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.

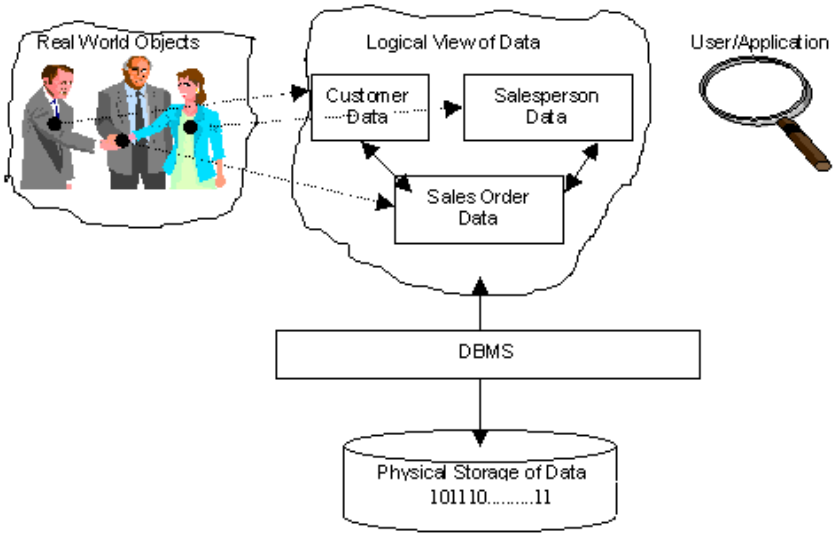


Ilustración 30: Típica aplicación basada en datos y archivos.[Eaglestone01]

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.

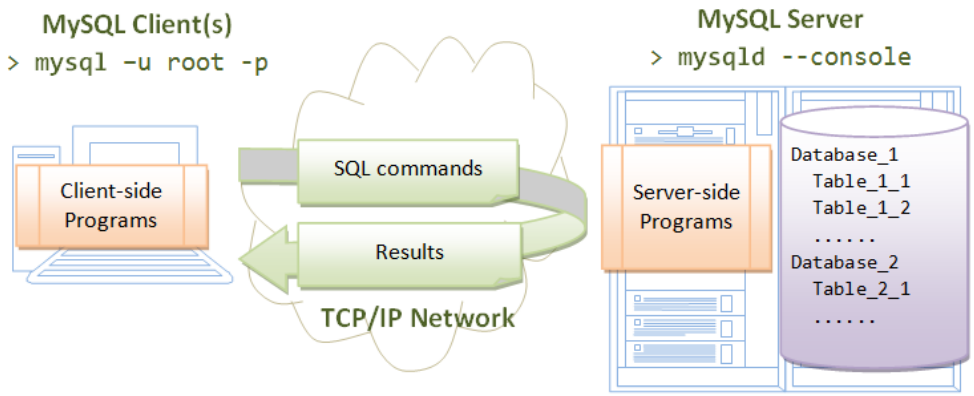


Ilustración 31: Cliente MySQL atacando un servidor MySQL.

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>
```

Ejercicio 36. *Documente en sus apuntes el formato de archivo SVG. Dibuje una escena y el árbol que forma el documento que la describe.*

Ejercicio 37. *MathML es un dialecto de XML, que describe fórmulas matemáticasusando XML, de forma que x^2+4x+4 se representa en preorden como*

(+ (^ x 2) (* 4 x) 4)

y en MathML con el siguiente XML:

