

Lenguajes de Marcas y Sistemas de Gestión de la Información



JAVIER S. ZURDO
PABLO TOHARIA RABASCO
LAURA RAYA GONZÁLEZ



Ra-Ma®

www.ra-ma.es/cf

Lenguajes de Marcas y Sistemas de Gestión de la Información

**JAVIER S. ZURDO
PABLO TOHARIA RABASCO
LAURA RAYA GONZÁLEZ**





LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

© Javier S. Zurdo, Pablo Toharía Rabasco, Laura Raya González

© De la edición: Ra-Ma 2011

MARCAS COMERCIALES. Las designaciones utilizadas por las empresas para distinguir sus productos (hardware, software, sistemas operativos, etc.) suelen ser marcas registradas. RA-MA ha intentado a lo largo de este libro distinguir las marcas comerciales de los términos descriptivos, siguiendo el estilo que utiliza el fabricante, sin intención de infringir la marca y solo en beneficio del propietario de la misma. Los datos de los ejemplos y pantallas son ficticios a no ser que se especifique lo contrario.

RA-MA es marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso ni tampoco de cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa o de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente.

Reservados todos los derechos de publicación en cualquier idioma.

Según lo dispuesto en el Código Penal vigente ninguna parte de este libro puede ser reproducida, grabada en sistema de almacenamiento o transmitida en forma alguna ni por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro sin autorización previa y por escrito de RA-MA; su contenido está protegido por la Ley vigente que establece penas de prisión y/o multas a quienes, intencionadamente, reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

Editado por:

RA-MA Editorial

Calle Jarama, 3A, Polígono Industrial Igarsa
28860 PARACUELLOS DE JARAMA, Madrid

Teléfono: 91 658 42 80

Fax: 91 662 81 39

Correo electrónico: editorial@ra-ma.com

Internet: www.ra-ma.es y www.ra-ma.com

ISBN: 978-84-9964-101-0

Depósito Legal: M-24.441-2011

Maquetación: Antonio García Tomé

Diseño de Portada: Antonio García Tomé

Filmación e Impresión: Closas-Orcoyen, S.L.

Impreso en España

Índice

INTRODUCCIÓN	8
CAPÍTULO 1. RECONOCIMIENTO DE LAS CARACTERÍSTICAS DE LENGUAJES DE MARCAS.....	9
1.1 DEFINICIÓN Y CLASIFICACIÓN DE LENGUAJES DE MARCAS	10
1.2 TIPOS DE LENGUAJES DE MARCAS	10
1.3 EVOLUCIÓN DE LOS LENGUAJES DE MARCAS	12
1.4 ETIQUETAS, ELEMENTOS Y ATRIBUTOS.....	13
1.5 ORGANIZACIONES DESARROLLADORAS.....	14
1.6 UTILIZACIÓN DE LENGUAJES DE MARCAS EN ENTORNOS WEB	16
1.7 GRAMÁTICAS	16
1.7.1 DTD.....	16
1.7.2 Esquema XML.....	17
1.7.3 Relax NG.....	18
RESUMEN DEL CAPÍTULO.....	18
TEST DE CONOCIMIENTOS	19
CAPÍTULO 2. LENGUAJES PARA LA VISUALIZACIÓN DE INFORMACIÓN	21
2.1 EL MODELO DE OBJETOS DEL DOCUMENTO	22
2.2 HTML	24
2.2.1 Versiones.....	24
2.2.2 Etiquetas y elementos	24
2.2.3 Atributos.....	29
2.2.4 Referencias a caracteres	30
2.2.5 Comentarios	30
2.2.6 Estructura básica de un documento	31
2.2.7 Cabecera del documento	34
2.2.8 Cuerpo del documento.....	36
2.2.9 Trabajando en el cuerpo del documento	38
2.2.10 Hipertexto	47
2.2.11 Marcadores	50
2.2.12 Trabajando en la cabecera del documento	53
2.2.13 Inclusión de imágenes	58
2.2.14 Formularios	62
2.2.15 Tablas.....	74
2.2.16 Marcos.....	79
2.2.17 Capas	87
2.3 XHTML	88
2.3.1 Sintaxis	88
2.3.2 Versiones	89
2.4 HOJAS DE ESTILO	90

2.4.1	Sintaxis de CSS	90
2.4.2	Uso de CSS dentro de HTML y XHTML.....	91
2.4.3	Información de estilo en el cuerpo del documento	92
2.4.4	Información de estilo en la cabecera del documento	93
2.4.5	Información de estilo en hojas externas	95
	RESUMEN DEL CAPÍTULO	99
	EJERCICIOS PROPUESTOS.....	99
	TEST DE CONOCIMIENTOS	100
	CAPÍTULO 3. LENGUAJES PARA EL ALMACENAMIENTO Y TRANSMISIÓN DE INFORMACIÓN.....	101
3.1	TIPOS DE LENGUAJES	102
3.2	DEFINICIÓN DE XML.....	103
3.3	ESTRUCTURA Y SINTAXIS DE XML	103
3.3.1	Etiquetas, elementos y atributos	104
3.3.2	Caracteres especiales	106
3.3.3	Instrucciones de procesamiento	106
3.3.4	Comentarios y secciones CDATA.....	106
3.4	DOCUMENTOS XML BIEN FORMADOS	107
3.5	ESPACIOS DE NOMBRES	108
3.5.1	Declaración de espacios de nombres	108
3.5.2	Espacios de nombres por defecto	109
	RESUMEN DEL CAPÍTULO	110
	EJERCICIOS PROPUESTOS.....	110
	TEST DE CONOCIMIENTOS	111
	CAPÍTULO 4. DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS EN XML.....	113
4.1	DTD.....	114
4.1.1	Bloques para construir una DTD.....	116
4.1.2	Secuencias de elementos: estructura con hijos	118
4.2	ESQUEMAS	120
4.2.1	Elemento raíz	121
4.2.2	Elementos simples	122
4.2.3	Atributos	124
4.2.4	Restricciones.....	124
4.2.5	Elementos complejos	126
4.2.6	Secuencia de elementos.....	127
4.3	VALIDACIÓN DE DOCUMENTOS XML	129
4.3.1	Documentos bien formados	130
4.3.2	Documentos válidos	130
4.3.3	Herramientas para validar	131
4.4	CASO PRÁCTICO	133
	RESUMEN DEL CAPÍTULO	134
	EJERCICIOS PROPUESTOS.....	134
	TEST DE CONOCIMIENTOS	135

CAPÍTULO 5. CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS XML	137
5.1 TRANSFORMACIÓN DE DOCUMENTOS (XSL).....	140
5.2 ELEMENTOS BÁSICOS	141
5.2.1 xsl:for-each.....	141
5.2.2 xsl:value-of.....	142
5.2.3 xsl:sort	142
5.3 OPERADORES EN XSL.....	143
5.3.1 Elemento xsl:if.....	143
5.3.2 Elemento xsl:choose	144
5.4 LAS PLANTILLAS	146
5.4.1 Elemento xsl:template	146
5.5 CASO PRÁCTICO	148
RESUMEN DEL CAPÍTULO.....	149
EJERCICIOS PROPUESTOS.....	150
TEST DE CONOCIMIENTOS	150
CAPÍTULO 6. ALMACENAMIENTO DE INFORMACIÓN	153
6.1 SISTEMAS DE ALMACENAMIENTO DE LA INFORMACIÓN	156
6.2 UTILIZACIÓN DE XML PARA EL ALMACENAMIENTO DE LA INFORMACIÓN	162
6.2.1 Bases de datos relacionales.....	163
6.2.2 Transformación a XML	164
6.3 LENGUAJES DE CONSULTA Y MANIPULACIÓN	166
6.3.1 Herramienta Qizx Studio.....	168
6.3.2 Administración de librerías XML.....	171
6.4 XQUERY.....	174
6.5 CONSULTAS.....	177
6.5.1 De FLWOR a HTML.....	180
6.6 ACTUALIZACIÓN	184
6.6.1 Inserción	184
6.6.2 Reemplazo	185
6.6.3 Borrado	186
6.7 EXPORTACIÓN DE LIBRERÍAS XML.....	186
6.8 OTRAS FUNCIONES O LIBRERIAS.....	187
6.9 CASO PRÁCTICO	187
RESUMEN DEL CAPÍTULO.....	188
EJERCICIOS PROPUESTOS.....	189
TEST DE CONOCIMIENTOS	189
CAPÍTULO 7. APLICACIÓN DE LOS LENGUAJES DE MARCAS A LA SINDICACIÓN DE CONTENIDOS	193
7.1 INTRODUCCIÓN A LA SINDICACIÓN DE CONTENIDOS	194
7.2 ESTRUCTURA DE UN SISTEMA DE SINDICACIÓN	194
7.3 ESTÁNDARES ACTUALES PARA SINDICACIÓN DE CONTENIDOS	195
7.3.1 RSS 0.91 y RSS 0.92	196
7.3.2 RSS 1.0.....	198
7.3.3 RSS 2.0.....	200
7.4 SISTEMAS DE AGREGACIÓN Y DIRECTORIOS DE CANALES	204

RESUMEN DEL CAPÍTULO.....	204
EJERCICIOS PROPUESTOS.....	205
TEST DE CONOCIMIENTOS	206
CAPÍTULO 8. SISTEMAS DE GESTIÓN EMPRESARIAL.....	207
8.1 INTRODUCCIÓN A LOS ERP	208
8.2 COMPOSICIÓN DE UN ERP	209
8.3 IMPLANTACIÓN.....	210
8.4 SEGURIDAD.....	213
8.5 IMPORTACIÓN Y EXPORTACIÓN DE INFORMACIÓN.....	216
8.6 CASO PRÁCTICO	218
RESUMEN DEL CAPÍTULO.....	219
EJERCICIOS PROPUESTOS.....	219
TEST DE CONOCIMIENTOS	220
CAPÍTULO 9. HERRAMIENTAS	221
9.1 SOFTWARE PARA DISEÑO WEB	222
9.1.1 Adobe Dreamweaver	222
9.1.2 KompoZer	223
9.1.3 Expression Web	223
9.2 HERRAMIENTAS PARA LA VALIDACIÓN DE DOCUMENTOS XML VÍA WEB.....	224
9.2.1 Markup Validation Service	224
9.2.2 XmlValidation	225
9.2.3 Validome	226
9.3 SOFTWARE PARA EDITAR Y VALIDAR DOCUMENTOS XML	226
9.3.1 Serna Free	226
9.3.2 <oXygen/>	227
9.3.3 TotalEdit.....	228
9.3.4 XML Notepad	228
9.3.5 Vervet Logic	228
9.4 SISTEMAS DE GESTIÓN DE BASES DE DATOS XML NATIVAS	228
9.4.1 Qizzx	228
9.4.2 Oracle.....	229
9.4.3 Microsoft SQL Server	230
9.4.4 MySQL.....	231
9.4.5 eXist	231
9.5 SOTWARE PARA GESTIÓN EMPRESARIAL.....	232
9.5.1 OpenBravo	232
9.5.2 OpenERP	232
9.5.3 WebERP	233
9.5.4 JD Edwards EnterpriseOne.....	233
9.5.5 AP Business All in One	233
RESUMEN DEL CAPÍTULO.....	234
MATERIAL ADICIONAL.....	235
ÍNDICE ALFABÉTICO	237

A mis padres, por enseñarme a tener paciencia, perseverancia y tesón.

A mi hermano, por hacerme entrar al trapo quiera o no quiera y porque siempre aprendo algo nuevo de él.

A mi primo Roberto porque se merece estar en esta dedicatoria.

Y a mis compañeros de trabajo porque ya son para mí como mi segunda familia después de todo lo que hemos vivido.

Javier S. Zurdo

A Beatriz, mis padres y mis hermanos, por todo el apoyo que me dan siempre.

Pablo Toharia

Introducción

Este libro surge con el propósito de acercar al lector a los aspectos más importantes que encierran los lenguajes de marcas ante la creciente demanda de personal cualificado para su utilización. Con tal propósito, puede servir de apoyo también para estudiantes de los Ciclos Formativos de Grado Superior de Administración de Sistemas Informáticos en Red y Desarrollo de Aplicaciones Multiplataforma, así como para profesionales de distinto rango.

Para todo aquél que use este libro en el entorno de la enseñanza (Ciclos Formativos, Profesionales o Universidad), se ofrecen varias posibilidades: utilizar los conocimientos aquí expuestos para inculcar aspectos genéricos de los lenguajes de marcas o simplemente centrarse en trabajar a fondo alguno de ellos. La extensión de los contenidos aquí incluidos hace imposible su desarrollo completo en la mayoría de los casos.

Ra-Ma pone a disposición de los profesores una guía didáctica para el desarrollo del tema que incluye las soluciones a los ejercicios expuestos en el texto. Puede solicitarlo a editorial@ra-ma.com, acreditándose como docente y siempre que el libro sea utilizado como texto base para impartir las clases.

1

Reconocimiento de las características de lenguajes de marcas

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer qué es un lenguaje de marcas.
- ✓ Conocer los orígenes y evolución de los lenguajes de marcas.
- ✓ Conocer las organizaciones desarrolladoras de los lenguajes de marcas.
- ✓ Distinguir la clasificación de los lenguajes de marcas.
- ✓ Conocer las gramáticas de los lenguajes de marcas.

1.1 DEFINICIÓN Y CLASIFICACIÓN DE LENGUAJES DE MARCAS

Los **lenguajes de marcas** (también llamados **lenguajes de marcado**) son aquellos que combinan la información, generalmente textual, que contiene un documento con marcas o anotaciones relativas a la estructura del texto o a la forma de representarlo. El lenguaje de marcas es el que especifica cuáles serán las etiquetas posibles, dónde deben colocarse y el significado que tendrá cada una de ellas. Así mismo, la presencia de etiquetas o marcas intercaladas en el contenido hace explícita la estructura del documento o cualquier información adicional que se quiera resaltar. Por otro lado, hay que tener en cuenta que las propias etiquetas o marcas generalmente no se suelen presentar al usuario final, ya que este suele estar interesado en el propio contenido del documento.

A continuación, se muestra un ejemplo en el que mediante una serie de marcas o etiquetas se ha representado una información relativa a una noticia:



EJEMPLO 1.1

```
<noticia>
  <lugar>Madrid</lugar>
  <fecha>27/08/2010</fecha>
  <desc>Se ha inaugurado una estación de tren</desc>
</noticia>
```



¿SABÍAS QUE...?

Los lenguajes de marcas han de diferenciarse de los lenguajes de programación. El lenguaje de marcas no tiene funciones aritméticas o variables, como sí poseen los lenguajes de programación.

1.2 TIPOS DE LENGUAJES DE MARCAS

Los lenguajes de marcas se suelen dividir en tres grupos si bien hay que tener en cuenta que existen lenguajes que combinan características de más de un grupo:

- **Lenguajes orientados a presentación.** Este tipo de lenguajes son los usados tradicionalmente por los procesadores de texto como puede ser Microsoft Word® y codifican cómo ha de presentarse el documento, por ejemplo, indicando que una determinada palabra debe presentarse en fuente itálica o que se debe dejar un espacio de 10 puntos al terminar el párrafo. Generalmente las marcas de los lenguajes orientados a

presentación se ocultan al usuario lo que permite obtener un efecto WYSIWYG¹. Este tipo de lenguajes de marcas no suelen ser flexibles ni reusables.



¿SABÍAS QUE...?

En Word puedes ver las marcas pulsando el ícono ¶ de la interfaz de Microsoft Word.

- **Lenguajes procedurales.** En este tipo de lenguajes las etiquetas son también orientadas a presentación pero se integran dentro de un marco procedural que permite definir macros (secuencias de acciones) y subrutinas. Entre los ejemplos más comunes de lenguajes procedurales podemos encontrar TeX, LaTeX y Postscript.



¿SABÍAS QUE...?

La mayoría de los documentos científicos, artículos de investigación o libros técnicos que contienen fórmulas matemáticas se escriben con Latex.



¿SABÍAS QUE...?

PostScript es un lenguaje de descripción de páginas (en inglés PDL, *Page Description Language*), utilizado en muchas impresoras y, de manera usual, como formato de transporte de archivos gráficos en talleres de impresión profesional.

- **Lenguajes descriptivos.** Este tipo de lenguajes no definen qué se debe hacer con un trozo o sección del documento sino que por el contrario las marcas sirven para indicar qué es esa información, esto es, describen que es lo que se está representando. La mayoría de los lenguajes de marcas que se usan hoy en día se encuentran dentro de este grupo como por ejemplo, el SGML y sus derivados (HTML, XML, etc.) que se verán a continuación.



¿SABÍAS QUE...?

El formato COLLADA está basado en XML y se utiliza para definir escenas de modelos tridimensionales, como el de los videojuegos.

¹ What You See Is What You Get (Lo que ves es lo que obtienes)

1.3 EVOLUCIÓN DE LOS LENGUAJES DE MARCAS

Los lenguajes de marcas comenzaron a usarse a finales de la década de los 60 para poder introducir anotaciones dentro de documentos electrónicos, de la misma forma que se hacía cuando la documentación estaba en papel. De esta posibilidad de incorporar marcas es de donde reciben su nombre. Es en esas fechas cuando se estandariza el lenguaje **SGML** (*Standard Generalized Markup Language*), que es un descendiente directo del lenguaje **GML** propuesto por IBM. Este lenguaje surgió para permitir compartir información por parte de sistemas informáticos. Este estándar tuvo una gran aceptación pero no consiguió asentarse del todo debido principalmente a su complejidad lo que provocaba que el software que usaría SGML terminaba siendo excesivamente extenso y complejo.

A finales de los 80 dentro del CERN (*Conseil Européen pour la Recherche Nucléaire*) se creó un lenguaje de marcado pensado para compartir información usando las redes de computadores y, de forma más general, a través de Internet. Este lenguaje se basaba en algunos principios de SGML y lo denominaron HTML (*Hyper-text Markup Language*). La aparición de este lenguaje supuso de alguna manera una revolución en la forma de compartir información, gracias principalmente a la sencillez de sus sintaxis y del software necesario para interpretarlo. En poco tiempo el lenguaje HTML se extendió y empezó a crecer de forma en ocasiones descontrolada y casi siempre influenciado por razones meramente comerciales.

A mediados de los años 90 el consorcio **W3C** (*World Wide Web Consortium*) comenzó una iniciativa para intentar dotar a la *web* de un lenguaje más potente y que pudiera dar una estructurar semántica a la misma. Para ello se marcaron el objetivo de crear un nuevo lenguaje de marcas basado en SGML y que fuera sencillo como HTML. Finalmente, en el 1998, W3C hizo público un nuevo estándar que denominaron XML (*eXtended Markup Language*), más sencillo que SGML y más potente que HTML.



¿SABÍAS QUE...?

