información que está mostrando. Por ejemplo:

```
<h1>Fiat Doblo</h1>
```

Ejercicio 35.

¿En cuál de estos enlaces se describen las reglas sintácticas de SGML? Inclúyalas en sus apuntes.
http://www.w3.org/MarkUp/SGML/
http://www.w3.org/MarkUp/SGML/sgml-lex/sgml-lex
http://www.isgmlug.org/

```
<div>
<img src=”foto.jpg”/>
<ul>
<li>115.000kms</li>
<li>Diesel</li>
<li>Matriculada en 2005</li>
</ul>
<p>Precio (en euros): 3500</p>
<p>Contacto: 8592</p>
</div>
```

Es un código HTML para mostrar la ficha de un vehículo en venta. ¿Cuánto cuesta? Los humanos podemos leerlo rápido, ¿pero un robot de búsqueda? ¿qué diferencia la cadena “3500” de “8592”?

XML surgió como una versión simplificada de SGML, de manera que los usuarios (entendiendo a los usuarios como organizaciones con requisitos de datos y de procesos similares) pudieran definir su propio lenguaje de marcas basado en el estándar. Por ejemplo, los químicos, que usan una notación común entre sí y los procesos reactivos son los que son, definieron el CML (Chemical Markup Language) y sus derivados CMLReact, etc. Se puede ver un ejemplo en el código 13.

Otros lenguajes estándar, o públicamente documentados, derivados de XML son:

- FpML (Financial products Markup Language)
- Google Site Map
- GenXML (Genealogy XML)
- NewsML (News Markup Language)
- Office XML (OOXML)
- VoiceXML

```
<cml title="atomArray CML1">
<list>
     <atomArray>
          <atom id="a1" elementType="O" hydrogenCount="1"/>
          <atom id="a2" elementType="N" hydrogenCount="1"/>
          <atom id="a3" elementType="C" hydrogenCount="3"/>
     </atomArray>
     <!-- is equivalent to -->
     <atomArray
          atomID="a1 a2 a3"
          elementType="O N C"
          hydrogenCount="1 1 3"/>
</list>
</cml>
```

Código 13: Ejemplo en CML

Es importante entender que XML no es un sustituto de HTML. Sus objetivos son distintos: HTML es un lenguaje de marcas para describir la organización de cierta información y facilitar cómo debe visualizarse, mientras que XML es un meta-lenguaje de marcas que proporciona un marco para desarrollar lenguajes específicos de marcas. De hecho XHTML es una versión de HTML basada en XML.

XML es mucho más que una solución a las deficiencias de HTML. Proporciona un mecanismo universal y simple de guardar y acceder a cualquier dato textual, de forma que estos datos puedan ser distribuidos y procesados por diferentes aplicaciones, que serán muy fáciles de programar pues la sintaxis de los documentos será clara y directamente estructurable. Podemos decir que XML es un generador de lenguajes universales de intercambio de datos.

XML no es un lenguaje de marcas, sino que es un meta lenguaje que especifica reglas para generar lenguajes de marcas. XML no tiene etiquetas, sino que éstas se definen para cada uno de los lenguajes que se crean. De hecho, cada uno de estos lenguajes a veces se denominan *aplicaciones XML*, pero para no inducir a error, reservaremos el término aplicaciones al software y se denominará como *conjunto de etiquetas* esos lenguajes basados en XML. Los documentos que usan cualquier lenguaje basado en XML se denominarán *documentos XML*.

Los documentos XML se pueden escribir con un simple editor de texto, aunque esta solución es inmanejable por ejemplo para introducir la información de un catálogo con 100.000 bienes culturales, cada uno con decenas de campo. Para ello, lo mejor es generar una aplicación que de un interfaz amigable.

Sintaxis

La sintaxis de XML se puede pensar a dos niveles distintos. Por un lado está la sintaxis a bajo nivel que obliga a una serie de reglas a cualquier documento XML y por otro lado están las reglas sintácticas que se pueden definir mediante DTDs (Document Type Definitions) o XML

Podemos entender un documento XML como una jerarquía de contenidos, de hecho es un árbol etiquetado, ilimitado y ordenado:

- etiquetado, porque cada nodo del árbol tiene una etiqueta asociada
- ilimitado, porque no hay límite en cuanto al número de hijos de un nodo
- ordenado, porque está definido el orden de los hijos en el árbol

Además, otra ventaja de XML es que es serializable, de forma que un documento como:

```
<entry>
<name>
```

```
  <fn>Javier</fn>
  <ln>Melero</ln>
</name>
<work>
ETSIIT UGR
<adress>
  <city>Granada</city>
  <zip>18071</zip>
</adress>
<email>fjmelero@ugr.es</email>
</work>
<course>Sistemas Gráficos</course>
</entry>
```

se puede serializar en un único string

```
<entry><name><fn>Javier</fn><ln>Melero</ln></name><work>ETSIIT
UGR<adress><city>Granada</city><zip>18071</zip>
</adress><email>fjmelero@ugr.es</email></work><course>Sistemas
Gráficos</course></entry>
```

Ello no niega la naturaleza de árbol del documento, sino que permite transmitirlo por la red.