```
<?xml version='1.0' ?>
<apply>
 <plus/>
 <apply>
       <power/>
       <ci>x</ci>
       <cn>2</cn>
 </apply>
 <apply>
 <times/>
       <cn>4</cn>
       <ci>x</ci>
 </apply>
 <cn>4</cn>
</apply>
```

Fíjese como los paréntesis son los <apply> y los signos las etiquetas <plus/> <times/> y <power />

Exprese las siguientes fórmulas en MathML:

- $(x^y+3xy)y$
- $x^{a+2}+y$

Ejercicio 38. *Documente en sus apuntes una seccíon para Xpath y otra para Xquery.*

Ejercicio 39. *Dado el siguiente código XML, genere el DTD y el XML Schema que lo valide.*

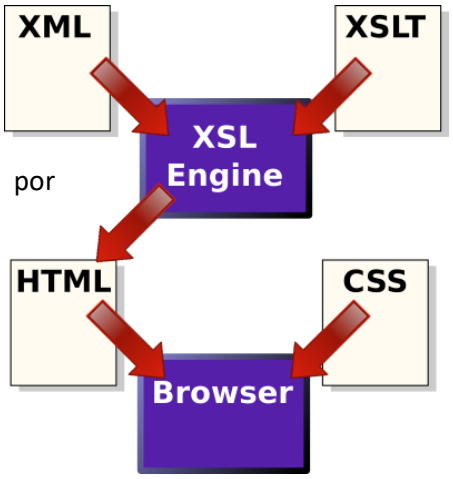
```
<zRobots>
 <Robot type="Astromech">
       <Id>R2D2</Id>
       <maker>Petric Engineering</maker>
       <components>
             <processor>42GGHT</processor>
             <store>1.5 zetabytes</store>
       </components>
 </Robot>
 <Robot type="Protocol">
       <Id>C-3PO</Id>
       <maker>Xyleme Inc</maker>
       <components>
             <processor>42GGHT</processor>
             <store>100 exabytes</store>
       </components>
 </Robot>
</Robots>
```

XSL

XSL significa eXtensible Stylesheet Language, y fue desarrollado por el W3C para cubrir las necesidades de visualización de documentos XML. Mientras que HTML tiene un conjunto de etiquetas con un significado bien conocido por los navegadores, y de hecho cada navegador tiene una hoja de estilo por defecto para esta visualización, para XML el significado de cada etiqueta depende del DTD o XML Schema en el que se base el documento a visualizar.

XSL consta de tres partes:

- XSLT, un lenguaje para transformar documentos XML
- Xpath, que sirve para navegar por los documentos XML
- XSL-FO, utilizado para formatear los documentos XML



Con XSL se pueden realizar cualquier tipo de transformaciones, por ejemplo:

- XML a HTML
- XML a XML
- XML a texto
- XML a PDF
- XML a …

```
<ACTO>
<TITULO>Acto Uno</TITULO>
<ESCENA>
   <TITULO>ESCENA UNA. Helsingor. Una terraza enfrente del castillo. </TITULO>
   <DIRESCENA>Francisco esta de centinela. Entra Bernardo.</DIRESCENA>
   <DIALOGO>
      <ACTOR>BERNARDO: </ACTOR>
      <FRASE>Quien esta ahi?</FRASE>
   </DIALOGO>
   <DIALOGO>
      <ACTOR>FRANCISCO: </ACTOR>
      <FRASE>El capitan Francisco.</FRASE>
   </DIALOGO>
</ESCENA>
</ACTO>
```

Código 15: XML de una escena teatral.

```
 <?xml version="1.0" ?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" />
  <xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>  <xsl:value-of select="//ACTO/TITULO" />     </TITLE>
    </HEAD>
   <BODY>
     <h1>     <xsl:value-of select="//ACTO/TITULO" />   </h1>
     <h2>     <xsl:value-of select="//ACTO/ESCENA/TITULO" />   </h2>
     <p><i>   <xsl:value-of select="//ACTO/ESCENA/DIRESCENA" />     </i></p>
     <xsl:for-each select="//ACTO/ESCENA/DIALOGO">
     <p><b>
       <xsl:value-of select="ACTOR" />
     </b></p>
     <p>
       <xsl:value-of select="FRASE" />
     </p>
        </xsl:for-each>
   </BODY>
  </HTML>
  </xsl:template>
  </xsl:stylesheet>
```

Código 14: XSL que convierte de la escena XML a HTML. Teatro.xsl

En el código 14 podemos ver un ejemplo de una escena teatral en XML, y en el código 15 el XSL que convertiría automáticamente esta información en un documento HTML. Los navegadores Web no saben mostrar documentos XML salvo que éstos vengan con su XSLT declarado y definido.

Para que el XSL del código 15 funcione con el ejemplo del código 14 habría que haber incluido al menos un par de líneas de cabecera al principio del documento de la escena: la versión de XML y el enlace al XSL que transforma.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="teatro.xsl"?>
```