HTML es el lenguaje de marcas predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

ACTIVIDADES 1.1

- Abra la página www.google.es en su explorador. Pulse el botón derecho del ratón y marque la opción "Ver código fuente".
- Busque información sobre el lenguaje de marcas XHTML.
- ¿En qué se diferencia un lenguaje de marcas a un lenguaje de programación?
- Visite la web http://es.wikipedia.org/wiki/Categoría:Lenguajes_de_descripción.

Eche una ojeada a los diferentes lenguajes de marcas que existen.

1.4 ETIQUETAS, ELEMENTOS Y ATRIBUTOS

Existen tres términos comúnmente usados para describir las partes de un documento de lenguajes de marcas: etiquetas, elementos y atributos.

Una **etiqueta** (*tag*) es un texto que va entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de inicio (como <nombre>) y etiquetas de fin (como </nombre>).

Los **elementos** representan estructuras mediante las que se organizará el contenido del documento o acciones que se desencadenan cuando el programa navegador interpreta el documento. Constan de la etiqueta de inicio, la etiqueta de fin y de todo aquello que se encuentra entre ambas.

Algunos elementos no tienen contenido. Se les denomina elementos vacíos y no deben llevar etiqueta de fin.

Un **atributo** es un par nombre-valor que se encuentra dentro de la etiqueta de inicio de un elemento e indican las propiedades que pueden llevar asociadas los elementos.



EJEMPLO 1.2

Fíjese en el texto siguiente:

```
<direccion>
  <nombre>
    <titulo>Mrs.</titulo>
    <nombre> Mary </nombre>
    <apellidos> McGoon </apellidos>
  </nombre>
  <calle> 1401 Main Street </calle>
  <ciudad estado="NC"> Anytown</ciudad>
  <codigo-postal> 34829 </codigo-postal>
</direccion>
```

En el ejemplo anterior, el elemento <nombre> contiene tres elementos hijos: <titulo>, <nombre> y <apellidos> y *estado* es un atributo del elemento <ciudad>.



En el capítulo 2 se profundizará más sobre estos tres conceptos.

ACTIVIDADES 1.2



► Fíjese en el siguiente texto.

```
<noticia>
  <lugar>Madrid</lugar>
  <fecha>27/08/2010</fecha>
  <desc>Se ha inaugurado una estación de tren</desc>
</noticia>
```

Indique el nombre de una etiqueta, de dos elementos y de algún atributo.

► Escriba un texto que contenga etiquetas, elementos y atributos.

1.5 ORGANIZACIONES DESARROLLADORAS

Dentro de las organizaciones que se han encargado de desarrollar los lenguajes de marcas se encuentran:

Organización Internacional para la Estandarización (ISO, International Organization for Standardization). Se formó después de la Segunda Guerra Mundial (23 de febrero de 1947) y es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

Es una red de los institutos de normas nacionales de 163 países, sobre la base de un miembro por país, con una Secretaría Central en Ginebra (Suiza) que coordina el sistema.

Las normas desarrolladas por ISO son voluntarias, ya que es un organismo no gubernamental y no depende de ningún otro organismo internacional, por tanto, no tiene autoridad para imponer sus normas a ningún país. El contenido de los estándares está protegido por derechos de copyright y para acceder a ellos el público en general ha de comprar cada documento.

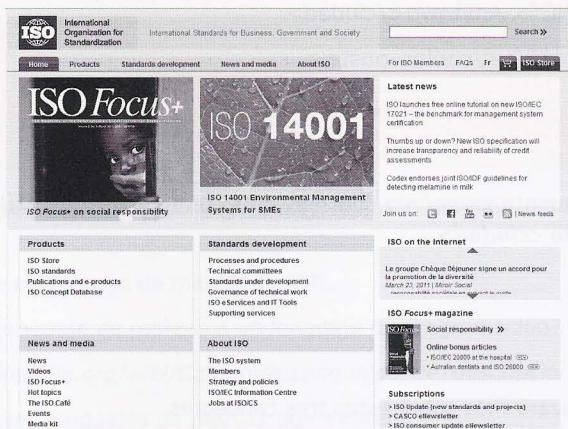


Figura 1.1. Página web de la ISO

Esta organización después del éxito que tuvo **GML** y, después de un largo proceso, publicó en 1986 el *Standard Generalized Markup Language (SGML)* con rango de Estándar Internacional con el código **ISO 8879**.

- **World Wide Web Consortium (W3C).** El W3C se creó en 1994 por Tim Berners-Lee en el MIT, actual sede central del consorcio. Posteriormente se unió, en abril de 1995, el INRIA en Francia, reemplazado por el ERCIM en el 2003 como el huésped europeo del consorcio y la Universidad de Kei (Shonan Fujisawa Campus) en Japón en septiembre de 1996 como huésped asiático. Su función principal es tutelar el crecimiento y organización de la web.

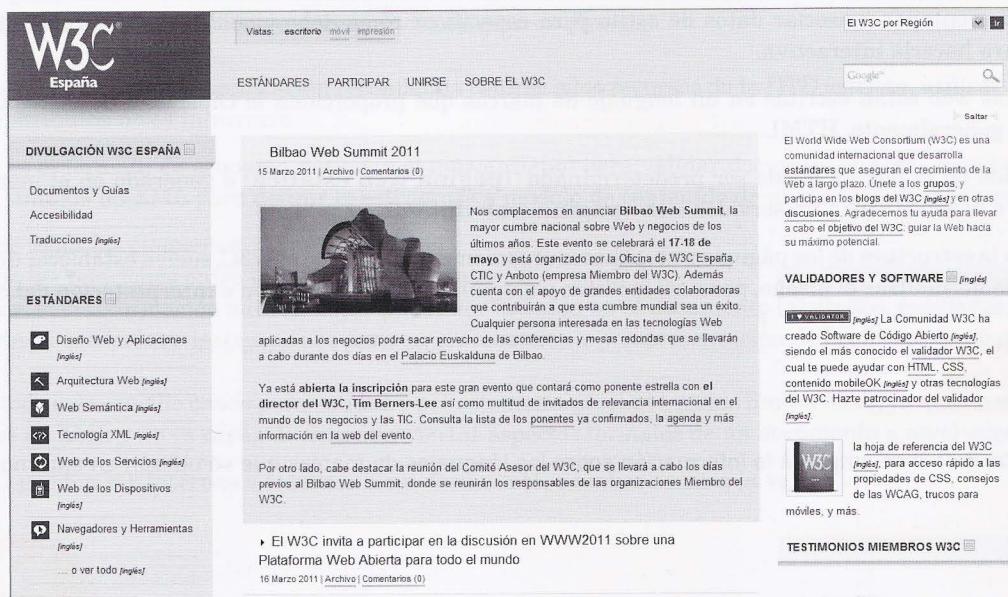


Figura 1.2. Página web de la W3C

Su primer trabajo fue normalizar el lenguaje HTML, el lenguaje de marcas con el que se escriben las páginas web. Al crecer el uso de la web, crecieron las presiones para ampliar el HTML. El W3C decidió que la solución no era ampliar el HTML, sino crear unas reglas para que cualquiera pudiera crear lenguajes de marcas adecuados a sus necesidades, pero manteniendo unas estructuras y sintaxis comunes que permitieran compatibilizarlos y tratarlos con las mismas herramientas. Ese conjunto de reglas es el XML, cuya primera versión se publicó en 1998.

ACTIVIDADES 1.3

- Busca información en Internet sobre las organizaciones ISO y W3C.

1.6 UTILIZACIÓN DE LENGUAJES DE MARCAS EN ENTORNOS WEB

Una **página web** es un documento electrónico adaptado para la *World Wide Web* que, normalmente, forma parte de un sitio web.

Está compuesta, principalmente, por información (solo texto o módulos multimedia) así como por hiperenlaces; además, puede contener o asociar datos de estilo para especificar cómo debe visualizarse, y también aplicaciones embebidas para hacerla interactiva.

Las páginas web están escritas en un lenguaje de marcas que proporciona la capacidad de manejar e insertar hiperenlaces, generalmente, HTML.

El contenido de la página puede ser predeterminado (**página web estática**) o generado en el momento de su visualización o al solicitarla a un servidor web (**página web dinámica**).

Respecto a la estructura de las páginas web, algunos organismos, en especial el W3C, suelen establecer directivas con la intención de normalizar el diseño, para así facilitar y simplificar la visualización e interpretación del contenido.



En el Capítulo 2 se ampliará la información sobre los lenguajes de marcas que se utilizan en entorno web.

1.7 GRAMÁTICAS

Todo documento de un lenguaje de marcas tiene en común una gramática que define el marcado permitido en esa clase, el marcado requerido y cómo debe ser utilizado dicho marcado en la instancia del documento.

1.7.1 DTD

El estándar define esta gramática mediante la **DTD (Definición de Tipo de Documento)** que establece las reglas de formación del lenguaje formal, es decir, qué combinaciones de símbolos elementales son sintácticamente correctas.

En la DTD se identifica la estructura del documento, es decir, aquellos elementos que son necesarios en la elaboración de un documento o un grupo de documentos estructurados de manera similar. Contiene las reglas de dichos elementos: el nombre, su significado, dónde pueden ser utilizados y qué pueden contener.

La especificación del W3C para HTML 4.0 contempla tres DTD:

- **DTD estricta (HTML 4.0 Strict DTD)**: incluye todos los elementos y atributos que no han sido declarados “desaprobados” (*deprecated*), interpretando la expresión en el sentido de que no se recomienda ya su uso proponiéndose nuevos y mejores recursos para hacer lo mismo.
- **DTD transicional o flexible -loose- (HTML 4.0 Transitional DTD)**: incluye todo lo que la anterior más los elementos y atributos desaprobados (*deprecated*).
- **DTD para documentos con marcos (HTML 4.0 Frameset DTD)**: engloba todo lo incluido en la transicional más lo relativo a la creación de documentos con marcos (*frames*).

Recuerde que aunque la especificación recomienda ceñirse a los recursos de la DTD estricta, utilizar el resto de los elementos y atributos no es incorrecto.

La DTD es el formato de esquema nativo (y el más antiguo) para validar documentos XML, heredado de SGML. Utiliza una sintaxis no-XML para definir la estructura o modelo de contenido de un documento XML válido:

- ✓ Define todos los elementos.
- ✓ Define las relaciones entre los distintos elementos.
- ✓ Proporciona información adicional que puede ser incluida en el documento (atributos, entidades, notaciones).
- ✓ Aporta comentarios e instrucciones para su procesamiento y representación de los formatos de datos.

Es el método más sencillo usado para validar, y por esta razón presenta varias limitaciones, ya que no soporta nuevas ampliaciones de XML y no es capaz de describir ciertos aspectos formales de un documento a nivel expresivo.

Las DTD pueden ser internas o externas a un documento, o ambas cosas a la vez.



En el Capítulo 4 se ampliará la información sobre los DTD.

1.7.2 ESQUEMA XML

XML Schema es la evolución de la DTD descrita por el W3C, también denominado XSD (*XML Schema Definition*). Es un lenguaje de esquema más complejo y más potente, basado en la gramática para proporcionar una potencia expresiva mayor que la DTD. Utiliza sintaxis XML, cosa que le permite especificar de forma más detallada un extenso sistema de tipos de datos. A diferencia de las DTD, soporta la extensión del documento sin problemas.

A la hora de la validación del documento, la utilización de XSD supone un gran consumo en recursos y tiempo debido a su gran especificación y complejidad en la sintaxis (los esquemas son más difíciles de leer y de escribir).

Después de validar el documento con XML Schema, es posible expresar su estructura y contenido en términos del modelo de datos usado por el esquema de validación. Esta funcionalidad, conocida como *Post-Schema-Validation Infoset* (PSVI), se puede utilizar para transformar el documento en una jerarquía de objetos, a los cuales se puede acceder a través de un lenguaje de programación orientada a objetos (OOP).

El modelo de datos de XML Schema incluye:

- ✓ El vocabulario (nombres de elemento y atributo).
- ✓ El contenido modelo (relaciones y estructura).
- ✓ Los tipos de datos.



En el Capítulo 4 se ampliará la información sobre los esquemas XML.

1.7.3 RELAX NG

RELAX NG es un lenguaje de esquema basado en la gramática, muy intuitivo y más fácil de entender que el XML Schema. Tiene un alto poder expresivo, ya que, por ejemplo, permite validar elementos intercalados que pueden aparecer en cualquier orden.

Las aplicaciones de definición de documentos y validación para *RELAX NG* son más sencillas que las de *XML Schema*, haciéndolo más fácil de utilizar e implementar.

RELAX NG se ha convertido recientemente en un estándar ISO como la parte 2 de **DSDL** (*Document Schema Definition Language*).



RESUMEN DEL CAPÍTULO



En este capítulo se ha llevado a cabo una breve descripción de lo que es un lenguaje de marcas.

Se han indicado los orígenes y la evolución de los distintos lenguajes de marcas.

Se ha descrito de forma genérica lo que es una etiqueta y el concepto de elemento y de atributo.

Se ha hablado sobre las dos organizaciones desarrolladoras de los lenguajes de marcas (ISO y W3C).

Se ha descrito una sencilla clasificación de los lenguajes de marcas.

Se ha indicado lo que es la gramática de los lenguajes de marcas, indicando lo que son los DTD, XML Esquema y Relax NG.



TEST DE CONOCIMIENTOS



1 Indicar cuál de las siguientes afirmaciones es cierta:

- a) LaTeX es un lenguaje orientado a presentación.
- b) LaTeX es un lenguaje procedural.
- c) LaTeX es un lenguaje descriptivo.

2 Indicar cuál de las siguientes afirmaciones es cierta:

- a) El consorcio W3C comenzó una iniciativa para intentar dotar a la *web* de un lenguaje más potente y que pudiera dar una estructurar semántica a la misma.
- b) ISO comenzó una iniciativa para intentar dotar a la *web* de un lenguaje más potente y que pudiera dar una estructurar semántica a la misma.
- c) El lenguaje **GML** fue propuesto por el consorcio W3C.

3 Indicar cuál de las siguientes afirmaciones es cierta:

- a) Algunos elementos no tienen contenido. Se les denomina elementos vacíos y no deben llevar etiqueta de fin.
- b) Todos los elementos tienen que llevar obligatoriamente etiqueta de inicio y de final.
- c) Un **atributo** es un nombre que se encuentra dentro de la etiqueta de inicio de un elemento.

4 Indicar cuál de las siguientes afirmaciones es falsa:

- a) En la DTD se identifica la estructura del documento.
- b) Las DTD pueden ser internas o externas a un documento, o ambas cosas a la vez.
- c) En la DTD no se definen todos los elementos.

5 Indicar cuál de las siguientes afirmaciones es falsa:

- a) XML Schema es la evolución de la DTD descrita por el W3C.
- b) A XML Schema también se le denomina XSD.
- c) A XML Schema también se le denomina RELAX NG.

6 Indicar cuál de las siguientes afirmaciones es falsa:

- a) El modelo de datos de XML Schema incluye el vocabulario.
- b) El modelo de datos de XML Schema no incluye los tipos de datos.
- c) El modelo de datos de XML Schema incluye los elementos.

2

Lenguajes para la visualización de información

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer lo que es el modelo de objetos del documento.
- ✓ Conocer el uso de los lenguajes de marcas para presentación de información en la web.
- ✓ Aprender los rudimentos de HTML así como las distintas versiones existentes.
- ✓ Introducir XHTML y su relación con HTML.
- ✓ Entender cómo se separa la información de estilo y la información estructural.
- ✓ Introducir los conceptos básicos de CSS y su sintaxis.

Una de las aplicaciones donde los lenguajes de marcas han tenido un gran éxito es en la presentación de información y en especial dentro de entornos web. En este sentido lenguajes como HTML, ampliamente usado en los albores de Internet, o XHTML, son los lenguajes de marcas predominantes para visualización de información en la web.

En este capítulo se van a exponer los lenguajes de marcas más usados en la web (además del XML) así como algunas herramientas relacionadas para representación de la información.

2.1 EL MODELO DE OBJETOS DEL DOCUMENTO

El **Modelo de Objetos del Documento (DOM)** es un **API (Application Programming Interface)** estándar del W3C para documentos HTML y XML. Proporciona una representación estructural del documento que permite la modificación de su contenido o su presentación visual. Esencialmente, comunica las páginas web con los scripts o los lenguajes de programación.

Con el modelo de objetos del documento los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando el Modelo de Objetos del Documento, salvo algunas excepciones. En particular, aún no se han especificado las interfaces DOM para los subconjuntos internos y externos de XML.



Un **API** es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece una biblioteca para ser utilizado por otro software como una capa de abstracción.

Una página web es un documento HTML que es interpretado por los navegadores en forma gráfica, permitiendo también el acceso al código.

El modelo de objetos del documento (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos para que un programa JavaScript pueda actuar sobre ellos.