Elementos y texto

Los componentes básicos de un documento XML son los elementos y el texto.

Un elemento tiene una etiqueta de apertura, un contenido textual y una etiqueta de fin:

```
<etiqueta atributo='valor' ….> contenido </etiqueta>
<etiquetavacia />
```

La etiqueta de apertura puede incluir una lista de pares (nombre,valor) denominados atributos, como por ejemplo:

```
<informe idioma='es_ES' fecha='22/05/14'>
```

No se permiten dos atributos con el mismo nombre en un elemento.

La principal diferencia entre el contenido y los atributos de un elemento es que:

- el contenido está ordenado mientras que los atributos no
- el contenido puede estar formado por subárboles, mientras que el valor de un atributo es complejo.

Un documento XML debe siempre representar un árbol. Si no es posible crear un árbol, es porque está mal descrito. Si es correcto, se dice que el documento XML está *bien formado.*

Cabeceras y entidades

En los documentos XML hay dos elementos que no tienen información en sí, pero que son importantes:

- el prólogo, que proporciona indicaciones tales como la versión de XML utilizada, la codificación, la localización de recursos externos, etc.

```
<?xml version="1.0" encoding="utf-8"  ?>
```

- entidades, que son referencias o contenidos asignados a una variable que luego es usada en el árbol:

```
<!ENTITY  tema1 "Tema 1: Modelado de Sólidos " >
<!ENTITY  tema2 SYSTEM "theme2.xml" >
<report> &tema1; &tema2 </report>
```

En este caso, se sustituye la referencia a la entidad (p.ej. &tema2) por el contenido de ese archivo XML

Tipos, esquemas y espacios de nombres

No es que los documentos XML necesiten tener un tipo, pero sí tener un tipo.El mecanismo original de tipificación de los documentos era los *Document Type Definition, DTDs,* que aún se siguen usando por su simplicidad, ya que define las reglas con una sintaxis BNF.

Ejemplos de DTDs

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">

]>
```

```
<!DOCTYPE  email [
<!ELEMENT  email ( header, body )>
<!ELEMENT  header ( from, to, cc? )>
<!ELEMENT  to (#PCDATA )>
<!ELEMENT  from (#PCDATA )>
<!ELEMENT  cc (#PCDATA )>
<!ELEMENT  body (paragraph*) >
<!ELEMENT  paragraph (#PCDATA )>
]>
```

Esto hace que un e-mail bien formado tenga que ser algo parecido a esto.:

```
<email>
<header>
       <from>…</from>
       <to>…</to>
</header>
<body>
       <paragraph>Hola</paragraph>
</body>
</email>
```

Nótese como XML no obliga a que sea con mayúsculas o minúsculas.

Los DTDs tienen la siguiente sintaxis:

- elementos

```
<!ELEMENT element-name category>
<!ELEMENT element-name (element-content)>
```

- elementos vacíos

```
<!ELEMENT element-name EMPTY>
```

- elementos con datos alfanuméricos analizables

```
<!ELEMENT element-name (#PCDATA)>
```

- elementos con cualquier contenido

```
<!ELEMENT element-name ANY>
```

- elementos con un hijo

```
<!ELEMENT element-name (child-name)>
```

- elementos con varios hijos (secuencias)

```
<!ELEMENT element-name (child1,child2,...)>
```

- elementos con al menos una ocurrencia de un hijo

```
<!ELEMENT element-name (child-name+)>
```

- elementos con ninguna o más ocurrencias de un hijo

```
<!ELEMENT element-name (child-name*)>
```

- elementos con hijos opcionales

```
<!ELEMENT note (to,from,header,(message|body))>
```

- elementos con contenido mixto

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

Quizá ahora tenga más sentido la sentencia que suele encabezar muchos documentos HTML que vemos diariamente:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Un concepto también a tener en cuenta en XML son los namespace, que ya le sonarán a usted de lenguajes de programación como C++. La idea que subyace es que un mismo término puede significar cosas distintas en función del contexto. Pensemos en un trabajo, no es lo mismo una oferta de trabajo, que una demanda de trabajo. Se pueden utilizar entonces dos namespaces distintos para distinguir una u otra etiqueta job:

```
<doc xmlns:hire='http://a.hire.com/schema'
     xmlns:asp='http://b.asp.com/schema' >
<hire:job> ... </hire:job> ...
<asp:job> ... </asp:job> ...
</doc>
```

Sin embargo , los namespaces no están soportados por los DTDs.

La estructura del documento viene dada por estos esquemas, ahora bien ¿se debe asignar tipos o no?

- ¿Por qué no? Pues porque los orígenes y formatos de los datos puede ser tan variado (pensemos en las fechas, por ejemplo), que puede ser un engorro tener que especificar un formato o tipo concreto.
- ¿Por qué si? Pues porque con una restricción en los formatos mejoramos la interoperatibilidad de los documentos, se mejora la consistencia de la información y se acelera la indexación de la información

XML Schema

XML-Schema es el lenguaje de descripción de tipos XML propuesto por el W3C. Aunque al principio se decía que era excesivamente complicado para lo que hacía falta, el hecho es que se ha convertido en lo más utilizado por su versatilidad, ya que soporta y usa otros estándares como Xpath, Xquery, XSLT.

Veamos primero un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="book">
            <xs:complexType>
            <xs:sequence>
                  <xs:element name="title" type="xs:string"/>
                  <xs:element name="author" type="xs:string"/>
                  <xs:element name="character"
                              minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                        <xs:sequence>
                              <xs:element name="name" type="xs:string"/>
                              <xs:element name="age" type="xs:integer"/>
                              <xs:element name="since" type="xs:date"/>
                              <xs:element name="qualification”
                                                type="xs:string"/>
                        </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Nótese como los tipos están definidos como xs:string, xs:integer, xs:date, etc., dando lugar a documentos XML como