XSLT

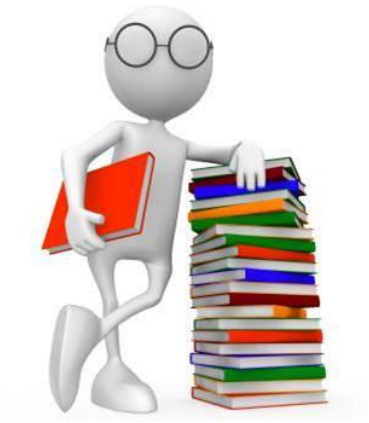
En la red hay inifinidad de tutoriales y recursos para aprender las tecnologías relacionadas con XSL. Destacamos los siguientes:

http://www.w3schools.com/xsl

http://www.xmlmaster.org/en/article/d01/c07/

http://www.uji.es/bin/ponencies/ixsl.pdf

Si usted desea aprender más sobre XSL, ahí tiene unos buenos puntos de partida antes de ponerlo en práctica.



La Web Semántica

El lenguaje natural es una herramienta evolutiva muy potente, que nos ha permitido distinguirnos de forma notable del resto de seres vivos del planeta. Es difícil pensar en una API que consiga tal potencia comunicativa.

Pensemos en dos frases simples (sujeto-verbo-predicado):

- Salvador disfruta los caracoles en salsa.
- Los caracoles repugnan a la mujer de Salvador.

Seguramente se ha sonreído al leer estas frases, si bien su estructura es similar. Aparecen nombres propios, un objeto que se convierte en sujeto en la siguiente y dos verbos. Lo que ha provocado su reacción seguramente ha sido la semántica. Conoce la diferencia entre “disfrutar” y “repugnar”. Esto es un ejemplo de semántica: los símbolos se pueden referir a cosas o conceptos, es la secuencia de símbolos (en este caso las palabras usadas en la frase) lo que ofrece un significado.

Otro ejemplo podría ser el uso de variables en un código fuente. Cuando uno programa le asocia un significado, una semántica, a cada variable. Normalmente el nombre que le ponemos está relacionado con la semántica que tiene en ese contexto (no solemos usar el identificador “leer” para una función que se encargue de ordenar una secuencia de elementos, por ejemplo). Si el significado no está claro se ponen comentarios en lenguaje natural, de forma que cualquier otra persona que acceda al código sea capaz de entender qué queremos hacer con cada identificador.

Pero en un entorno distribuido, con intercambio de datos entre distintas fuentes y en distintos contextos culturales, como es la Web, no hay posibilidad de acceder al código o a la estructura interna de las bases de datos relacionales. Antes de la WWW, cuando la comunicación a través de la red se realizaba entre interlocutores que querían comunicarse expresamente, se utilizaban unos acuerdos o protocolos de comunicación, los denominados RFC (Request For Comments), que se escribían en inglés y ofrecían una interpretación semántica de la estructura de los mensajes.

Con la explosión de la WWW, la comunicación entre las aplicaciones Web ya sólo requieren del protocolo HTTP. Esto ha posibilitado un mayor uso de la información, un intercambio más fluido y más fácil, pero a costa de no tener un RFC que interprete para cada aplicación la semántica de la información. ¿Cómo se gestiona ahora esa semántica? ¿Es realmente importante?