El DOM permite acceder a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente vea el siguiente ejemplo:



EJEMPLO 2.1

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>
<ul>
<li>Primera entrada en la lista</li>
<li>Segunda entrada en la lista</li>
</ul>
</body>
```

Como puede ver en el ejemplo, el elemento *a* se encuentra localizado dentro del elemento *p* del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo *p*, de manera similar, *p* es el nodo padre. Los dos nodos *li* son hijos del mismo padre, llamándose nodos hermanos o, simplemente, hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos, normalmente, están asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como <p> por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

En el DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol (para ser más precisos, es más bien como un “bosque” o una “arboleda”, que puede contener más de un árbol). Sin embargo, el DOM no especifica que los documentos deban ser *implementados* como un árbol o un bosque, ni tampoco especifica cómo deben implementarse las relaciones entre los objetos. El DOM es un modelo lógico que puede implementarse de cualquier manera que se considere conveniente. En esta especificación, se usa el término *modelo de estructura* para describir la representación en forma de árbol de un documento, evitando la utilización de términos tales como “árbol” o “bosque” para evitar la implicación de una implementación en particular. Una propiedad importante de los modelos de estructura del DOM es su *isomorfismo estructural*: si dos implementaciones cualesquiera del modelo de objetos del documento se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

Se eligió el nombre *modelo de objetos del documento* porque es un *modelo de objetos* en el sentido tradicional del diseño orientado a objetos: 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. Como modelo de objetos, el 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.

Tradicionalmente, la estructura de los documentos SGML se ha representado mediante un modelo de datos abstractos, no con un modelo de objetos. En un modelo de datos abstractos, el modelo se centra en los datos. En los lenguajes de programación orientados a objetos, los datos se encapsulan en objetos que ocultan los datos, protegiéndolos de su manipulación directa desde el exterior. Las funciones asociadas con estos objetos determinan cómo pueden manipularse los objetos, y son parte del modelo de objetos.

El modelo de objetos del documento consiste actualmente de dos partes: el núcleo del DOM y el DOM HTML. El núcleo del DOM representa la funcionalidad usada para los documentos XML, y también sirve de base para el DOM HTML. Una implementación conforme del DOM debe implementar todas las interfaces fundamentales del capítulo sobre el núcleo con la semántica definida. Además, debe implementar el DOM HTML, las interfaces extendidas (XML) o ambas, con la semántica definida.

ACTIVIDADES 2.1



- Vaya a la página <http://www.w3.org/DOM> y obtenga más información sobre el DOM.

2.2 HTML

El lenguaje de marcas **HTML** (*Hyper Text Markup Language*) surgió por la complejidad del lenguaje SGML creándose un lenguaje mucho más simple y adaptado expresamente al cometido de representar contenido para la web. En este sentido, el número de etiquetas del que se dotó a HTML era considerablemente reducido, lo que hacía que su curva de aprendizaje fuera bastante rápida.

A continuación, se detallarán algunas características básicas sobre la sintaxis de los documentos HTML.

2.2.1 VERSIONES

HTML fue por primera vez plasmado como HTML 2.0 en un RFC en 1995 y este RFC fue dando lugar a sucesivos RFC hasta la aparición en 1997 de un RFC que propia HTML 3.2 y que fue publicado como recomendación por parte del consorcio W3C.

Poco después, en 1998, W3C publicó una nueva versión denominada HTML 4.0 en tres versiones distintas (*strict*, *transitional* y *frameset* que se explicarán en el apartado de XHTML por su similitud) y, enseguida, se publicó la versión 4.1 para incorporar algunas erratas.

En 2008 el consorcio W3C publicó un borrador de HTML5 en el cual se encuentran trabajando hoy en día.

Para que los navegadores sepan que versión de HTML es el documento que debe presentar, es necesario incluir una cabecera DOCTYPE que lo identifique. A continuación, se muestra un ejemplo para HTML 4.01 Transitional:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```



¿SABÍAS QUE...?

Un RFC (del inglés *Request For Comments*) son unas notas sobre tecnologías asociadas a Internet que recogen propuestas para nuevos protocolos o estándares. En un principio servían también para obtener realimentación acerca de la propuesta, de ahí su nombre. Hoy en día el nombre original no tiene demasiado sentido ya que para llegar a ser propuesto como RFC se debe seguir un proceso muy estricto que asegura la calidad de la propuesta y, por lo tanto, recibirá pocos o ningún comentario.

2.2.2 ETIQUETAS Y ELEMENTOS

HTML, tal y como se ha explicado en el apartado anterior, es un lenguaje orientado a la presentación de información para la web. Por esta razón, las etiquetas que usa están condicionadas con este hecho. HTML es un lenguaje de marcado que usa el mismo tipo de etiquetas presentadas en XML. A continuación, se hará una introducción a las **etiquetas** de HTML más importantes.

En primer lugar, se presenta en el siguiente ejemplo la estructura básica de un documento HTML, que consta del par `<html></html>` que contienen todo el resto del documento. Dentro de este par de etiquetas se pueden encontrar los dos siguientes **elementos** con sus pares de etiquetas:

- `<head></head>`: contiene información relativa a la página web, como puede ser el título, metadatos, estilos, etc.
- `<body></body>`: contiene la información que se desea presentar.

```
<html>
<head>
<title>Pagina ejemplo HTML</title>
</head>
<body>
<p>Primer párrafo de ejemplo</p>
<p>Segundo párrafo de ejemplo</p>
<imgsrc="imagen1.jpg"alt="Imagen ejemplo">
</body>
</html>
```

Dentro de las etiquetas `<body></body>` se representa el contenido del propio documento. Este puede contener distintos tipos de elementos, de los cuales se van a enumerar los más importantes a continuación:

- `<h1></h1>`: define un encabezado de tipo 1. Se pueden usar desde `<h1>` hasta `<h6>` para representar un árbol de encabezados siendo los de tipo 1 los de mayor nivel.
- `<p></p>`: sirve para representar un párrafo de texto.
- `Enlace`: permite definir un hipervínculo a una URL. El contenido que aparece entre `<a>` y `` será el que se represente en la página web.
- `src="imagen.jpg" width="200" height="100" />`: permite incorporar una imagen en el contenido que se va a representar. El atributo `src` sirve para indicar la URL donde se encuentra la imagen, mientras que el atributo `alt` permite definir un texto alternativo a la imagen que se mostrará en navegadores sin soporte gráfico. Además, se pueden usar los atributos `width` y `height` para representar la imagen con un determinado tamaño.

```
<imgsrc="imagen.jpg" width=100 height=200 alt="Imagen">
```

- `
`: inserta un salto de línea.
- `` y `<i></i>`: sirven para insertar texto en negrita y en cursiva respectivamente.
`Texto en negrita`
`<i>Texto en cursiva</i>`
- ``: define listas no numeradas en las que cada nuevo ítem se encuentra delimitado por ``. Para definir listas ordenadas se puede usar la etiqueta ``.

```
<ul>
<li>Item 1</li>
<li>Item 2</li>
</ul>
```

- <table></table>: sirve para escribir una tabla en el contenido que se desea representar. Durante un tiempo las tablas se usaban para maquetar lo que está desaconsejado. Las etiquetas <tr></tr> sirven para empezar una nueva fila mientras que las etiquetas <td></td> sirven para delimitar una nueva columna.

```
<table>
<tr>
<td>Fila 1 Columna 1</td>
<td>Fila 1 Columna 2</td>
</tr>
<tr>
<td>Fila 2 Columna 1</td>
<td>Fila 2 Columna 2</td>
</tr>
</table>
```

- <form></form>: permite crear formularios usando las etiquetas <input>, <option>, <textarea>, <button>, etc. Los atributos *action* y *method* permiten definir la URL a la que se enviarán los datos del formulario y el método de envío (*get* o *post*).

```
<form action="procesar.cgi" method="get">
Apellidos: <input type="text" name="apellidos"><br>
Nombre: <input type="text" name="nombre"><br>
Comentario: <textarea name="comentario">
Escriba su comentario
</textarea><br>
<input type="submit" value="Enviar">
</form>
```



Además de los elementos y atributos mostrados a modo de ejemplo previamente, en HTML existen otros elementos y atributos. Para un conocimiento completo de HTML se recomienda consultar la web del consorcio W3C.

PRÁCTICA 2.1

La lista: Elementos, Atributos, Referencias a caracteres y Comentarios, se representaría en lenguaje HTML con el siguiente texto:

```
<ul>
<li>Elementos</li>
<li>Atributos</li>
<li>Referencias a caracteres</li>
<li>Comentarios</li>
</ul>
```

Algunos elementos pueden introducirse omitiendo la etiqueta de fin, por ejemplo el elemento **P**, que representa un párrafo, o el elemento **LI** que representa un ítem de lista (delimitada en el ejemplo anterior por las etiquetas de inicio y fin del elemento **UL** que identifica un determinado tipo de lista). Así, se podría reescribir el texto anterior:

```
<ul>
  <li>Elementos
  <li>Atributos
  <li>Referencias a caracteres
  <li>Comentarios
</ul>
```

PRÁCTICA 2.2

Fíjese en el documento HTML siguiente:

```
<HTML>
<HEAD>
<TITLE>Ante la Ley</TITLE>
</HEAD>
<BODY>
<P>Ante la ley hay un guardián. Un campesino se presenta frente a este guardián y solicita que le permitan entrar en la ley. Pero el guardián contesta que por ahora no puede dejarlo entrar. El hombre reflexiona y pregunta si más tarde lo dejarán entrar.
<P>—Es posible —dice el portero—, pero no ahora.
<P>[...]
<P><A href="http://huizen.dds.nl/~nfkk/">Franz Kafka</A>
Traducción de J.R. Wilcock
</BODY>
</HTML>
```

Ahora cópielo a un fichero de texto, guárdelo, cambie su extensión a HTML y ábralo con su **navegador**. Verá algo similar a lo siguiente:

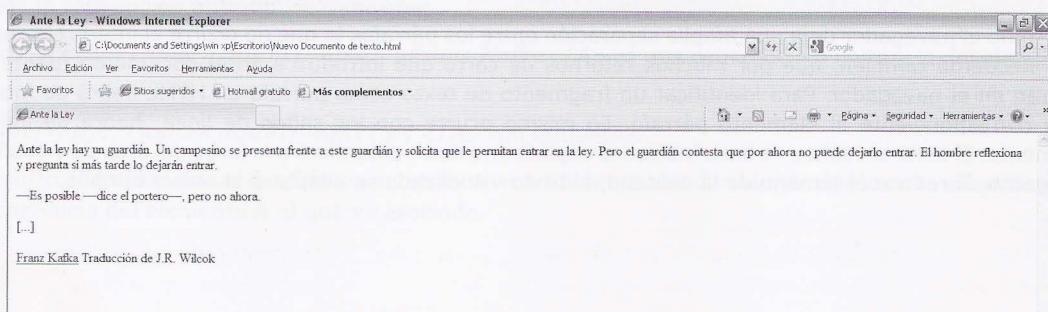


Figura 2.1. Árbol de directorios

Algunos elementos no tienen contenido. Se les llama elementos vacíos y no deben llevar etiqueta de fin. El elemento **BR** es un ejemplo de elemento vacío. Su función es provocar un salto de línea dentro de un mismo párrafo.

Nuestro primer ejemplo puede mejorarse si se reduce a uno los tres primeros párrafos y se utiliza el elemento **BR**. También se introducirá un salto de línea para separar la referencia al autor de la referencia al traductor del texto:

```
<HTML>
<HEAD>
<TITLE>Ante la Ley</TITLE>
</HEAD>
<BODY>
<P>Ante la ley hay un guardián. Un campesino se presenta frente a este guardián y solicita que le permitan entrar en la ley. Pero el guardián contesta que por ahora no puede dejarlo entrar. El hombre reflexiona y pregunta si más tarde lo dejarán entrar.<BR> —Es posible —dice el portero—, pero no ahora.<BR> [...]<BR>
<P><A href="http://huizen.dds.nl/~nffkk/">Franz Kafka</A><BR>
Traducción de J.R. Wilcock
</BODY>
</HTML>
```

El documento se visualizará en su navegador tal como se reproduce en la figura siguiente:

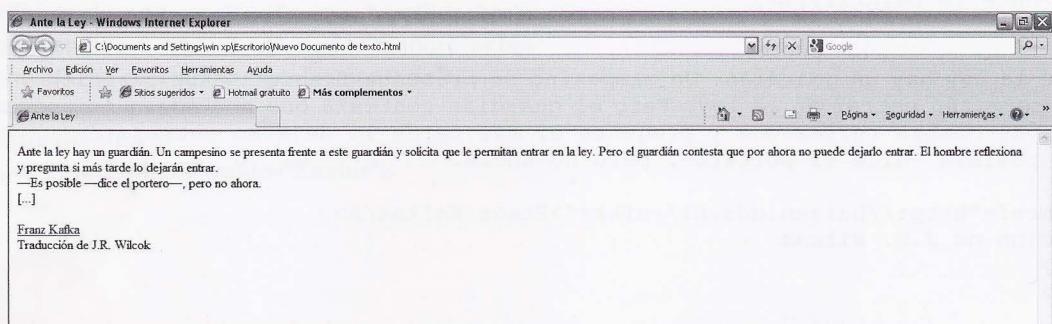


Figura 2.2

Observe como el navegador deja una amplia separación entre los párrafos lo que no ocurre al interpretar un salto de línea. Recuerde también, que por muchos retornos de carro que introduzca en el documento fuente no se visualizarán en el navegador. Para identificar un fragmento de texto como párrafo en HTML habrá de introducir la marca correspondiente al elemento párrafo. Lo mismo ocurre con los saltos de línea, habrá de indicarlos expresamente. El texto sin saltos de línea indicados explícitamente se extenderá a lo largo del área de visualización del navegador. Si reduce el tamaño de la ventana, el texto visualizado se adaptará al nuevo tamaño.

ACTIVIDADES 2.2

- Fijándose en las prácticas anteriores, escriba un documento HTML y observe cómo queda en su navegador.

2.2.3 ATRIBUTOS

Los elementos pueden llevar asociadas propiedades llamadas **atributos**. Los atributos pueden tener valores. Los pares atributo/valor siempre se colocan antes del final de la etiqueta de inicio del elemento –es decir, antes del signo ‘>’– nunca en la etiqueta de fin. Puede utilizarse cualquier número de pares atributo/valor separados por espacios. Los atributos pueden sucederse en cualquier orden.



EJEMPLO 2.2

En el ejemplo siguiente se va a utilizar el atributo **href** con el elemento A:

```
<A href="http://huizen.dds.nl/~nfkk/"Franz Kafka</A>
```

El nombre del atributo va seguido por el signo “=” y el valor del atributo. El valor de los atributos se delimita por comillas dobles (ASCII decimal 34) o simples (ASCII decimal 39). Las comillas simples pueden formar parte del valor del atributo cuando éste se delimita por comillas dobles y viceversa. En algunos casos puede prescindirse de las comillas pero la especificación recomienda utilizarlas siempre.

Los nombres de los atributos pueden escribirse indiferentemente con mayúsculas o minúsculas. Los valores de los atributos generalmente también.

El W3C hace una advertencia explícita recomendando no confundir elementos y etiquetas. Dado que el uso del término etiquetas está tan extendido en la literatura sobre HTML, para evitar confusiones se va a adoptar la siguiente convención:

- Al utilizar el término *elemento* nos vamos a referir tanto a sus etiquetas de inicio y fin (si ésta existe) como a su contenido (si éste existe).
- Al utilizar el término *etiqueta* nos vamos a referir a las etiquetas de inicio y/o fin y, en general, a todo lo que encontramos entre cada par de símbolos ‘<’ y ‘>’.

Siguiendo la convención anterior, se tiene que:

- `Franz Kafka` es una ocurrencia o instancia del elemento de tipo “A”.
- `` y `` son etiquetas.
- `href="http://huizen.dds.nl/~nfkk/` es un atributo con un determinado valor, incluido en la etiqueta de inicio de la ocurrencia del elemento A al que va asociado.



Para modificar un documento HTML que ya ha escrito, cambie su extensión a *txt*, modifique el texto que desee, guárdelo, cambie su extensión a HTML y ábralo con su **navegador**.

ACTIVIDADES 2.3



- Modifique el documento HTML de la actividad anterior cambiando el atributo del elemento A y observe cómo queda en su navegador.

2.2.4 REFERENCIAS A CARACTERES

Las **referencias a caracteres** son nombres numéricos o simbólicos de caracteres que pueden incluirse en un documento HTML. Comienzan con el signo “&” y terminan con un punto y coma “;”. Resultan útiles y a veces imprescindibles para representar caracteres especiales u otros que no pueden introducirse directamente desde el editor de textos o herramienta de autor utilizada para crear el documento HTML:

Carácter	Referencia a entidades	Referencia numérica
á	á	á
é	é	é
í	í	í
ó	ó	ó
ú	ú	ú



EJEMPLO 2.3

Fíjese como se pone el acento en la palabra *Programación* en un documento HTML.

```
<TITLE>Ejemplo 1 - Programaci&oacute;n JavaScript</TITLE>
```

ACTIVIDADES 2.4



- Modifique el documento HTML de la actividad anterior añadiendo acentos utilizando referencias a caracteres y observe cómo queda en su navegador.

2.2.5 COMENTARIOS

En HTML, los comentarios tienen la siguiente sintaxis:

```
<!-comentario en una linea -->  
<!-primera linea del comentario,  
segunda linea del mismo comentario -->
```

Los comentarios serán ignorados por el navegador y no se visualizarán. Debe evitarse introducir cadenas de guiones en el texto del comentario.

Quizás se pregunte: si esto es todo lo que puedo encontrar en un documento HTML, ¿dónde quedan las imágenes, los sonidos, etc., que constantemente observo en las páginas web? Todo lo mencionado son recursos a los que se puede hacer referencia desde el documento HTML. El documento HTML contiene referencias a esos recursos e instrucciones para que el navegador los recupere y ubique en el documento cuando es interpretado, pero son recursos que no se almacenan con el propio documento HTML. De hecho, podría hacer referencia a imágenes u otros recursos ubicados en cualquier servidor del mundo que, una vez descargados de la red, se visualizarían en su página web.

ACTIVIDADES 2.5



- Modifique el documento HTML de la actividad anterior añadiendo comentarios y observe cómo queda en su navegador.

2.2.6 ESTRUCTURA BÁSICA DE UN DOCUMENTO

Todo documento HTML consta de tres partes:

1. Una declaración de la versión de HTML con la que se ha creado el documento.
2. Una cabecera con información acerca del documento u otros datos que no pueden considerarse parte de su contenido.
3. Un cuerpo con el contenido real del documento.

Las partes 2 y 3 ya se han mencionado en los ejemplos anteriores, no así la primera. Un documento sin información sobre la versión de HTML a la que se ajuste será, generalmente, interpretado por el navegador sin ningún problema pero debe aparecer si se quiere que el documento sea estrictamente correcto (especialmente si se crea pensando colocarlo en Internet).

Como ya se ha visto anteriormente, la sección de cabecera se delimita con el elemento **HEAD**. El cuerpo se delimita con el elemento **BODY** o con el elemento **FRAMESET** en el caso de tratarse de un documento con marcos.



Para obtener más información sobre los marcos, vea el epígrafe correspondiente de este capítulo.

Aunque no sea obligatorio, si se omite será inferida por el navegador, delimita siempre ambos elementos con el elemento **HTML**.



EJEMPLO 2.4

El ejemplo siguiente incluye la declaración sobre la versión de HTML con la que ha sido creado:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"  
  "http://www.w3.org/TR/REC-html40/strict.dtd">  
<HTML>  
<HEAD>  
<TITLE>Ante la Ley</TITLE>  
</HEAD>  
<BODY>  
<P>Ante la ley hay un guardián. Un campesino se presenta frente a este guardián y  
solicita que le permitan entrar en la ley. Pero el guardián contesta que por ahora  
no puede dejarlo entrar. El hombre reflexiona y pregunta si más tarde lo dejarán  
entrar.<BR>  
-Es posible -dice el portero-, pero no ahora.<BR>  
[...]<BR>  
<P><A href="http://huizen.dds.nl/~nfkk/">Franz Kafka</A><BR>  
Traducción de J.R. Wilcock  
</BODY>  
</HTML>
```

En HTML 4, al estar basado en **SGML**, la declaración **<!DOCTYPE>** hace referencia a la DTD a la que se ajusta el documento. Dado que la recomendación para el HTML 4 contempla tres DTD se podrán utilizar tres declaraciones:

- Para documentos que se ajustan a la **DTD estricta**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"  
  "http://www.w3.org/TR/REC-html40/strict.dtd">
```



Esta DTD incluye todos los elementos y atributos que no han sido declarados "desaprobados" (*deprecated*).

- Para documentos que se ajustan a la **DTD transicional**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"  
  "http://www.w3.org/TR/REC-html40/loose.dtd">
```



Esta DTD incluye lo que la anterior más los elementos y atributos desaprobados (*deprecated*).