```
<name>Yo</name>
<age>34</age>
<since>1968-03-27</since>
```

También se pueden definir atributos, resultando un elemento como el book.

```
<book isbn=”314-2322-22”>…</book>
```

La etiqueta xs:complexType indica que el elemento puede contener un subárbol y dale un nombre a ese nodo:

```
<xs:complexType name="personinfo">
<xs:sequence> <xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/> </xs:sequence>
</xs:complexType>
```

XML Schema también permite ir más allá de un simple autómata, incluyendo referencias de clave primaria (`<xs:key />`) y claves externas (`<xs:keyref />`) de forma similar a las claves en las bases de datos relacionales.

A continuación pongo un extracto de un documento XML real, obtenido de un periódico online:

```
<?xml version="1.0" encoding="UTF-8"?>
 <elemento-mm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <plantilla>abc/ABC_itemGaleriaFullscreen</plantilla>
 <id>1511385032327</id>
 <portal>abc.es</portal>
 <edicion>nacional</edicion>
 <tipo>Galeria</tipo>
 <url>http://www.abc.es/fotos-sociedad/20131101/arte-retratar-muerte-
1511385032327.html</url>
 <tipoEspecial>
   <![CDATA[]]>
 </tipoEspecial>
 <fechacreacion>
  <![CDATA[2013-10-31T11:05:57Z]]>
 </fechacreacion>
 <fechapublicacion>
 <![CDATA[2013-11-01T08:16:28Z]]>
 </fechapublicacion>
 <fechamodificacion>
 <![CDATA[2013-11-01T09:16:29Z]]>
 </fechamodificacion>
 <titulo>
 <![CDATA[El arte de retratar la muerte ]]>
 </titulo>
 <entradilla>
<![CDATA[En España, desde mediados del siglo XIX hasta los años 80 del siglo XX fue
habitual fotografiar a los difuntos. El libro «El retrato y la muerte» recopila algunas
de estas instantáneas realizadas por fotógrafos de la época y explica el porqué de
esta tradición]]>
</entradilla>
<cuerpo/>
<imagen>http://www.abc.es/abc-nacional/multimedia/201311/01/media/godas-
velatorio.jpg</imagen>
<autores>
 <autor>
  <firma>
    <![CDATA[c. g.]]>
  </firma>
  <id/>
 </autor>
</autores>
<categorias>
 <categoriappal>
  <id/>
  <nombre>
  <![CDATA[Sociedad_sociedad]]>
  </nombre>
  <idpadre/>
  <nombrepadre>
  <![CDATA[Sociedad_sociedad]]>
  </nombrepadre>
</categoriappal>
<categoria>
<nombre>
```

```
<num_fotos>
<![CDATA[9]]>
</num_fotos>
<num_videos>0</num_videos>
<num_versiones>0</num_versiones>
<imagenes>
<imagen>
<url>http://www.abc.es/abc-nacional/multimedia/201311/01/media/reboredo-
bebe-difunto.jpg</url>
<titulo>
<![CDATA[ ]]>
</titulo>
<descripcion>
<![CDATA[Bebé difunto. (1892-1899) Además de los detalles vegetales, era
habitual la introducción en la escena de algún símbolo religioso. En este
caso, un crucifijo en medio de la corona de flores]]>
</descripcion>
<autor>
<![CDATA[Maximino Reboredo/Archivo Maximino Reboredo. Archivo Histórico
Provincial de Lugo]]>
</autor>
</imagen>
</imagenes>
<topics/>
<comscore>
<ns_site>abc-es</ns_site>
<voc_site>abc-es</voc_site>
<voc_se>multimedia</voc_se>
<voc_s1>
<![CDATA[Sociedad]]>
</voc_s1>
<voc_s2/>
<voc_s3/>
<voc_s4/>
<voc_ed>
<![CDATA[nacional]]>
</voc_ed>
<voc_tn>
<![CDATA[el-arte-de-retratar-la-muerte]]>
</voc_tn>
<voc_tc>fotogaleria</voc_tc>
<voc_or>RL</voc_or>
<voc_au>
<![CDATA[c. g.]]>
</voc_au>
<voc_fep>20131031110557</voc_fep>
<voc_fem>20131101091628</voc_fem>
<voc_pr>1</voc_pr>
</comscore>
     </elemento-mm>
```