La visión de la Web Semántica es la de una arquitectura distribuida por todo el mundo donde los servicios y datos interactúan fácilmente. Esto no es aún una realidad, ya que es difícil encontrar

Ejercicio 40. *¿Cómo se establece la semántica en los sistemas de bases de datos relacionales? Es decir, ¿cómo se representa, por ejemplo, que un restaurante abre a ciertas horas y se dedica a un tipo de cocina concreto (india, china, libanesa, etc.)?*

recursos para una necesidad particular, y tampoco es fácil entender lo que estos servicios ofrecen. Para mejorar estas prestaciones de la Web actual y convertirla en algo realmente potente, la idea clave es publicar *descripciones semánticas* de los recursos Web, basadas en *anotaciones semáticas.*

Ontología. Una ontología es una descripción formal que proporciona a los usuarios humanos un conocimiento compartido sobre un dominio concreto.

En el ámbito que ocupa este tema, las ontologías pueden ser interpretadas y procesadas por máquinas gracias a una semántica lógica que permite el razonamiento. Las ontologías proporcionan la base para compartir el conocimiento y, por ello, son muy útiles para:

- ***organizar datos***, pues las ontologías son la forma natural de organizar o estructurar la información de forma flexible. Una ontología permitiría por ejemplo buscar una asignatura por titulación, cuatrimestre, universidad o curso.
- ***mejorar la búsqueda***, pues no es lo mismo que a la búsqueda *"jaguar negro"* se nos responda con fotografías de coches que de animales.
- ***integrar información***, pues lo que en un idioma se escribe *"car"* en otro es *"voiture"*,pero alineando sus ontologías se podrán integrar ambos símbolos bajo un mismo concepto.

Un aspecto esencial de las ontologías es su potencial, ya que gracias a la lógica inherente que tienen, serían capaces de inferir respuestas a preguntas que formuladas de forma específica no tendrían solución.

Una ontología es, como hemos dicho antes, una descripción formal de un dominio de interés basado en un conjunto de inviduos (entidades u objetos), clases de individuos y las relaciones entre esos individuos. La pertenencia de los individuos a las clases o las relaciones entre los individuos crean una base de *hechos*, una *base de datos*. Además, se usan sentencias lógicas que describen el conocimiento sobre clases y relaciones que especifican restricciones sobre la base de datos y forman la *base del conocimiento.* Cuando hablamos de una ontología, a veces se piensa sólo en este conocimiento que especifica el dominio de interés, pero algunas veces incluye tanto los hechos como las restricciones.

Tomemos un ejemplo para entender lo que es una ontología: una universidad. La ontología de la universidad incluye ciertas clases que denotan conceptos bien conocidos: :*Profesor*, :*Estudiante*, :*Departamento* o :*Asignatura*. Una clase tiene un conjunto de *instancias*, por ejemplo :*FJMelero* es una instancia de la clase :Profesor. La ontología además incluye relaciones entre las clases, por ejemplo, la ontología de la universidad podría incluir la relación *:Imparte* de forma que se modelaría *:Imparte(:FJMelero, :SIBW).*

Otros elementos que forman parte de las ontologías son las relaciones de subclases, por ejemplo :*Profesor* es una subclase de :*Personal*. Lo usual es representar este conjunto de relaciones de subclases con una jerarquía de clases, también denominada *taxonomía*.

Hay más restricciones que se pueden definir:

- *incompatibilidades*. Si un individuo pertenece a la clase :*Estudiante*, no puede pertenecer a la clase :*Profesor*
- *cardinalidad*. Por ejemplo: un :*Departamento* sólo puede tener un *:Director*
- *restricciones de dominio*, como que un *:ProfesorNoDoctor* sólo puede impartir docencia en :*AsignaturasDeGrado*

Cualquier entidad puede definir su propia ontología. Obviamente, cuanto más organismos participen en la definición de dicha ontología, mayor será su implantación y nivel de estandarización. Es el caso, por ejemplo, del Dublin Core, que contiene las ontologías relativas a las librerías digitales.

RDF, RDFS y OWL

RDF (Resource Description Framework) proporciona un lenguaje para describir anotaciones sobre recursos Web identificados mediante URIs (Unified Resource Identifiers). Esto es lo que hemos llamado *hechos.* Las restricciones o relaciones entre estos *hechos* se modelarán con RDFS u OWL.