■ Para documentos con marcos:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
```

```
"http://www.w3.org/TR/REC-html40/frameset.dtd">
```



Esta DTD engloba todo lo incluido en la transicional más lo relativo a la creación de documentos con marcos (*frames*).

Las últimas dos letras de la declaración indican el idioma de la DTD. Para HTML este será siempre el inglés (“EN”).

El URI incluido en cada declaración permitiría a los navegadores descargar la DTD y cualquier juego de entidades que se necesitase para interpretar el documento.



Un **URI** (*Uniform Resource Identifier*) es una cadena de caracteres utilizada para representar un recurso físico o abstracto, de tal forma que cada URI representará a un único recurso, pudiendo un recurso ser representado por distintas URI.



Un **URI** puede ser clasificado como un localizador, un nombre o ambos. En caso de referirse a una localización, se empleará un **URL** (*Uniform Resource Locator*), que no es más que un subconjunto de *URI* que permite la localización de un recurso utilizando una identificación abstracta de su situación. Ejemplos de URL puede ser `ftp://ftp.rediris.es/pub/index.html`, que identifica un documento HTML utilizando para su localización el protocolo FTP en una máquina remota. De una forma simple, un URL es lo que comúnmente se ha dado en llamar una dirección Internet.



Un **URN** (*Uniform Resource Name*) utiliza para la localización un servicio de nombres, por tanto es otra forma de identificación que intenta recoger la norma URI.

Sin embargo, HTML 5, al no estar basado en SGML, no necesita la referencia a una DTD. Sin embargo, requiere la declaración `<!DOCTYPE HTML>` para que los navegadores se comporten como deben.

ACTIVIDADES 2.6



- Utilice el documento HTML del ejemplo anterior para la DTD estricta de HTML 4 y observe cómo queda en su navegador.
- Modifique el documento HTML de la actividad anterior añadiendo la declaración **<!DOCTYPE>** para HTML 5 y observe cómo queda en su navegador.

2.2.7 CABECERA DEL DOCUMENTO

La cabecera del documento, delimitada por el elemento **HEAD**, contiene información sobre el mismo: título, información a utilizar por los programas buscadores y, en general, datos que no se consideran parte del contenido del documento pero aportan información sobre el mismo. El más importante, y obligatorio, de los posibles componentes de la cabecera es el título del documento (elemento **TITLE**). Los navegadores no presentan como contenido lo incluido en HEAD aunque pueden hacer disponible esta información mediante otros recursos.

El elemento HEAD, además de **TITLE**, puede contener, en cualquier orden, los siguientes elementos: *META*, *LINK*, *BASE*, *ISINDEX*, *STYLE* y *SCRIPT*.

2.2.7.1 Título del documento

Es el más importante de los componentes de la cabecera y se introduce mediante el elemento **TITLE**. Todo documento HTML ha de incluirlo. El navegador no lo presentará junto al contenido del documento pero lo hará siempre disponible al usuario. Los navegadores gráficos usuales lo presentan en la barra de títulos de la ventana y lo utilizan como referencia a incluir en su lista de recursos favoritos del usuario (lo que constituye una razón más para que el título proporcione una descripción adecuada del documento).

El elemento **TITLE** no puede contener otros elementos ni tener ninguna indicación de formato. Sí pueden utilizarse entidades de caracteres para acentos, símbolos especiales, etc.

No debe confundirse el **elemento TITLE**, que afecta a todo el documento, con el **atributo title** que afecta a múltiples elementos.

PRÁCTICA 2.3



Fíjese en el documento HTML siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"  
      "http://www.w3.org/TR/REC-html40/loose.dtd">  
<html>
```

```
<head>
<title>Películas de culto: Nosferatu</title>
</head>

<body bgcolor="#000000">

<h1 align="center"><font color="#800000">NOSFERATU de Murnau (1921)</font></h1>

<p align="center"></p>
</body>
</html>
```

Al pasarlo al navegador, verá una imagen parecida a la siguiente:



Figura 2.3

El título del documento "Películas de culto: Nosferatu" es el contenido del elemento **TITLE** y es visualizado por el navegador en la barra de títulos de la ventana del programa; el atributo **title** afecta al elemento **IMG** (que inserta una imagen "nosferatu.jpg" en el documento) designando su contenido como "Fotograma de Nosferatu". Cuando el ratón se detenga sobre la imagen el navegador debería mostrar un mensaje con el contenido del atributo **title**.

ACTIVIDADES 2.7



- Muestre en su navegador el documento HTML del ejemplo anterior y observe cómo queda.
- Una vez tenga la página web en su navegador, vea su código fuente.



Para ver el código fuente en Internet Explorer 8, vaya a **Ver código fuente** del menú **Página**.



Para ver el código fuente en Mozilla Firefox, vaya a **Código fuente de la página** del menú **Ver**.

2.2.8 CUERPO DEL DOCUMENTO

El cuerpo de un documento HTML constituye el contenido del mismo que será visualizado en la pantalla de su programa navegador (si se trata de un navegador no visual, será presentado como sonido, en braille, etc.).

El cuerpo del documento puede contener una gran variedad de elementos que soportan una aún mayor variedad de atributos.

En los documentos con marcos el elemento **BODY** se reemplaza por el elemento **FRAMESET**.

2.2.8.1 Elementos de bloque y de línea

La mayoría de los elementos que pueden aparecer en el cuerpo del documento se pueden clasificar en elementos a nivel de bloque y elementos a nivel de texto o *inline*. La distinción, de acuerdo a la especificación del W3C, se basa en varios criterios:

- ✓ Generalmente, los elementos a nivel de bloque pueden contener otros elementos de bloque y también elementos de línea. En cambio, generalmente, los elementos a nivel de línea solo pueden contener texto y otros elementos de línea. Los elementos de bloque organizan el texto en estructuras más grandes que los elementos de línea.
- ✓ Por defecto, los elementos a nivel de bloque se formatean de manera diferente que los elementos a nivel de línea. Los primeros, generalmente, comienzan en una nueva línea y los segundos no.
- ✓ Por razones técnicas relacionadas con el algoritmo de texto bidireccional, unos elementos y otros difieren en cómo heredan la información acerca de la dirección del texto

Son elementos de bloque: ADDRESS, BLOCKQUOTE, CENTER, DIR, DIV, DL, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, ISINDEX, MENU, NOFRAMES, NOSCRIPT, OL, P, PRE, TABLE, UL.

Son elementos a nivel de texto: A, ACRONYM, APPLET, B, BASEFONT, BDO, GIG, BR, CITE, CODE, DFN, EM, FONT, I, IFRAME, IMG, KBG, MAP, OBJECT, Q, S, SAMP, SCRIPT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TT, U, VAR.

2.2.8.2 Identificación de elementos: atributos **class** e **id**

Por su importancia en HTML cabe destacar dos atributos **id** y **class** que pueden ir asociados a casi todos los elementos.

El atributo **id** asigna un nombre a un elemento en el documento. Ese nombre no puede repetirse en un mismo documento por lo que permite identificar cualquier instancia, cualquier aparición en el documento, de un mismo elemento. Tiene varios usos en HTML:

- ✓ Como selector en una hoja de estilo.
- ✓ Como ancla de destino en enlaces de hipertexto.
- ✓ Como recurso para hacer referencia a un elemento particular desde un *script*.
- ✓ Como nombre declarado de un elemento **OBJECT**.
- ✓ Para propósito general en el funcionamiento de los navegadores (por ejemplo, para identificar campos cuando se extraen datos desde una página HTML a una base de datos).



EJEMPLO 2.5

Vea el siguiente trozo de documento HTML:

```
<p id="Fuente">De la Presentación de Luis Monreal al catálogo&nbsp; de la exposición  
de William Blake organizada por Fundación "laCaixa" en 1996 </p>
```

Mediante la asociación con el atributo **id** este párrafo concreto recibe un identificador exclusivo al que se puede hacer referencia en otras partes del documento.

El atributo **class** asigna uno o más nombres de clase a un elemento, puede decirse entonces que el elemento pertenece a esa clase o clases. Un mismo nombre de clase puede ser compartido por varias instancias (varias apariciones en un documento) de un mismo elemento y distintos elementos pueden pertenecer a la misma clase. Tiene varios usos en HTML:

- ✓ Como selector de estilo cuando el autor desea asignar información de estilo a un grupo de elementos.
- ✓ Para propósito general.



EJEMPLO 2.6

En el siguiente ejemplo se establece que el elemento **SPAN** pertenece a la clase *cursiva* para aplicarle la regla de estilo correspondiente.

```
<p>En 1809, cuando William Blake expuso por primera vez, Robert Hunt, crítico del  
<span class="cursiva">Examiner</span>, escribió de él </p>
```

2.2.8.3 Agrupación de elementos: elementos div y span

Los elementos **DIV** y **SPAN** constituyen un potente recurso para estructurar documentos teniendo gran relevancia en HTML. Sirven para definir su contenido como de nivel de bloque, elemento **DIV**, o de nivel de texto, elemento **SPAN**, no implican nada más pero ello será de gran importancia al combinarse con información de estilo. Más adelante ofreceremos múltiples ejemplos.

2.2.9 TRABAJANDO EN EL CUERPO DEL DOCUMENTO

2.2.9.1 Párrafos, líneas y espacios

Al escribir, se organiza el discurso en párrafos. Esta es la unidad básica con la que se ha de estructurar el cuerpo de los documentos HTML (el elemento **P** no admite en su modelo de contenido otros elementos de bloque, ni siquiera el propio **P**). Muy probablemente, estará acostumbrado al uso de procesadores de texto en los que un retorno de carro provoca el comienzo de un nuevo párrafo. En HTML esta práctica no funciona, para identificar un fragmento de discurso como párrafo habrá de delimitarlo con las etiquetas **<P>** y **</P>**. La última puede ser omitida, pero en los primeros documentos será más fácil controlar los posibles errores cometidos si no omite la etiqueta de fin.

Como los párrafos son presentados visualmente, dependen de los programas navegadores, normalmente los visualizará con una línea en blanco antes y después de cada párrafo.

PRÁCTICA 2.4

Fíjese en el documento HTML siguiente:

```
<P>[...] Ni que decir tiene que la lógica, en el transcurso de su larga y sinuosa historia, ha tenido a menudo conflictos fronterizos con algunas otras disciplinas, o incluso ha sido, pura y simplemente confundida con ellas: con la psicología del razonamiento, con la teoría de la ciencia, con la teoría del conocimiento o con la ontología.</P>
<P> Por supuesto también que la lógica, en cuanto ciencia del análisis formal del razonamiento, no pretende agotar todos los aspectos de éste.</P>
```

Al pasarlo al navegador, verá una imagen parecida a la siguiente:

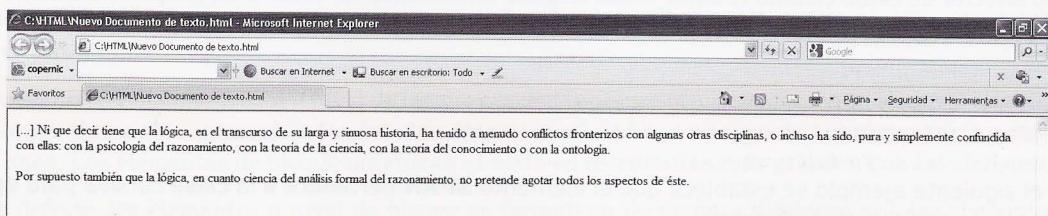


Figura 2.4

Como se dijo anteriormente, los navegadores acomodan el texto de los párrafos a lo largo del tamaño de la ventana, por lo tanto, si disminuye el tamaño de la ventana, se modificará la anchura de los párrafos.

En muchos casos se deseará forzar un salto de línea dentro de un párrafo. Para ello, se ha de utilizar el elemento vacío **
**. Este elemento provocará un salto a la línea siguiente sin que el navegador muestre una línea en blanco.

ACTIVIDADES 2.8



- Se puede reescribir la práctica anterior en un solo párrafo. Para ello, elimine las etiquetas que marcaban el segundo párrafo e introduzca un salto de línea donde antes se indicaba el inicio de un nuevo párrafo:

<P>

[...] Ni que decir tiene que la lógica, en el trascurso de su larga y sinuosa historia, ha tenido a menudo conflictos fronterizos con algunas otras disciplinas, o incluso ha sido, pura y simplemente confundida con ellas: con la psicología del razonamiento, con la teoría de la ciencia, con la teoría del conocimiento o con la ontología.
 Por supuesto también que la lógica, en cuanto a ciencia del análisis formal del razonamiento, no pretende agotar todos los aspectos de éste.

</P>

Compruebe en el navegador que el resultado sea el correcto.

Una serie de saltos de línea tendrán el efecto correspondiente en la presentación del documento produciendo, ahora sí, líneas en blanco.

En HTML una sucesión de espacios en blanco, introducidos mediante la tecla espaciadora o la tecla del tabulador en el documento fuente, será visualizada por el navegador como un solo espacio en blanco. Se debe evitar la tendencia a utilizarlos para formatear. Si se desea introducir de manera efectiva un espacio en blanco o una sucesión de los mismos se debe utilizar la entidad **&nbsp**. Su uso no está orientado a formatear el texto sino a impedir que el navegador produzca una ruptura de línea entre dos palabras determinadas.

De la misma manera, se debe evitar el uso de elementos **P** vacíos o elementos **P** conteniendo solo la entidad **&nbsp**; para formatear el texto.



EJEMPLO 2.7

En el siguiente ejemplo se va a utilizar la entidad **&nbsp** para evitar, sea cual sea el tamaño de la ventana del navegador, una ruptura de línea entre los términos *lógicas* y *no-clásicas* en la segunda ocurrencia de la expresión en el cuerpo del texto:

```
<h1>Las lógicas llamadas «no-clásicas»</h1>
<p>Acaso el rasgo más llamativo -si no el más importante- del estado actual de la ciencia de la lógica formal sea la existencia, e incluso la proliferación, en derredor suyo, de las llamadas «lógicas&ampnbspno-clásicas»</p>
<p>DEAÑO Alfredo: Introducción a la lógica formal.</p>
```

Si compara la presentación de los párrafos en el navegador (líneas en blanco entre párrafos, alineación a la izquierda) con el control de su presentación en los textos impresos o en pantalla mediante procesadores de texto o programas de autoedición, sin duda parecerá muy pobre. Se echa en falta la justificación a ambos márgenes, la sangría de la primera línea, etc. Sin embargo, se puede mejorar la presentación de los documentos con los recursos orientados a la presentación en HTML. Si además, se utilizan las hojas de estilo, recomendadas insistenteamente desde la versión 4.0 de HTML, se podrá tener un control mucho mayor de la maquetación del documento.



EJEMPLO 2.8

Vea cómo esto es posible con hojas de estilo. No se va a explicar ahora su sintaxis sino que se va a limitar mostrar su uso para controlar la presentación de los párrafos. Más adelante, encontrará un epígrafe específico dedicado a las hojas de estilo.

```
<html>
<head>
<title>Párrafos con estilo</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
P { font-family: Arial, Verdana; font-size: 10pt; color: rgb(0,0,128); text-indent: 15px; text-align: justify; margin-left: 10px }
</style>
</head>
<body>
<p>[...]</p>
<p>Ni que decir tiene que la lógica, en el transcurso de su larga y sinuosa historia, ha tenido a menudo conflictos fronterizos con algunas otras disciplinas, o incluso ha sido, pura y simplemente confundida con ellas: con la psicología del razonamiento, con la teoría de la ciencia, con la teoría del conocimiento o con la ontología.</p>
<p>Por supuesto también que la lógica, en cuanto ciencia del análisis formal del razonamiento, no pretende agotar todos los aspectos de éste. Hay en el razonamiento-dicho sea cometiendo la vulgaridad de parafrasear una vez más&nbsp; una frase de Hamlet- muchas más cosas que su pura forma, otras muchas cosas que la lógica no busca. Ocurre simplemente que la actividad científica -y precisamente por eso la actividad científica necesita de la filosofía- opera sobre la base de la división -técnica, y no social- del trabajo. De ahí que no hayamos dicho que la lógica sea la ciencia del razonamiento a secas, sino la ciencia que se ocupa de los aspectos formales del razonamiento.</p>
<p>[...] </p>
<p><span style="font-weight: bold">DEAÑO</span>, Alfredo:<em> Introducción a la lógica formal</em>, Alianza Universidad Textos, Madrid, 1978.</p>
</body>
</html>
```

En la cabecera del documento, entre las etiquetas **<STYLE>** y **</STYLE>** se ha establecido mediante reglas de estilo, que:

- Los párrafos (**<style> P ... </style>**) han de mostrarse en la fuente arial o, en el caso de que esta fuente no esté disponible, en la fuente verdana (**font-family: Arial, Verdana;**).
- El tamaño de la fuente (**10pt;**).
- El texto se visualizará en color azul marino (**color: rgb(0,0,128);**).
- La primera línea de cada párrafo aparecerá indentada 15 píxeles (**text-indent: 15px;**).
- El párrafo estará justificado a derecha e izquierda (**text-align: justify;**).
- El párrafo estará indentado 10 píxeles respecto al margen izquierdo (**margin-left: 10px;**).

De todo ello resulta la siguiente página:

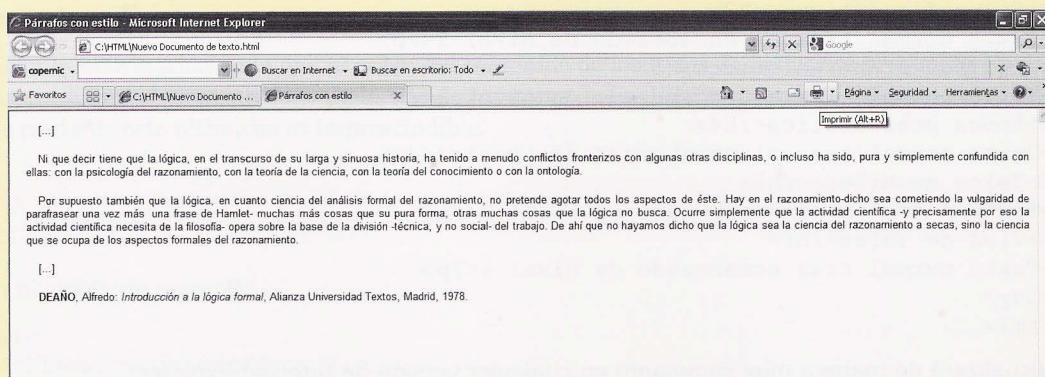


Figura 2.5

Relacionado también con la presentación del texto está el uso de guiones. Además de su uso como cualquier otro carácter, el guión puede utilizarse para indicar dónde puede producirse una ruptura de línea (igual que cuando utilizamos una máquina de escribir clásica).

El guión normal, carácter “-”, se consigue directamente por el teclado o mediante las referencias a caracteres - y -.

El guión que indica la posible rotura de línea se consigue mediante la entidad de carácter ­ (y también mediante las referencias numéricas ­ y ­)

No todos los navegadores son capaces de interpretar el segundo tipo de guión. Si lo son y una línea está rota con un guión de este tipo, deben colocar un guión al final de la primera línea. Si la línea no se rompe el guión no debe ser mostrado.

2.2.9.2 Encabezados

Las cabeceras o encabezados proporcionan un medio simple, pero útil, para estructurar el cuerpo del documento permitiéndonos dividirlo en secciones.

Existen seis niveles de cabeceras **H1 ... H6**. Los navegadores presentan cada una de ellas con tipografías diferentes y normalmente se comportarán como el elemento P provocando una línea en blanco.



EJEMPLO 2.9

El siguiente ejemplo:

```
<html>
<head>
<title>Encabezados</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
</head>
<body>
<h1>Mínima historia de la filosofía</h1>
<p>Texto normal tras el en cabezado de nivel 1</p>
<h2>Filosofía antigua</h2>
<p>Texto normal tras el encabezado de nivel 2</p>
<h3>Periodo prehelenístico</h3>
<p>Texto normal tras el encabezado de nivel 3</p>
<h4>Época presocrática</h4>
<p>Texto normal tras el encabezado de nivel 4</p>
<h5>Tales de Mileto</h5>
<p>Texto normal tras encabezado de nivel 5</p>
<h6>Vida de Tales</h6>
<p>Texto normal tras encabezado de nivel 6</p>
</body>
</html>
```

se visualizará de manera muy semejante en cualquier versión de Internet Explorer:

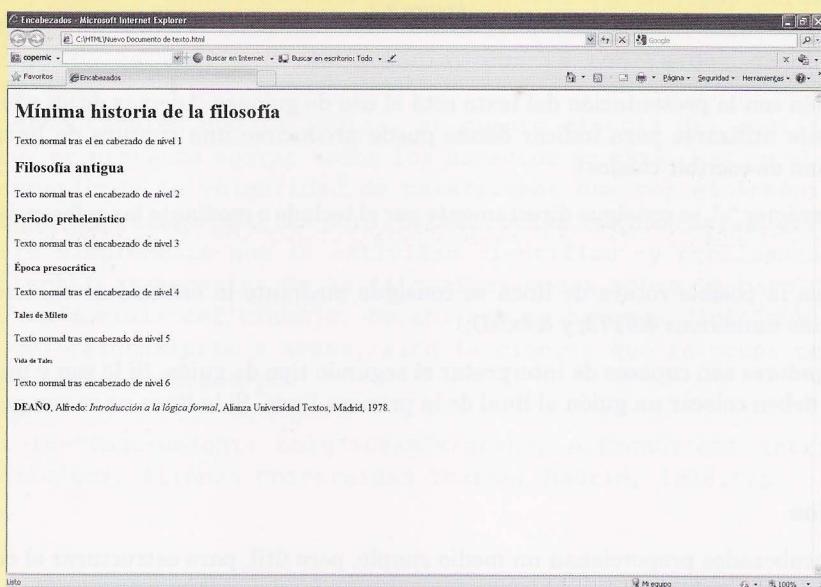


Figura 2.6

ACTIVIDADES 2.9



- Si dispone de un navegador distinto a Internet Explorer, compruebe cómo se visualizaría el documento HTML del ejemplo anterior.

2.2.9.3 Listas

Las listas constituyen un recurso muy versátil que permite estructurar muy diversos tipos de contenido. En función de la información que contienen se clasifican en listas desordenadas, listas ordenadas y listas de definición.

Listas desordenadas

Organizan la información en la que el orden del contenido de los ítems de la lista se presupone no determinante. Se declaran mediante el elemento UL. Cada uno de los ítems de las lista se introduce mediante un elemento LI. La etiqueta de cierre de este último no es imprescindible.



EJEMPLO 2.10

En el siguiente ejemplo:

```
<ul>
    <li>Marco geográfico</li>
    <li>Política</li>
    <li>Economía</li>
    <li>Sociedad</li>
    <li>Cultura</li>
    <li>Vida cotidiana</li>
</ul>
```

Los navegadores mostrarán las listas indentadas y añadiendo una viñeta:

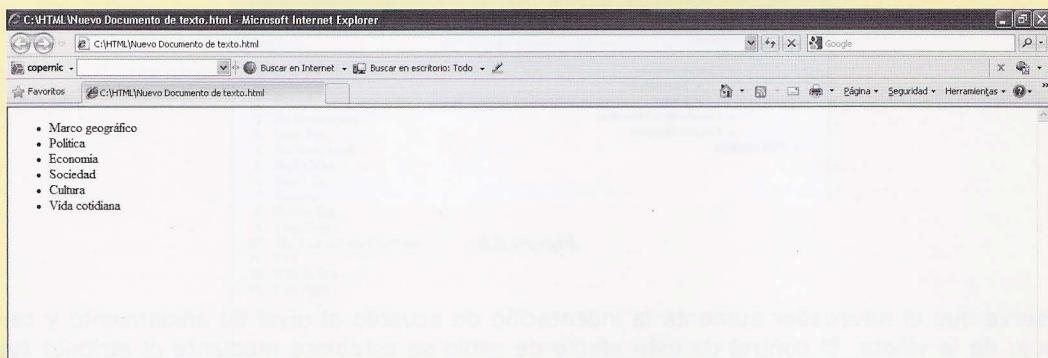


Figura 2.7



EJEMPLO 2.11

Las listas pueden anidarse. Por ejemplo:

```
<ul>
  <li>Marco geográfico</li>
  <li>Política</li>
  <li>Economía</li>
  <li>Sociedad</li>
  <li>Cultura<ul>
    <li>Arte</li>
    <li>Pensamiento</li>
    <li>Ciencia<ul>
      <li>Ciencias formales<ul>
        <li>Matemáticas</li>
        <li>Lógica</li>
      </ul>
    </li>
    <li>Ciencias de la Naturaleza</li>
    <li>Ciencias Humanas</li>
  </ul>
  </li>
</ul>
</li>
<li>Vida cotidiana</li>
</ul>
```

El navegador presentará así la lista:

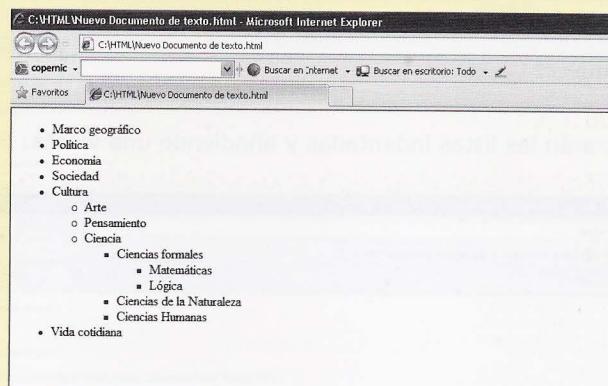


Figura 2.8

Observe que el navegador aumenta la indentación de acuerdo al nivel de anidamiento y cambia el dibujo de la viñeta. El control de este efecto de estilo se establece mediante el atributo **type** que admite los valores *disc*, *square* y *circle*. El navegador utiliza, como en el ejemplo anterior, un valor por defecto que dependerá del nivel de anidamiento de los ítems de la lista.

ACTIVIDADES 2.10



- Prepare un documento HTML para una lista desordenada anidada.

Listas ordenadas

Estas listas se consiguen con el elemento OL. Los ítems de lista se definen también mediante elementos LI para los que se puede omitir la etiqueta de cierre. El navegador numera automáticamente los ítems de la lista.



EJEMPLO 2.12

En el siguiente ejemplo:

```
<ol>
  <li>Terrapin</li>
  <li>No Good Trying</li>
  <li>Love You</li>
  <li>No Man's Land</li>
  <li>Dark Globe</li>
  <li>Here I Go</li>
  <li>Octopus</li>
  <li>Golden Hair</li>
  <li>Long Gone</li>
  <li>She Took A Long Cold Look</li>
  <li>Feel</li>
  <li>If It's In You</li>
  <li>Late Night</li>
</ol>
```

observe que el navegador presenta la lista numerada:

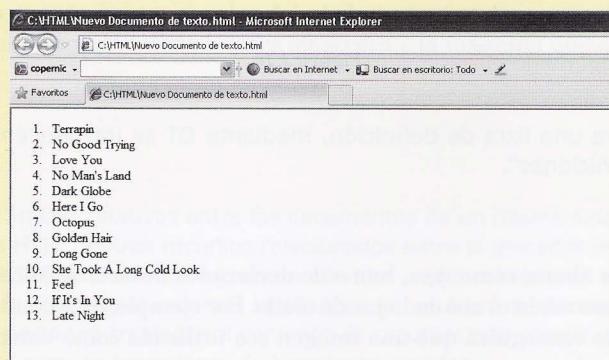


Figura 2.9

ACTIVIDADES 2.11



- Prepare un documento HTML para una lista ordenada.

Listas de definición

Este tipo de listas se suelen usar para elaborar glosarios, aunque como se ha mostrado en el ejemplo anterior, pueden adecuarse a otros tipos de contenidos.



EJEMPLO 2.13

En el siguiente ejemplo:

```
<dl>
  <dt>SYD BARRET</dt>
  <dd>Miembro fundador de Pink Floyd "eclipsado" después de sus primeras
  grabaciones</dd>
  <dd>Crazi Diamond (Grabaciones completas-BOX-1)</dd>
</dl>
```

observe que el navegador presenta la lista de definición:

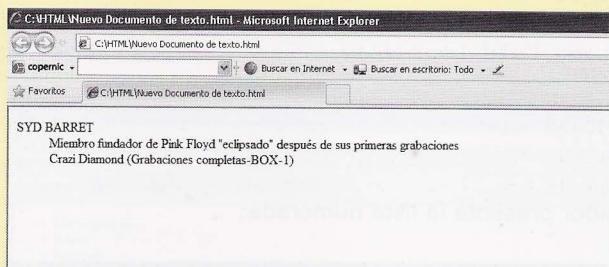


Figura 2.10

Mediante DL se declara una lista de definición, mediante DT se introducen los "términos a definir" y mediante DD las "definiciones".

Atributos comentados hasta ahora, como *type*, han sido declarados desaconsejados en la recomendación del W3C para HTML. En su lugar, se recomienda el uso de hojas de estilo. Por ejemplo, añadiendo las siguientes reglas de estilo en la cabecera del documento se conseguirá que una imagen sea utilizada como viñeta en las listas desordenadas y que la numeración de las ordenadas se establezca en números romanos:

```
UL { list-style-image: url('images/right.gif') }
OL { list-style-type: lower-roman }
```

ACTIVIDADES 2.12

- Prepare un documento HTML para una lista de definición.

2.2.10 HIPERTEXTO

HTML no es el único lenguaje con capacidades de hipertexto pero, sin duda, la sencillez con que, apoyándose en la convención URI, permite crear vínculos entre recursos del Internet es uno de los factores que explican su éxito como lenguaje de publicación en Internet y su expansión a otros ámbitos.

Las capacidades **hipertexto** de HTML descansan, fundamentalmente aunque no exclusivamente, en dos elementos **LINK** y **A**. *LINK*, se utiliza en la cabecera para definir relaciones entre los documentos (se tratará en el capítulo *Trabajando en la cabecera del documento*) mientras que *A* se utiliza en el cuerpo del documento para crear enlaces de hipertexto. Será el que ahora se abordará.

Un enlace de hipertexto consta de dos partes: un punto de origen y un punto de destino. El comportamiento típico del enlace consiste en que cuando el usuario actúa sobre el punto de origen, localizado en un determinado documento, el navegador recuperará el documento, imagen o cualquier otro tipo de recurso, que haya sido definido como punto de destino de ese enlace.

El elemento **A** tiene dos usos según vaya asociado al atributo **href** o a los atributos **name** o **id**. Cuando se utiliza con **href** establecerá el punto de origen de un enlace de hipertexto: el contenido del elemento se transformará en una *zona activa* (que los navegadores, generalmente, muestran subrayada y destacada mediante un color determinado) y el valor del atributo **href**, un URI, constituye el punto de destino del enlace. El usuario desencadenará la recuperación del recurso enlazado cuando activa el enlace, normalmente, mediante una pulsación de ratón sobre la zona activa.

Cuando se utiliza con los atributos **name** o **id** el elemento **A** especificará el punto de destino de un enlace de hipertexto, de manera que al acceder al documento, el navegador no se situará en el comienzo de la página sino el punto marcado mediante el elemento **A** asociado a los atributos **name** o **id**. En esto consiste el uso del elemento **A** como **marcador** de páginas.

Un tercer factor a tener en cuenta es que el valor del atributo **href** es siempre un URI que puede expresarse de manera absoluta (incluyendo el protocolo utilizado para su acceso y la máquina que alberga el recurso) o relativa.



Normalmente, se utilizarán URI relativos entre los documentos de un mismo sitio web, es decir, un conjunto de documentos HTML y otros recursos relacionados entre sí ubicados en un servidor de Internet, en el servidor HTTP de una intranet, en un servidor web local ubicado en nuestro propio ordenador e, incluso, para referenciar recursos localizados en un árbol de directorios del disco duro de nuestro ordenador. Esto permitirá trasladar el conjunto de recursos que constituyen el sitio web de un servidor a otro sin que se rompan los enlaces de hipertexto creados.



Cuando se creen enlaces a otros sitios web o, dentro de un mismo sitio web, se creen enlaces a recursos ubicados en un servidor distinto, se utilizarán referencias absolutas.

Se va a aclarar los dicho anteriormente con varios ejemplos. Suponga en primer lugar que se ha creado un sitio web con la siguiente estructura de directorios:

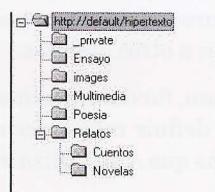


Figura 2.11

El directorio raíz del sitio web es *hipertexto*, allí está ubicada la página principal que da acceso a todas las demás llamada *default.htm*. En el directorio *images* se ha almacenado todas las imágenes utilizadas en el sitio web.

Imagine que en la página principal, archivo *default.htm*, se enlaza con diversos recursos ubicados en los directorios de la web:

```

<ul class="navegar">
    <li><a href="indice.htm">Índice de autores</a></li>
    <li>Índice de textos </li>
    <li><a href="Poesia/preformatoado.htm">Poesía</a></li>
    <li><a href="Ensayo/inteligencia_artificial.htm">Ensayo</a></li>
    <li><a href="Relatos/Novelas/Musil.htm">Novela</a></li>
    <li><a href="Relatos/Cuentos/texto.htm">Relatos</a></li>
</ul>
  
```

Observe como las etiquetas de inicio y cierre del elemento **A** delimitan el texto que funcionará como zona activa. Cada uno de los recursos enlazados se especifica como valor del atributo **href** expresando su URI de manera relativa, es decir, indicando exclusivamente la ruta de acceso al recurso y el nombre del mismo pero no el nombre del protocolo y de la máquina que alberga el recurso.

Recuerde que para resolver un URI relativo se utilizan los siguientes criterios en orden de precedencia:

1. A partir de un URI establecido en la cabecera del documento, mediante el elemento **BASE**, como referencia absoluta a partir de la cual resolver las referencias relativas.
2. A partir de un URI base dado a través de datos META encontrados en la actividad del protocolo de comunicación como, por ejemplo, una cabecera http.
3. Por defecto se toma como URI base el del documento actual.

En los ejemplos se toma siempre el valor por defecto, es decir, el del documento actual que para nuestro documento *default.htm* no es otro que *http://default/hipertexto/default.htm*².

Vea ahora el ejemplo de enlaces a recursos externos ubicados en servidores remotos. En este caso se emplearán referencias absolutas:

```
<p>Puede encontrar otros recursos interesantes en las páginas del <a href="http://www.promo.net/pg/">Proyecto Gutenberg</a>, en la <a href="http://www.hum.aau.dk./Institut/rom/borges/vb3/helft.htm">Biblioteca Total</a> y en el <a href="http://www.perseus.tufts.edu./Texts.html">Proyecto Perseus </a>.</p>
```

El primero de los enlaces lleva a la página de inicio del sitio oficial del proyecto Gutenberg. Se indica protocolo, nombre del servidor y ruta de acceso al recurso. No se indica el nombre de este *index.html* pues los sitios web tienen una página de inicio por defecto, normalmente, llamada *index.html* o *default.html* (recuerde que los URI son sensibles al uso de mayúsculas o minúsculas pues la mayoría de los servidores web están instalados en máquinas Unix).

Ahora, se mostrará un ejemplo de enlace de correo:

```
<address>
    <a href="mailto:nombre@servidor.es">Aquí</a> podrá contactar con nosotros
</address>
```

Suponga ahora que en cada una de las páginas ubicadas en los diversos directorios representados en la Figura 2.11 se incluye un enlace a la página de inicio de la web de ejemplo utilizando direcciones relativas.

Desde el directorio *Poesia* el enlace tendrá la forma:

```
<p><a href="../Default.htm">Página de inicio</a></p>
```

Donde “..” representa el directorio que contiene al directorio actual. Recuerde que el URI base que se toma por defecto para resolver un URI relativo es el del documento actual.

Sin embargo, desde el directorio *Novelas* sería:

```
<p><a href=".../Default.htm">Página de inicio</a></p>
```

Donde “...” indica que hay que subir dos niveles en el árbol de directorios pues el directorio que contiene al directorio actual (*Novelas* cuelga de *Relatos*) a su vez está contenido en el directorio raíz de la web.

Para hacer referencia desde un documento ubicado en el directorio *Poesia* a uno ubicado en el directorio *Novela* sería:

```
<p><a href=".../Relatos/Novelas/Musil.htm">Novela </a> </p>
```

Hay que subir un nivel, pues el menú *Poesía*, del que cuelga el documento actual, cuelga del directorio raíz.

Para hacer referencia desde un documento ubicado en *Novela* a otro ubicado en *Cuentos* sería:

```
<p><a href=".../Cuentos/texto.htm">Cuentos</a></p>
```

Pues *Novela* y *Cuentos* cuelgan del mismo directorio (*Relatos*).

2 En el ejemplo se está trabajando con un servidor web personal, si está reproduciendo los ejemplos y almacenándolos en su disco duro, su URI base para este documento podría ser, por ejemplo, *C:\WEBSHARE\WWWROOT\hipertexto\Default.htm*. Dependerá del directorio en que esté ubicando los ejemplos. En todo caso, al tomar como *URI* base el del documento actual los enlaces siempre funcionarán correctamente.

Para hacer referencia desde un documento ubicado en *Ensayo* a otro ubicado en el mismo directorio sería:

```
<p><a href="turing.htm">Más ensayos</a></p>
```

Es decir, bastará con introducir el nombre del recurso.

ACTIVIDADES 2.13



- Prepare un documento HTML que tenga un enlace de hipertexto con un URI con referencia absoluta.
- Prepare un documento HTML que tenga un enlace de hipertexto con un URI con referencia relativa.

2.2.11 MARCADORES

Cuando el elemento **A** va asociado al atributo **name** o **id** se convierte en punto de destino de un enlace. Un valor del atributo *name* no puede repetirse dentro de un mismo documento (aunque al ser sensible al uso de mayúsculas o minúsculas, *Arriba* es un valor distinto que *ARRIBA*).