Un URI es básicamente un identificador Web, como las cadenas que comienzan por "http:" o "ftp:" que se suelen ver en la WWW. Cualquiera puede crear una URI, siguiendo las reglas del RFC 2396 del W3C. Cualquier cosa que tenga un URI asociado se puede decir que está en la Web.

En RDF un hecho expresa algunos metadatos sobre un recurso identificable con un URI, y dicho hecho se expresa mediante una *tupla: sujeto, predicado y objeto.* La tupla *<a P b>* expresa el hecho de que *b* es el valor de la propiedad *P* para el sujeto *a.* En realidad, los términos relación, predicado o propiedad son sinónimos. En estas tuplas, el sujeto y también el predicado son URIs que apuntan a recursos Web, mientras que el objeto puede ser una URI o un *literal* que representa un valor.

Un ejemplo de RDF basado en las ontologías del Dublin Core puede ser el siguiente:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="http://en.wikipedia.org/Tony_Benn">
  <dc:title>Tony Benn</dc:title>
  <dc:publisher>Wikipedia</dc:publisher>
</rdf:Description>
</rdf:RDF>
```

RDFS (RDF Schema) es el lenguaje esquemático para RDF. Permite declarar objetos y sujetos como instancias de ciertas clases, y la relación entre ellos o restricciones de dominios. Algunos ejemplos:

```
:FJMelero rdf:type :Profesor
:Profesor rdfs:subClassOf :Personal
:Alberto rdfs:matriculadoEn :SIBW
:Imparte  rdfs:domain :Profesor
:Imparte rdfs:range :Asignatura
```

OWL (Web Ontology Language) amplía RDFS para permitir reflejar nuevas restricciones e incorpora una lógica de primer orden que facilita la inferencia de conocimiento (cardinalidades, intersecciones, uniones, propiedades de transitividad, simetría o similitud, etc.). Más información puede encontrarla en http://www.w3.org/TR/owl-features/.

GESTIÓN DE DATOS DESESTRUCTURADOS: LA BÚSQUEDA DE INFORMACIÓN

Además de la información almacenada en los SGBD de las infraestructuras Web de miles de sitios, de las ingentes cantidades de documentos XML que almacenan información más o menos estructurada, en la WWW hay billones de documentos accesibles con total libertad y gratuidad, pero sin ninguna estructura aparte de la que proporciona el HTML y la sintaxis propia de los lenguajes naturales en los que están escritos estos documentos.

Vamos a intentar describir de forma sucinta y rápida cómo funcionan los motores de búsqueda.

Rastreadores

Como bien saben, la primera tarea que realizan los buscadores es rastrear la web. La palabra inglesa es *to crawl* (gatear, reptar, trepar, hacer la pelota…), aunque los *crawlers* también se denominan *spiders* o *robots*. En cualquier caso, independientemente del término que se use, el proceso consiste en examinar el contenido tras una URL, examinarlo, indexarlo, detectar hiperenlaces y repetir recursivamente (o iterativamente) el proceso por cada uno de estos enlaces. En realidad es un proceso de búsqueda por un grafo que puede seguirse en anchura o en profundidad.

Para facilitar el trabajo de los robots, es recomendable el uso de sitemaps. Un archivo XML que se utiliza para listar de forma jerárquica los contenidos de una web, junto con alguna información extra (fecha de modificación, refresco…). Estos sitemap ofrecen al buscador un contexto ideal: toda la información con un solo archivo.

Una tarea importante de los rastreadores es la identificación de duplicados en la web, es decir, que si llega por dos url distintas al mismo contenido, sepa reconocerlo e indexarlo una única vez. Si la duplicación está en el mismo dominio, puede ser relativamente fácil su indexación, pero si son rutas o webs distintas, se deben aplicar técnicas de cálculo de distancias o de similtud entre los documentos de texto. Obviamente, no se puede medir la distancia exacta entre el casi infinito número de webs, pero sí se realizan muestreos aleatorios donde se buscan lo que se denominan *shingles*: una secuencia de tokens (palabras) de longitud prefijada (p.ej. k=3) que se encuentran en un documento. P.ej. con k=2

"I like to watch the sun set with my friend" →

S= {I like, like to, to watch, watch the, the sun, sun set, set with, with my, my friend}

"My friend and I like to watch the sun set" →

S'={my friend, friend and, and I, I like, like to, to watch, watch the, the sun, sun set}

Si se utiliza el coeficiente de Jaccard, la similitud es de 0.64:

$$J(S,S') = \frac{|S \cap S'|}{|S \cup S'|} = \frac{7}{11} \approx 0.64$$

Es obvio que estos cálculos son muy costosos, por eso hay técnicas basadas en hashing y computación distribuida que simplifican los costes de cómputo.

Podemos entonces inferir que el proceso de indexación de una página consiste en procesar el texto, generar una serie de índices y valorar numéricamente la importancia de cada sitio web en función de la búsqueda realizada.

Procesamiento del texto

El procesamiento del texto de una página web depende lógicamente del idioma en que ésta esté escrita, pero básicamente se puede resumir en la siguientes fases:

- tokenización, esto es, extraer las palabras
- limpieza, cuando se eliminan tokens que no aportan nada (artículos, determinantes, etc.) y se detectan posibles errores ortográficos o tipográficos.
- análisis semántico, donde se relacionarán términos similares (p.ej. política, politico, políticas…)
- indexación.

La indexación consise básicamente en crear una matriz entre términos de búsqueda y términos existentes en el artículo. Lo más habitual es tener índices invertidos, que es algo parecido a lo que hay al final de muchos libros y manuales, donde cada término tiene asociadas las páginas en las que es citado. La similitud con la web es obvia: donde hay números de páginas, poner URLs.

¿Cómo se evalúa la relevancia de un documento para una búsqueda concreta? Aquí es donde intervienen los pesos y repeticiones de los términos buscados. Un término que aparece varias veces en el documento es más relevante a la hora de indexarlo que términos aislados y sueltos, pero un término que aparece poco en el conjunto de documentos indexados hace que cuando éste aparece cobre especial relevancia.

Hay diversos algoritmos que se dedican a la indexación de las páginas, la creación de índices optimizados en tamaño y tiempos de acceso y técnicas para ordenar los resultados de la búsqueda, pero se escapan del objetivo de esta asignatura, por lo que no incidiremos más en estos aspectos.

Ejercicio 41. *Un algoritmo que revolucionó la indexación y las búsquedas en las webs es el PageRank de Google. Hay numerosos recursos en Internet donde se explica el algoritmo. Nútrase de ellos para completar sus apuntes.*

La ética de los buscadores

Cuando un rastreador visita una URL, debe ceñirse a lo que el propietario de la Web le autoriza mediante el archibo robots.txt. Este archivo contiene una serie de reglas estándar que indican a los rastreadores si pueden o no indexar el contenido de dicho directorio, y si deben seguir descendiendo por el árbol o ceñirse al primer nivel.

Otra opcion es utilizar la etiqueta <meta name="robots"> en la cabecera del documento:

```
<meta name="robots" content="noindex, nofollow">
```

que en este ejemplo prohibe a los robots indexar y seguir los enlaces de la página.

Además, los buscadores deben intentar no realizar muchas peticiones en poco tiempo a un mismo servidor, para evitar caer en un DoS (Denial of Service) del servidor. Lo normal es esperar entre 100ms y 1'' entre dos peticiones consecutivas a un mismo servidor Web.

Por otro lado, está la no-ética de muchos desarrolladores, que incluyen texto oculto en la página para adquirir más relevancia o ser mejor indexados para ciertas cadenas de búsqueda. Los buscadores actuales son capaces de detectar estos "trucos", y por ejemplo, casi ignoran completamente las etiqeutas `<meta name="keyword">` o `<meta name="description">`.

Para mejorar el pagerank de las paginas hay quienes desarrollan *granjas de enlaces*, esto es, páginas artificales que no aportan nada y lo único que tienen es enlaces al sitio cuyo pagerank se desea mejorar. Obviamente, google y el resto de buscadores lo detectan al poco tiempo y penalizan este tipo de trucos.