EJEMPLO 2.14

Observe el siguiente fragmento de código:

```
<p>Debo a la conjunción de un espejo y de una enciclopedia el descubrimiento de Uqbar. El espejo inquietaba el fondo de un corredor en una <a name= "quinta">quinta</a> de la calle Gaona, en Ramos Mejía; la enciclopedia falazmente se llama <em>The Anglo-American Cyclopaedia</em> (Nueva York, 1917) y es una reimpresión literal, pero también morosa, de la <em>Encyclopaedia Britannica</em> de 1902. El hecho se produjo hará unos cinco años. <a name="Bioy Casares" href="bioy.htm">Bioy Casares</a> había cenado conmigo esa noche y nos demoró una vasta polémica sobre la ejecución de una novela en primera persona, cuyo narrador omitiera o desfigurara los hechos e incurriera en diversas contradicciones, que permitieran a unos pocos lectores -a muy pocos lectores- la adivinación de una realidad atroz o banal. Desde el fondo remoto del corredor, el espejo nos acechaba. Descubrimos (en la alta noche ese descubrimiento es inevitable) que los espejos tienen algo de monstruoso. Entonces Bioy Casares recordó que uno de los <a name= "heresiarcas" href="heresiarch.htm">heresiarcas</a> de Uqbar había declarado que los espejos y la <a name="cópula">cópula </a> son abominables, porque multiplican el número de los hombres.
```

Se trata del inicio de un cuento de Jorge Luis Borges: *Tlön, Uqbar, Orbis Tertius*.

En la primera aparición del elemento A se ha utilizado como un marcador:

```
<a name="quinta">quinta</a>
```

Este marcador puede convertirse en el punto de destino de un enlace de hipertexto. Por ejemplo:

También encontrará enlaces a relatos en los que utilizan [Relatos/Cuentos/texto.htm#quinta](#) términos en acepciones no habituales.

Observe que el valor de *href* es un *URI* relativo y el nombre del recurso va seguido del signo “#” y del nombre del marcador que se había definido anteriormente. Al activar el enlace, el navegador se situará en la línea que contiene el marcador en lugar de en el comienzo del documento:

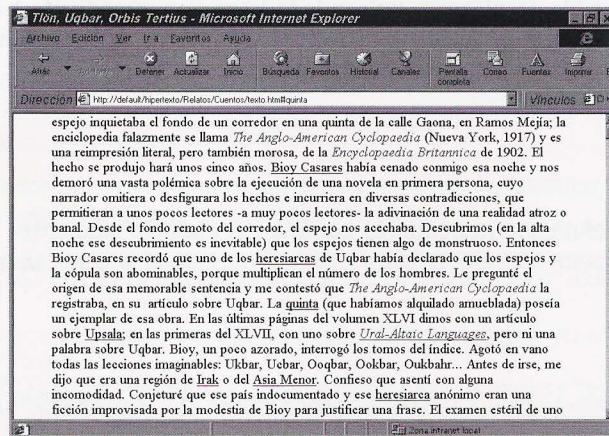


Figura 2.12

También se podría haber utilizado un URI absoluto para referirse a un documento ubicado en un servidor remoto:

También encontrará enlaces a relatos en los que utilizan `términos` en acepciones no habituales.

Observe ahora las siguientes instancias de A con *name* que aparecen en el ejemplo propuesto. La que sigue a la ya comentada es:

hará unos cinco años. `Biyo Casares` había

Se han utilizado simultáneamente los atributos *name* y *href*. La expresión *Biyo Casares* se convierte con ello en punto de origen de un enlace de hipertexto, gracias al atributo **href**, y al mismo tiempo punto de destino de otros enlaces gracias al uso del atributo **name**.

Los marcadores son especialmente útiles cuando se utilizan dentro de un mismo documento. Se recurre a ellos, generalmente, cuando la página es de cierta extensión o cuando es adecuado a su contenido, por ejemplo, en un glosario de términos. Observe la siguiente página:

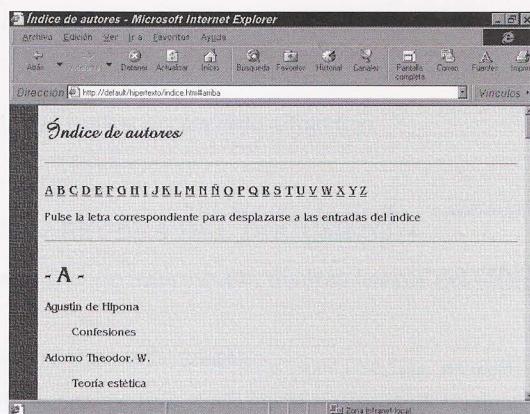


Figura 2.13

Utilizando marcadores es posible desplazarse pulsando la letra correspondiente a la sección del índice donde se encontrarán los autores clasificados alfabéticamente. Un enlace al final de cada sección alfabética devolverá al lector al comienzo de la página permitiéndole realizar otra consulta.



EJEMPLO 2.15

El efecto se consigue definiendo marcadores como en el ejemplo siguiente:

```
<h1><font face="Amaze"><a name="arriba">Índice de autores</a></font></h1>

<hr>

<h3><a href="#sectA">A</a> <a href="#sectB">B</a> <a href="#sectC">C</a>
<a href="#sectD">D</a>
<a href="#sectE">E</a> <a href="#sectF">F</a> <a href="#sectG">G</a>
<a href="#sectH">H</a>
<a href="#sectI">I</a> <a href="#sectJ">J</a> <a href="#sectK">K</a>
<a href="#sectL">L</a>
<a href="#sectM">M</a> <a href="#sectN">N</a> <a href="#sectNT">Ñ</a>
<a href="#sectO">O</a>
<a href="#sectP">P</a> <a href="#sectQ">Q</a> <a href="#sectR">R</a>
<a href="#sectS">S</a>
<a href="#sectT">T</a> <a href="#sectU">U</a> <a href="#sectV">V</a>
<a href="#sectW">W</a>
<a href="#sectX">X</a> <a href="#sectY">Y</a> <a href="#sectZ">Z</a> </h3>

<p>Pulse la letra correspondiente para desplazarse a las entradas del índice</p>

<hr>

<h1><a name="sectA">- A -</a></h1>

<dl>
  <dt><strong>Agustín de Hipona</strong></dt>
  <dd><br>
    Confesiones</dd>
</dl>

<dl>
  <dt><strong>Adorno Theodor. W.</strong></dt>
  <dd><br>
    Teoría estética</dd>
</dl>

<h5><a href="#arriba">Nueva consulta</a></h5>
```

En dicho ejemplo:

1. Se define una marcador (`name="Arriba"`) para la expresión *Índice de autores*. Este será el destino de los enlaces *Nueva consulta* con los que acaba cada sección alfabética (`<h5>Nueva consulta</h5>`).
2. Se define un marcador para cada una de las letras, en orden alfabético, que abren cada una de las secciones (`<h1>- A -</h1>`).
3. Un enlace desde cada letra a la sección correspondiente permite desplazarse a las entradas del índice:

```
<h3><a href="#sectA">A</a> <a href="#sectB">B</a> <a href="#sectC">C</a>
<a href="#sectD">D</a>
<a href="#sectE">E</a> <a href="#sectF">F</a> <a href="#sectG">G</a>
<a href="#sectH">H</a>
<a href="#sectI">I</a> <a href="#sectJ">J</a> <a href="#sectK">K</a>
<a href="#sectL">L</a>
<a href="#sectM">M</a> <a href="#sectN">N</a> <a href="#sectNT">Ñ</a>
<a href="#sectO">O</a>
<a href="#sectP">P</a> <a href="#sectQ">Q</a> <a href="#sectR">R</a>
<a href="#sectS">S</a>
<a href="#sectT">T</a> <a href="#sectU">U</a> <a href="#sectV">V</a>
<a href="#sectW">W</a>
<a href="#sectX">X</a> <a href="#sectY">Y</a> <a href="#sectZ">Z</a> </h3>
```

Observe como para hacer referencia a marcadores dentro de un mismo documento se utiliza el nombre del marcador precedido del signo "#" sin que sea necesario indicar el nombre del archivo al especificar valores para `href`. (`href="#arriba"`).

Si desde otro documento se quisiera enlazar con un punto concreto de esta página sí habría de indicar el nombre de la misma. Por ejemplo:

```
<p>Esta noche visitamos la Biblioteca Nacional. En vano fatigamos atlas, <a href=".../indice.htm#sectU"> catálogos </a>, anuarios de sociedades geográficas, memorias de viajeros e historiadores: nadie había estado nunca en Uqbar.</p>
```

ACTIVIDADES 2.14



➤ Prepare un documento HTML que realice un glosario alfabético.

2.2.12 TRABAJANDO EN LA CABECERA DEL DOCUMENTO

La cabecera del documento puede incluir información diversa acerca del mismo. Para ello, generalmente, habrá que:

1. Declarar una propiedad y un valor para la misma.
2. Indicar un archivo de perfil donde se han definido la propiedad y sus valores válidos. Para especificar un archivo de este tipo debe usarse el atributo **profile** del elemento **HEAD**.

El primer método puede ejecutarse de dos formas:

- ✓ Desde la cabecera del documento: utilizando el elemento **META**.
- ✓ Desde fuera del documento: enlazando la metainformación mediante el elemento **LINK**.

El elemento **META** puede ser usado para identificar propiedades de un documento (por ejemplo, el autor, fecha en que debería eliminarse del caché del navegador y ser actualizado, lista de palabras clave para facilitar la tarea a los programas de búsqueda automática de documentos, etc.) y para asignar un valor a esta propiedades.

Cada elemento **META** especifica un par propiedad /valor. El atributo **name** identifica la propiedad y el atributo **content** especifica el valor de la misma.

Por ejemplo, la siguiente declaración asigna un valor a la propiedad *Autor*:

```
<META name="Author" content="Esther del Oro">
```

El atributo **lang** puede usarse con el elemento **META** para indicar el idioma del atributo **content**. Este permite a los navegadores que pueden presentar los documentos como discurso hablado aplicar reglas de pronunciación en función del idioma. Por ejemplo en:

```
<META name="Author" lang="fr" content="Esther del Oro">
```

se declara que el autor es francés.

El atributo **http-equiv** puede utilizarse en lugar del atributo **name** y posee un significado especial cuando los documentos se obtienen por medio de HTTP (*Hypertext Transfer Protocol*). Los servidores HTTP utilizan el nombre de la propiedad especificada por el atributo **http-equiv** para crear una cabecera en la en la respuesta HTTP.

En los ejemplos se ha usado frecuentemente este atributo para especificar la codificación de caracteres del documento:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

Algunos navegadores soportan el uso de **META** para refrescar la página actual después de un determinado número de segundos, con la opción de reemplazarlo por un URI diferente.

```
<meta http-equiv="refresh" content="3; URL=http://default/clio/clio.htm">3
```

El contenido es un número que especifica el retraso en segundos, seguido del URI a cargar cuando se agota el tiempo. Este mecanismo se utiliza generalmente para presentar al usuario una página introductoria.

Un uso común del elemento **META** es especificar palabras clave que permitirán a los programas buscadores mejorar su rendimiento. Cuando varios elementos **META** proporcionan información dependiente del idioma acerca de un documento, los buscadores pueden utilizar el atributo **lang** para mostrar los resultados de la búsqueda de acuerdo a las preferencias de idioma del usuario.

La efectividad de los buscadores también puede mejorarse utilizando el elemento **LINK** para especificar enlaces a traducciones del documento a otros idiomas, enlaces a versiones de documentos en otros formatos (por ejemplo, PDF) y, cuando el documento forma parte de una colección, enlaces al punto de inicio adecuado para hojearla.

³ El ejemplo que se encuentra en la especificación: <META http-equiv="refresh" content="3, http://www.acme.com/intro.html"> no funcionará correctamente con algunas versiones de navegadores.

El elemento **META** también puede utilizar para especificar la información predefinida para un documento en los casos siguientes:

- ✓ El lenguaje de *script* por defecto.
- ✓ El lenguaje de hoja de estilo por defecto.
- ✓ La codificación del documento por defecto.

El elemento **HEAD** establece enlaces entre recursos. Solo puede utilizarse en la cabecera del documento (a diferencia del elemento **A** que solo puede utilizarse en el cuerpo del documento) pero puede aparecer cualquier número de veces. El elemento **LINK** no tiene contenido pero conlleva información sobre la relación entre el documento actual y el documento o recurso enlazado.



EJEMPLO 2.16

Imagine que se dispone de una colección de documentos HTML con diversos relatos del mismo autor y un documento que funciona como tabla de contenidos desde la que recuperar esos documentos. Pues bien, utilizando el elemento **LINK** con el atributo **rel** se pueden definir las relaciones entre los diversos documentos.

En el documento que funciona como tabla de contenidos se utilizará un elemento **LINK** para indicar cuál es el siguiente elemento en la colección de documentos. Para ello, se asignará el valor *next* al atributo **rel**. Para identificar el recurso enlazado, se asignará su URI al atributo **href** del elemento **LINK**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>

<head>
<link rel="Next" href="kafka1.htm">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Tabla de contenidos</title>
</head>

<body>

<h1>Relatos de Kafka</h1>

<ul>
  <li><a href="kafka1.htm">Ante la ley</a></li>
  <li><a href="kafka2.htm">Para que mediten los jinetes</a></li>
  <li><a href="kafka3.htm">El deseo de ser piel roja</a></li>
</ul>
</body>
</html>
```

En el primero de los relatos, el archivo *kafka1.htm*, se utilizarán dos elementos **LINK**: para enlazar el documento con la tabla de contenidos (archivo *tabla.htm*) y con el siguiente documento de la colección (archivo *kafka2.htm*), utilizando el atributo **rel** con los valores *index* y *next*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>

<head>
<link rel="Index" href="tabla.htm">
<link rel="Next" href="kafka2.htm">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Ante la Ley</title>
</head>

<body>

<h1>Ante la ley</h1>

<p>Ante la ley hay un guardián. Un campesino se presenta frente a este guardián y solicita que le permitan entrar en la ley. Pero el guardián contesta que por ahora no puede dejarlo entrar. El hombre reflexiona y pregunta si más tarde lo dejarán entrar.<br>
-Es posible -dice el portero-, pero no ahora.<br>
[...]<br>
</p>

<p><a href="http://huizen.dds.nl/~nfkk/">Franz Kafka</a><br>
Traducción de J.R. Wilcock </p>
</body>
</html>
```

En el segundo relato (archivo *kafka2.htm*) se utilizarán tres elementos LINK: para enlazar el documento con la tabla de contenidos, con el anterior documento de la colección (archivo *kafka1.htm*) y con el siguiente documento de la colección (archivo *kafka3.htm*). Se utilizarán tres elementos LINK asignando a los respectivos atributos *rel* los valores *index*, *prev* y *next*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>

<head>
<link rel="Index" href="tabla.htm">
<link rel="Prev" href="kafka1.htm">
<link rel="Next" href="kafka3.htm">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Para que mediten los jinetes</title>
</head>

<body>

<h1>Para que mediten los jinetes</h1>
```

<p>Si bien se piensa, no es tan envidiable ser vencedor en una carrera de caballos.

La gloria de ser reconocido como el mejor jinete de un país marea demasiado, junto al estrépito de la orquesta, para no sentir a la mañana siguiente cierto arrepentimiento.

La envidia de los contrincantes, hombres astutos y bastante influyentes, nos tristece al atravesar el estrecho pasaje que recorremos después de cada carrera y que pronto aparece desierto ante nuestra mirada, exceptuando algunos jinetes retrasados, que se destacan diminutos sobre el borde del horizonte.</p>

<p>[...] </p>

<p>Franz Kafka

Traducción de J.R. Wilcock </p>

</body>

</html>

Por fin, el último de los documentos (archivo *kafka3.htm*) se relacionará con la tabla de contenidos y con el documento anterior en la colección:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
  "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>
```

<head>

<link rel="Index" href="tabla.htm">

<link rel="Prev" href="kafka2.htm">

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<title>El deseo de ser piel roja</title>

</head>

<body>

<h1>El deseo de ser piel roja</h1>

<p>Si uno pudiera ser un piel roja siempre alerta, cabalgando sobre un caballo veloz, a

través del viento, constantemente sacudido sobre la tierra estremecida, hasta arrojar las

espuelas porque no hacen falta espuelas, hasta arrojar las riendas porque no hacen falta

riendas, y apenas viera ante sí que el campo era una pradera rasa, habrían desaparecido

las crines y la cabeza del caballo.</p>

<p>Franz Kafka

Traducción de J.R. Wilcock </p>

</body>

</html>

Los atributos **rel** y **rev** del elemento **LINK** juegan papeles complementarios. Con **rel** se puede definir la relación existente entre el documento *x* y el documento *y*. A su vez, el documento *y* estará relacionado con el elemento *x* en una relación inversa que se expresará con el atributo **rev**.

Se puede utilizar el elemento **LINK** para facilitar diversa información a los programas buscadores:

- Enlaces a versiones alternativas de un documento escritas en otro idioma.
- Enlaces a versiones alternativas de un documento diseñadas para un dispositivo diferente, por ejemplo, una versión especialmente pensada para su impresión.
- Enlaces a las páginas de inicio de una colección de documentos.

En HTML, los enlaces y referencias a imágenes, microaplicaciones **Java**, programas *cgi*, hojas de estilo, etc., se indican siempre mediante un *URI*. El elemento **BASE** permite especificar a los autores la *URI* en base a la cual se referencian las *URI* relativas. Cuando está presente este elemento deberá aparecer siempre en la cabecera del documento antes de cualquier otro elemento que haga referencia a un recurso externo. La ruta de acceso especificada solo afecta las *URI* a las que se hace referencia en el documento en el que el elemento **BASE** aparece.

ACTIVIDADES 2.15



- Prepare cuatro documentos HTML como los del ejemplo anterior y enlácelos.

2.2.13 INCLUSIÓN DE IMÁGENES

Observe la siguiente página:

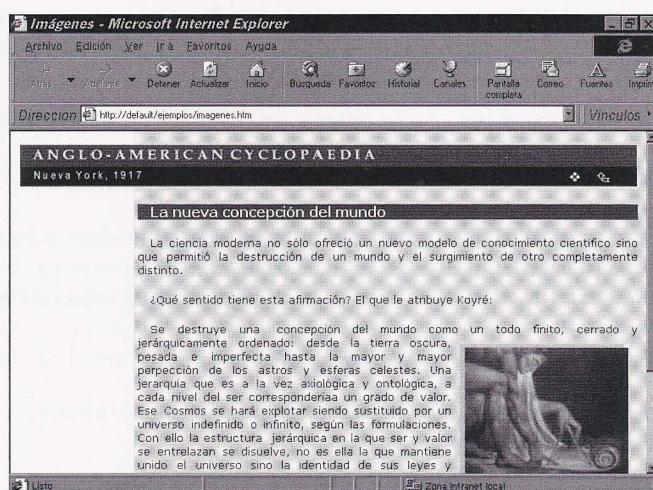
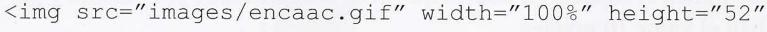


Figura 2.14

Con independencia de la imagen *gif* que, repitiéndose en mosaico forma el fondo de la página, contiene dos imágenes insertadas: la superior, texto convertido en imagen, y la del margen derecho. En el documento fuente están referenciadas mediante las siguientes etiquetas:

■ Para la primera:

```
[...]
<BODY>

<h1 class="indentado">La nueva concepción del mundo</h1>
[...]
```

Cuando el navegador interpreta el documento fuente y llega a la etiqueta **** recupera el recurso cuya ubicación está indicada por el atributo **scr** (de *source*, origen o fuente) y reemplaza el elemento por la imagen (si no se ha indicado mediante el atributo **align** que flote a derecha o izquierda del texto).

Observe que **IMG** es un atributo vacío que, por tanto, se utiliza siempre sin etiqueta de cierre. Con **IMG** hay que utilizar siempre el atributo *scr*. Mediante el mismo se indica la ubicación de la imagen a recuperar siendo su valor un *URI* absoluto o relativo.

Mediante los atributos **width** y **height** (ancho y alto) se indica el espacio que el navegador deberá reservar para la imagen. Es conveniente indicarlo siempre pues, de lo contrario, el navegador esperará a la descarga completa de la imagen antes de seguir presentando el contenido del documento. La anchura y la altura no tienen por qué coincidir con el tamaño real de la imagen dado que el navegador la redimensionará, escalándola, en base a los valores de *width* y *height*. Así, las dimensiones reales de la primera imagen del ejemplo son 700 × 52 píxeles pero se ha redimensionado su anchura para que ocupe el 100% de la superficie horizontal disponible alterando el valor de *width* en términos de porcentaje. Se ha manteniendo el valor real de *height*: *width="100%" height="52"*.

Otro atributo cuyo uso HTML establece como requerido es **alt**. Mediante el mismo se ofrece una breve descripción en texto de la imagen. Esta descripción es muy útil si el navegador está configurado para no mostrar imágenes o si simplemente no las soporta. En lugar de la imagen se mostraría la descripción y algún símbolo que represente la ausencia del objeto. Incluso si el navegador está configurado para mostrar las imágenes, el valor de *alt* puede ser mostrado, por ejemplo, cuando el puntero del ratón se deja inmóvil unos segundos sobre la imagen. Para ofrecer una descripción más extensa del recurso puede utilizarse el atributo **longdesc** cuyo valor es un *URI* al documento que contendría esta descripción. Hasta aquí los atributos implicados en la primera imagen.

■ Las etiquetas correspondientes a la segunda imagen en el documento fuente son:

```
<p class="indentado">Se destruye una; concepción del mundo como un todo finito,
cerrado y jerárquicamente ordenado: desde la tierra
oscura, pesada e imperfecta hasta la mayor y mayor perfección de los astros y esferas
celestes.
[...]
```

En este ejemplo se encuentran tres atributos no utilizados en el anterior: *align*, *hspace* y *vspace*.

Observe en primer lugar como el elemento **IMG** se introduce en medio de un párrafo y, sin embargo, el navegador no inserta exactamente en ese punto la imagen sino que la sitúa en el margen derecho fluyendo

el texto a su izquierda. Este efecto, que el objeto flote y el texto fluja a su alrededor, se consigue utilizando el atributo **align** con los valores *right*, para flotar a la derecha, o *left*, para flotar a la izquierda. En un ejemplo posterior se verá el efecto del resto de los valores admitidos por *align* cuando se utiliza con imágenes (los valores difieren cuando se utiliza con texto). El control del flujo en torno a los elementos flotantes se consigue con el atributo **clear** del elemento BR.

El valor del atributo **hspace** Indica la cantidad de espacio en blanco que debe ser insertado a izquierda y derecha de la imagen. Con **vspace** se indica el espacio en blanco que debe ser insertado por encima y por debajo de la imagen. En el ejemplo se utiliza el valor 10 píxeles.

En este ejemplo también se ha reducido el tamaño de la imagen mediante el uso de *width* y *height*. Las dimensiones reales de la imagen son 483×383 píxeles⁴. Fácilmente se podría convertir la imagen en un hiperenlace a la imagen con sus dimensiones reales:

```
<a href="images/alegoria.jpg">
align="right" hspace="10" vspace="10" border="0"></a>
```

Observe cómo ahora el elemento **IMG** se incluye dentro de un elemento **A** que convierte la imagen en punto de origen de un hiperenlace. Se ha añadido a **IMG** un atributo **border**. El valor predefinido para este atributo dependerá del navegador, pero cuando la imagen funciona como enlace de hipertexto, los navegadores la rodearán con un borde del mismo color que el resto de los hiperenlaces, para evitar que el navegador muestre este borde se ha puesto su valor a "0".

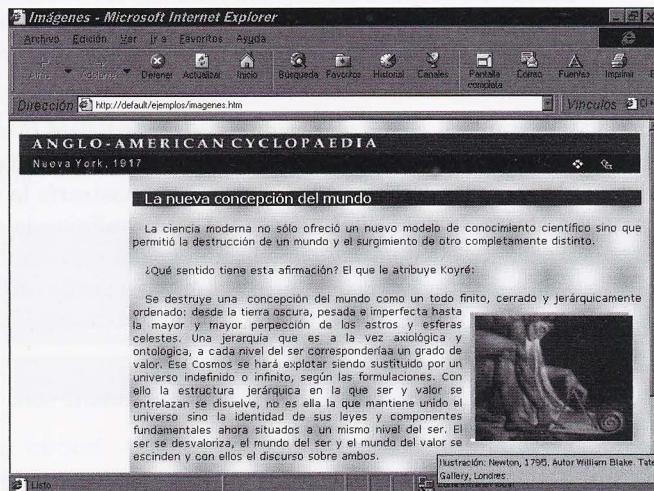


Figura 2.15

Al pulsar sobre la imagen, el navegador recuperará la imagen *alegoria.jpg* (ubicada en el directorio *images*) mostrándola en sus dimensiones reales.

⁴ Para reproducir el ejemplo utilice cualquier imagen GIF, JPG o PNG que tenga disponible y redimensionela al inscrustrarla en el documento fuente.

En la siguiente ilustración podrá observar el comportamiento del resto valores posibles para el atributo **align** cuando se utiliza con imagen (existen bastantes extensiones propietarias que afectan a la alineación de imágenes).

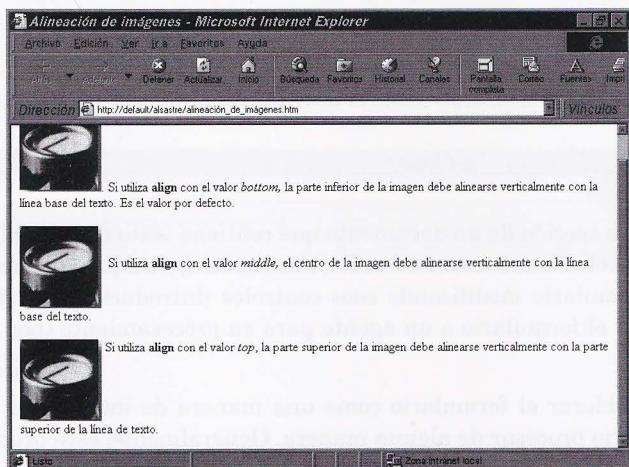


Figura 2.16

A continuación, se muestra el documento fuente correspondiente a la Figura 2.16:

```
<html>
<head>
<title>Alineación de imágenes</title>
</head>
<body>
<p>. Si utiliza <strong>align</strong> con el valor <em>bottom, </em>la parte inferior de la imagen debe alinearse verticalmente con la línea base del texto. Es el valor por defecto.</p>
<p>&ampnbsp Si utiliza <strong>align</strong> con el valor <em>middle, </em>el centro de la imagen debe alinearse verticalmente con la línea base del texto.</p>
<p> Si utiliza <strong>align</strong> con el valor <em>top</em>, la parte superior de la imagen debe alinearse verticalmente con la parte superior de la línea de texto.</p>
</body>
</html>
```

Hasta aquí lo esencial acerca de IMG.



La inclusión de imágenes puede conseguirse también usando el elemento **OBJECT**.

ACTIVIDADES 2.16



- Prepare un documento HTML que incluya alguna imagen con texto alineado.

2.2.14 FORMULARIOS

Un formulario HTML es una sección de un documento que contiene texto normal, etiquetas, elementos especiales llamados *controles* (cuadros de selección, botones de radio, menús, etc.), y etiquetas para estos controles. Los usuarios generalmente “rellenan” el formulario modificando esos controles (introduciendo texto, seleccionando opciones de un menú, etc.), antes de enviar el formulario a un agente para su procesamiento (por ejemplo, un servidor Web, un servidor de correo, etc.).

Lo más importante es considerar el formulario como una manera de interactuar con el usuario que generará una información que es necesario procesar de alguna manera. Generalmente, este procesamiento se lleva a cabo por programas ubicados en el servidor web, también es frecuente dirigir la información recopilada mediante un formulario a una dirección de correo electrónico. La programación del servidor mediante diversos métodos (CGI, ISAPI, NSAPI, ASP) y lenguajes de programación (PERL, C, Tcl, VBScript de servidor (ASP), shell de UNIX, etc., queda fuera del alcance de este libro. Únicamente se van a introducir los elementos que permiten crear formularios sin entrar en el tratamiento de la información enviada al servidor mediante los mismos.

Se va a comenzar comentando un ejemplo sencillo:



EJEMPLO 2.17

```
<form action="mailto:alsastre@redestb.es" method="post" enctype="text/plain">
  <p>Apellidos:<input type="text" name="aapp" size="50"><br>
  Nombre:<input type="text" name="Nombre" size="25"><br>
  Dirección de correo electrónico:<input type="text" name="email" size="20"><br>
  ¿Cómo llegó hasta aquí? <select name="Vias" size="1">
    <option>Al azar</option>
    <option>Alguien le indicó el URI de la página</option>
    <option selected>A través de un programa buscador</option>
    <option>Mediante un enlace desde otra página</option>
  </select></p>
  <p>&nbsp;<input type="submit" value="Enviar"> <input type="reset" value="Borrar">
  </p>
</form>
```

El formulario está delimitado por las etiquetas de inicio y cierre del elemento **FORM**. Este lleva asociado dos atributos **action** y **post**. Mediante el primero se indica que los datos introducidos en el formulario serán procesados enviándolos a una dirección de correo electrónico. Mediante el segundo se indica el método del protocolo HTTP utilizado para enviar esos datos. Si el método es *get* los datos introducidos en el formulario, separados mediante el

carácter “?”, serán añadidos al URI que constituye el valor del atributo *action* y el nuevo URI así construido es enviado al servidor para su procesamiento; si el método es *post* los datos introducidos en el formulario se incluirán en el cuerpo del formulario y serán enviados para su procesamiento, en el ejemplo, a una dirección de correo electrónico. Además, el atributo **enctype** indica el tipo de contenido (tipo MIME) utilizado para enviar el formulario.

El contenido del elemento **FORM** del ejemplo consiste en varios controles de formulario. Además contiene otros elementos que se pueden encontrar en cualquier documento HTML, como son **P** y **BR**.

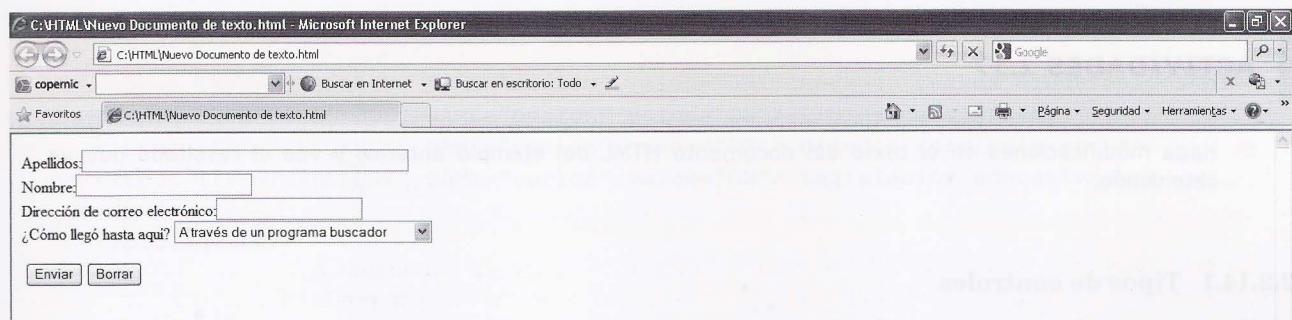


Figura 2.17

Los controles de formulario permiten al usuario operar sobre los mismos de diversas formas de acuerdo al tipo de control: introduciendo una cadena de texto, seleccionando una o varias opciones de un menú, pulsando un botón con el ratón, etc. En el ejemplo aparecen tres tipos de controles:

- De **entrada de texto** (*text input*). Introducido en el ejemplo mediante el elemento INPUT con un atributo *type="text"* asociado:

```
<input type="text" name="aapp" size="50">
<input type="text" name="Nombre" size="25">
<input type="text" name="email" size="20">
```

Mediante el atributo **name** se ha asignado un nombre a cada control y mediante el atributo **size** se ha establecido su tamaño en número de caracteres.

- Un **control de menú**. Ofrece al usuario varias opciones a seleccionar. En el ejemplo, se obtiene mediante el elemento **SELECT** combinado con varios elementos **OPTION**.

```
<select name="Vias" size="1"> ... </SELECT>
```

Con el elemento **SELECT** se declara un menú de opciones. El atributo *name* asigna también ahora un nombre al control y el atributo *size* indica ahora el número de valores de la lista de opciones del menú que el navegador debe presentar simultáneamente.

Las diversas opciones se obtienen mediante elementos **OPTION**:

```
<option value="Al azar">Al azar</option>
```

Observe que la opción del menú viene dada por el contenido del elemento **OPTION**. Si **OPTION** lleva asociado el atributo **selected**, se establecerá como opción preseleccionada.

- **Botones.** En el ejemplo se crean mediante el elemento **INPUT** con el atributo *type="submit"* para crear un botón que transmita los datos introducidos por el usuario (*submit*) y con el atributo *type="reset"* para crear un botón que reinicia o borra (*reset*) los datos introducidos por el usuario.

```
<input type="submit" value="Enviar"> <input type="reset" value="Borrar">
```

Observe como **INPUT** es un siempre un elemento vacío. En este caso el atributo **value** establecerá el texto que se mostrará en el control.

ACTIVIDADES 2.17



- Haga modificaciones en el texto del documento HTML del ejemplo anterior y vea el resultado que va obteniendo.

2.2.14.1 Tipos de controles

En un formulario pueden crearse los siguientes tipos de controles:

- **Botones (Buttons).** Pueden ser de tres tipos:

- **Botones submit.** Cuando se activan, los datos del formulario son enviados, por el método correspondiente, para su procesamiento por un programa en el servidor web o a un servidor de correo. Un formulario puede contener más de un botón *submit*.
- **Botones reset.** Cuando se activa, un botón *reset* restablece todos los controles a sus valores iniciales. Todo control tiene un *valor inicial* y un *valor actual*. En general, el valor inicial de un control puede ser especificado con el valor del atributo *value* del elemento. El valor actual es el mismo que valor inicial si todavía el usuario o un *script* no ha operado sobre el control. Después, toma el valor introducido por el usuario o el *script*. El valor inicial de los controles no cambia. Así cuando un formulario se reinicia al activar el botón *reset*, borrando los datos introducidos por el usuario, el valor actual de cada control cambia a su valor inicial.
- **Botones push.** El funcionamiento de un botón *push* depende de scripts asociados con atributos de evento. Cuando un evento ocurre (por ejemplo, el usuario pulsa sobre el botón o pasa el ratón por encima) el *script* asociado se desencadena.

Se debe especificar el lenguaje de *script* para un botón *push* mediante una declaración de *script* por defecto (con el elemento **META**).

Los botones pueden crearse con los elementos **BUTTON** e **INPUT**. Para crear un botón *submit* con el elemento **INPUT** deberá llevar asociado el atributo *type* con el valor *submit* (como se vio en el ejemplo). Para crear un botón *reset* con el elemento **INPUT** deberá llevar asociado el atributo *type* con el valor *reset*. Para crear un botón *push* con el elemento **INPUT** el valor de *type* deberá ser *button*.

ACTIVIDADES 2.18



- Cree un formulario en un documento HTML que incorpore distintos tipos de botones y vea el resultado en el navegador.

- **Casillas de selección (Checkboxes).** Se crean mediante el elemento *INPUT* asociado al atributo *type* con el valor *checkbox*. Son de tipo sí/no, on/off o verdadero/falso, es decir, marcado o no marcado. Varias casillas de selección pueden compartir el mismo nombre de control, mediante el atributo *name*, de manera que el usuario pueda seleccionar varios valores. Si la casilla está marcada, tomará el valor del atributo *value*. El atributo *checkbox* hace que la casilla esté preseleccionada. Por ejemplo:

```
<p>¿Desearía más información sobre: ?</p>
<blockquote>
    <p><input TYPE="CHECKBOX" NAME="opcion" value="ON" checked> Material
    audивisual<br>
        <input TYPE="CHECKBOX" NAME="opcion" value="ON" checked> Textos electrónicos<br>
        <input TYPE="CHECKBOX" NAME="opcion" value="ON"> Libros de texto<br>
        <input TYPE="CHECKBOX" NAME="opcion" value="ON"> Diseños curriculares<br>
        <input TYPE="CHECKBOX" NAME="opcion" value="ON"> Legislación educativa<br>
    </p>
</blockquote>
```

ACTIVIDADES 2.19

- Cree un formulario en un documento HTML que incorpore casillas de selección y vea el resultado en el navegador.

- **Botones de radio (Radio Buttons).** Se crean con *INPUT* y el atributo *type* con el valor *radio*. Los botones de radio se comportan como las casillas de selección excepto que cuando comparten el mismo nombre de control son mutuamente excluyentes. El atributo *checked* tiene la misma función que en las casillas de selección. Por ejemplo:

```
<p>¿Sexo?</p>
<blockquote>
    <p><input TYPE="radio" NAME="sexo" value="Hombre" checked> Hombre<br>
        <input TYPE="radio" NAME="sexo" value="Mujer"> Mujer<br>
    </p>
</blockquote>
```

ACTIVIDADES 2.20

- Cree un formulario en un documento HTML que incorpore botones de radio y vea el resultado en el navegador.

■ **Menú desplegable (Menus).** En el primer ejemplo se vio cómo se obtiene un menú desplegable de opciones a partir de los elementos SELECT y OPTION. También admite el elemento **OPTGROUP** para agrupar lógicamente las opciones. Más adelante, se exemplificará el uso de éste último. Si el elemento SELECT lleva asociado el atributo *multiple*, el usuario podrá elegir varias opciones, de lo contrario, solo podrá optar por una (como en el ejemplo). Si se usa el atributo *selected* con OPTION, se está indicando que se trata de una opción preseleccionada. Si varios elementos OPTION llevan asociado el atributo *selected*, el elemento SELECT que los contiene deberá llevar asociado el atributo *multiple*. Con el atributo *value* se puede indicar el valor inicial de OPTION, si se omite, como en el primer ejemplo, el valor vendrá dado por el contenido del elemento.

ACTIVIDADES 2.21



➤ Cree un formulario en un documento HTML que incorpore menús desplegables y vea el resultado en el navegador.

■ **Cuadros de texto (Text Input).** Este control puede de ser de dos tipos: cuadro de texto de una sola línea y cuadros de texto de varias líneas. Ambos permiten la entrada de texto al usuario. El primero se obtiene mediante el elemento INPUT y asignando el valor *text* al atributo *type*. El segundo se consigue mediante el elemento **TEXTAREA**.

ACTIVIDADES 2.22



➤ Cree un formulario en un documento HTML que incorpore cuadros de texto y vea el resultado en el navegador.

■ **Selección de archivos (File Select).** Este tipo de control permite al usuario seleccionar archivos para que sus contenidos puedan enviarse con un formulario. Se obtiene con INPUT asociado al atributo *type* con el valor *file*. Por ejemplo:

```
<form action="mailto:alsastre@redestb.es" method="post" enctype="multipart/form-data">
    <p>Fichero a enviar <input type="file" name="fichero"> </p>
</form>
```

ACTIVIDADES 2.23



➤ Cree un formulario en un documento HTML con el ejemplo anterior y vea el resultado en el navegador.

Observe como el navegador muestra un botón **examinar** que permite recorrer la estructura de directorios para seleccionar el archivo deseado.

Puede utilizarse el atributo *value* para indicar un nombre de archivo predeterminado. Para utilizar este tipo de control hay que utilizar el atributo *enctype* con el valor *multipart/form-data* asociado al elemento *FORM*.

- **Controles ocultos** (*Hidden controls*). Estos controles no son visualizados por el navegador pero sus contenidos son enviados con el resto del formulario. Se utilizan para almacenar información relacionada con los intercambios cliente/servidor que de otra forma se perdería. Se obtienen con *INPUT* y tomando el atributo *type* el valor *hidden*.
- **Controles de objeto** (*Object controls*). Se crean con el elemento **OBJECT**. Los valores asociados al control de objeto serán enviados para su procesamiento junto con el resto de controles del formulario.
- **Otros controles**. El elemento *INPUT* permite crear otro control mediante la asignación del valor *password* al atributo *type*. Es un control de texto en el que la cadena de caracteres introducidos por el usuario se enmascara mediante el uso de un carácter, normalmente un asterisco. De esta forma, el texto introducido por el usuario se oculta en pantalla pero se transmite por la red al servidor como texto perfectamente legible por lo que su uso como mecanismo de seguridad no ofrece garantías.

Los controles creados con *type="image"* son botones *submit* gráficos. El URI de la imagen se especifica mediante el atributo *src*. Cuando se pulsa sobre la imagen, el formulario se envía y las coordenadas de la imagen donde se ha pulsado con el ratón también son pasadas al servidor. El valor de la coordenada *x* se mide en píxeles desde la parte izquierda de la imagen, la coordenada *y* también en píxeles desde la parte superior de la misma. En función de las coordenadas pasadas al servidor pueden desencadenarse diversas acciones en éste. La especificación para HTML recomienda usar en su lugar mapas de imagen gestionados por el cliente o varios botones *submit* cada uno con su gráfico en lugar de un solo botón de imagen.

A continuación, se muestra un ejemplo con controles *password* e *image*:

```
<form action="mailto:alsastre@redestb.es" method="post" enctype="text/plain">
    <p>Introduca su login :<br>
    <input type="password" name="login" size="20"><br>
    Introduzca su password:<br>
    <input type="password" name="password" size="20"></p>
    <p><input type="image" src="images/enviar.gif"></p>
        <p><input type="reset" value="Borrar"></p>
</form>
```

ACTIVIDADES 2.24



- Cree un formulario en un documento HTML con el ejemplo anterior y vea el resultado en el navegador.

2.2.14.2 Principales atributos del elemento FORM

- **Accept.** Especifica una lista de tipos de contenido (MIME) separados por comas que el servidor que procese el formulario es capaz de manejar correctamente.

- **Accept-charset.** Especifica una lista de sistemas de codificación de caracteres para entradas de datos que deben ser aceptados por el servidor que procesa el formulario. El valor es una lista de *charset* delimitada por espacios y/o comas. El servidor debe interpretar esta lista como una disyunción exclusiva, es decir, el servidor debe aceptar solo uno de los sistemas de codificación de la lista. El valor por defecto para este atributo es la expresión reservada *UNKNOWN*.
- **Action.** Especifica un agente de procesamiento para el formulario. El valor podría ser un URI HTTP para enviar el juego de datos a un programa ubicado en un servidor web o un URI *mailto* para enviar los datos a una dirección de correo electrónico.
- **Enctype.** Especifica el tipo de contenido (MIME) usado al enviar el formulario al servidor (cuando el método es *post*). El valor predeterminado para este atributo es *application/x-www-form-urlencoded*. El valor *multipart/form-data* debe usarse en combinación con un elemento *INPUT* de tipo *type="file"*.
- **Method.** Especifica el método HTTP utilizado para enviar el juego de datos del formulario. Los valores posibles son: *get* (valor por defecto) y *post*.

2.2.14.3 Principales atributos del elemento INPUT

- **Checked.** Cuando el atributo *type* toma los valores *radio* o *checkbox*, indicará si el botón está marcado. Es un atributo booleano, es decir alterna entre los valores sí/no, on/off o verdadero/falso.
- **Maxlength.** Cuando el valor del atributo *type* es *text* o *password*, este atributo indicará el número máximo de caracteres que pueden ser introducidos por el usuario. Este número puede sobrepasar el especificado mediante *size*, en cuyo caso el navegador deberá ofrecer un mecanismo de *scrolling*. El valor por defecto es un número ilimitado de caracteres.
- **Name.** Asigna un nombre al control.
- **Size.** Indica al navegador la anchura inicial del control. La anchura viene dada en píxeles excepto cuando el atributo *type* tiene los valores *text* o *password*. En estos casos, el valor estará referido al número de caracteres.
- **Src.** Cuando el valor de *type* es *image*, indicará la localización de la imagen utilizada para decorar un botón *submit* (enviar).
- **Type.** Especifica el tipo de control a crear. Los valores (tipos) posibles: *text*, *password*, *checkbox*, *radio*, *submit*, *reset*, *file*, *hidden*, *image* y *button*. El valor por defecto es *text*.
- **Value.** Especifica el valor inicial del control. Es opcional excepto cuando el atributo *type* tiene el valor *radio*.

2.2.14.4 Principales atributos del elemento SELECT

- **Name.** Asigna un nombre al control.
- **Size.** Si el elemento *SELECT* es presentado por el navegador como un cuadro de lista que admite desplazamientos, este atributo indicará el número de filas de la lista que deben ser visibles al mismo tiempo.
- **Multiple.** Si está presente, indicará que el usuario puede seleccionar varias opciones. Si no se utiliza solo puede optarse por una de las opciones.

2.2.14.5 Principales atributos del elemento OPTION

- **Label.** Permite presentar una etiqueta más corta para una opción que el contenido del elemento OPTION.
- **Selected.** Si está presente, indicará que la opción está preseleccionada.
- **Value.** Establece el valor inicial del control. Si no se especifica este atributo, el valor inicial vendrá dado por el contenido del elemento OPTION.

2.2.14.6 Principales atributos del elemento TEXTAREA

- **Cols.** Establece la anchura visible de las líneas de texto mediante un número entero que indica promedio de caracteres a introducir en cada línea. El usuario debe poder introducir líneas más largas, para lo que el navegador debe facilitar algún mecanismo de desplazamiento.
- **Name.** Asigna un nombre al control.
- **Rows.** Establece el número de líneas de texto visibles para el usuario aunque el usuario pueda introducir más líneas de texto para lo cual el navegador debe proporcionar un mecanismo de desplazamiento.

2.2.14.7 Etiquetas

El elemento **LABEL** se utiliza para especificar etiquetas para controles que no tienen asociadas etiquetas implícitas, como entradas de texto, botones de radio, menús de opciones o casillas de selección.

El atributo **for** permite asociar explícitamente una etiqueta con un control. Para ello, el valor del atributo *for* deberá ser el mismo que el del atributo **id** del elemento de control asociado y deberá estar presente en este. Más de un elemento **LABEL** puede ser asociado con el mismo control creando referencias múltiples mediante el atributo *for*.



EJEMPLO 2.18

Fíjese en el siguiente documento fuente:

```
<form action="mailto:alperez@redestb.es" method="post" enctype="text/plain">
    <label for="aapp">Apellidos: </label><input type="text" id="aapp" size="50"><br>
    <label for="nombre">Nombre: </label><input type="text" id="Nombre" size="25"><br>
    <label for="email">Dirección de correo electrónico: </label><input type="text" id="email" size="20"><br>
    <label for="Vias">¿Cómo llegó hasta aquí?</label> <select id="Vias" size="1">
        <option>Al azar</option>
        <option>Alguien le indicó el URI de la página</option>
        <option>A través de un programa buscador</option>
        <option>Mediante un enlace desde otra página</option>
    </select>
    <input type="submit" value="Enviar"> <input type="reset" value="Borrar">
</form>
```

Observe cómo se ha prescindido del atributo *name* y se ha utilizado el atributo *id* y cómo coinciden los valores de *n* y *for* en las asociaciones etiqueta/control.

También pueden asociarse etiqueta y control de manera implícita. Para ello, el control ha de utilizarse como contenido del elemento **LABEL**. Como en:

```
<LABEL>  
Fecha de nacimiento  
<INPUT type="text" name="nacido">  
</LABEL>
```

En este uso, **LABEL** solo puede contener un control. Observe que ahora no es necesario utilizar los atributos *for* e *id*.

ACTIVIDADES 2.25



- Cree un formulario con el ejemplo 2.18 y compruebe su funcionamiento.

2.2.14.8 Elemento Button

Como ya se dijo anteriormente, los botones pueden crearse tanto con el elemento INPUT como el elemento **BUTTON**. Ambos funcionan de manera similar, la diferencia estriba en que BUTTON admite contenidos (INPUT es un elemento vacío) y un diseño y presentación más sofisticados.



EJEMPLO 2.19

Este ejemplo muestra cómo crear botones *submit* y *reset* mediante **BUTTON**:

```
<BUTTON name="submit" value="submit" type="submit"> Enviar datos ;  
</BUTTON>  
<BUTTON name="reset" type="reset"> Borrar datos ;  
</  
BUTTON>
```

Observe cómo tanto texto como imágenes (elemento IMG), forman parte del contenido de **BUTTON**.

ACTIVIDADES 2.26



- Cree un formulario con el ejemplo 2.16 y compruebe su funcionamiento.

2.2.14.9 Controles desactivados y de solo lectura

En determinadas circunstancias puede ser interesante poder desactivar un control o presentarlo en modo de solo lectura.

Para conseguir desactivar un control, se introduce el nuevo atributo **disabled** que es soportado por los elementos BUTTON, INPUT, OPTGROUP, OPTION, SELECT y TEXTAREA.

Para conseguir que un control sea de solo lectura y, por tanto, el usuario no pueda modificarlo, se introduce el atributo **readonly** que es soportado por los elementos INPUT y TEXTAREA.

Ambos atributos son atributos *booleanos*, es decir, tienen el valor sí/no según estén o no presentes o modificando el elemento asociado. Ambos atributos pueden cambiar dinámicamente de valor mediante un *script*.

2.2.14.10 Estructura del formulario: elementos OPTGROUP, FIELDSET y LEGEND

Estos tres elementos permiten estructurar lógicamente los elementos incluidos en el formulario de manera que el usuario capte de manera intuitiva su sentido.

El elemento **FIELDSET** permite agrupar temáticamente los controles y etiquetas relacionados.

El elemento **LEGEND** asigna un encabezado a FIELDSET.

Por su parte, **OPTGROUP** asigna una etiqueta a un grupo de opciones definidas mediante los elementos **SELECT** y **OPTION**.



EJEMPLO 2.20

Este ejemplo muestra el uso de estos elementos y sintetiza todo lo tratado respecto a los formularios:

```
<html>
<head>
<title>Encuesta al profesorado</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<style>
<!--
BODY { font-family: Verdana; font-size: 10pt; background-color: rgb(0,128,128) }
LEGEND { font-weight: bold; text-align: center; font-family: Verdana; font-size: 12pt }
LABEL { color: rgb(0,128,128); font-weight: bold }
FIELDSET { background-color: rgb(192,192,192); margin-bottom: 10px; padding: 10px }
-->
</style>
</head>
<body>
<h1>¿Por favor! Si es usted docente responda a las siguientes cuestiones:<br>
</h1>
<input type="text" name="inactivo" size="100"
```

value="Este formulario se visualizará mejor con navegadores de la última generación"
disabled readonly>

```
<form action="mailto:alsastre@redestb.es" method="post" enctype="text/plain">
<input type="hidden" name="tipo" value="Docentes"><FIELDSET>
<LEGEND accesskey="1">
1. Información personal</LEGEND>
<label for="aa">Apellidos:</label><input NAME="Apellidos" type="text" size="30" id="aa"
tabindex="1"> <label for="nombre">Nombre:</label><input type="text" name="Nombre"
size="20" id="nombre" tabindex="2"><br>
<label for="fn">Fecha de nacimiento:</label><input type="text" name="Nacido"
SIZE="8" id="fn"
tabindex="3"> <label>Sexo:</label><input TYPE="RADIO"
VALUE="Masculino" tabindex="4" name="Sexo"> Hombre
<input TYPE="RADIO"
VALUE="Femenino" tabindex="5" name="Sexo"> Mujer
</FIELDSET>
<FIELDSET>
<LEGEND accesskey="2">
2. Información de contacto</LEGEND>
<label for="telefono">Teléfono</label><input NAME="Telefono" SIZE="10"
maxlength="10"
id="telefono" tabindex="6"> <label for="email">Email:</label><input
type="text"
name="email" size="15" id="email" tabindex="7">
</FIELDSET>
<FIELDSET>
<LEGEND accesskey="3">
3. Datos profesionales</LEGEND>
<label for="ct">Centro de trabajo</label><input type="text" name="Centro" size="30"
id="ct"
tabindex="12"> <label for="localidad">Localidad:</label><input type="text"
name="localidad" size="30" id="localidad" tabindex="13"><BR>
<label for="esfun">¿Es funcionario?</label><input type="checkbox"
name="Funcionario"
value="ON" id="esfun" tabindex="14"> <label for="nrp">NRP:</label><input
type="password"
name="NRP" size="15" id="nrp" tabindex="15"> <label for="sn">Seminario/Nivel:</label>
<select NAME="Seminario" size="2" id="sn" multiple tabindex="16">
<option>Lengua y Literatura </option>
<option>Matemáticas </option>
<option>Geografía e Historia </option>
<option>Biología y Geología </option>
<option>Tecnología </option>
<option>Física y Química </option>
<option>Educación Plástica y Visual </option>
```

```
<option>Música </option>
<option>Educación Física </option>
<option>Cultura y Lenguas Clásicas </option>
<option selected>Ética y Filosofía </option>
<option>Primer Ciclo ESO </option>
<option>Segundo ciclo ESO </option>
<option selected>Bachillerato </option>
</select>
</FIELDSET>
<FIELDSET>
<LEGEND accesskey="4">
4. Por favor, facilítenos la siguiente información:</LEGEND>
<label for="Vias">¿Cómo llegó hasta aquí?</label>
<select name="Vias" id="Vias" size="1" tabindex="17">
    <option value="Azar">Al azar</option>
    <option value="URI">Alguien le indicó el URI de la página</option>
    <option value="Buscador">A través de un programa buscador</option>
    <option value="Enlace">Mediante un enlace desde otra página</option>
</select>
<P>
<label for="comentarios">Introduzca aquí sus comentarios:</label><BR>
<textarea name="Comentarios" rows="5" name="comentarios" cols="60" id="comentarios" tabindex="18"></textarea>
<blockquote>
    <label>¿Desearía más información sobre ... ?</label><br>
    <input TYPE="CHECKBOX" NAME="Mas" value="audio" checked tabindex="19">
Material
    audivisual<br>
    <input TYPE="CHECKBOX" NAME="opcion" value="etext" checked tabindex="20">
Textos
    electrónicos<br>
    <input TYPE="CHECKBOX" NAME="opcion" value="libros" tabindex="21"> Libros de texto<br>
    <input TYPE="CHECKBOX" NAME="opcion" value="diseños" tabindex="22">
    Diseños curriculares<br>
    <input TYPE="CHECKBOX" NAME="opcion" value="leyes" tabindex="23">
Legislación educativa</p>
</blockquote>
</FIELDSET>
<BUTTON name="submit" value="submit" type="submit" accesskey="e" tabindex="25">
    [E]nviar datos &nbsp; </BUTTON> <BUTTON name="reset" type="reset" accesskey="b" tabindex="26">
    [B]orrar datos &nbsp; </BUTTON>
</form>
</body>
</html>
```

Mostrará una página web parecida a la siguiente:

The screenshot shows a Microsoft Internet Explorer window with the title bar 'Encuesta al profesorado - Microsoft Internet Explorer'. The menu bar includes Archivo, Edición, Ver, Tra, Favoritos, Ayuda. Below the menu is a toolbar with icons for Back, Forward, Stop, Refresh, Home, Search, Favorites, History, Favorites, Screen, Mail, Fonts, Print, and Edit. The address bar shows the URL 'http://personal.redetb.es/alcاستة/encuesta_al_profesorado.html'. The page content starts with the heading 'a las siguientes cuestiones:' followed by a note: 'Este formulario se visualizará mejor con navegadores de la última generación'. The form is divided into three sections: 1. Información personal (Apellido: [input], Nombre: [input], Fecha de nacimiento: [input], Sexo: Hombre [radio] Mujer [radio]), 2. Información de contacto (Teléfono: [input], Email: [input]), and 3. Datos profesionales (Centro de trabajo: [input], Localidad: [input], ¿Es funcionario? [checkbox] NRP: [input], Seminario/Nivel: [dropdown menu showing 'Lengua y Literatura' and 'Matemáticas']).

Figura 2.18

ACTIVIDADES 2.27



- Cree un formulario con el ejemplo 2.20 y compruebe su funcionamiento.

2.2.15 TABLAS

Las tablas se utilizan en HTML no solo para estructurar información en forma de filas y columnas, también ha sido el recurso más poderoso del que han dispuesto los autores para maquetar sus páginas, consiguiendo efectos de diseño imposibles de obtener en HTML por otros medios. Con los recursos de posicionamiento que las hojas de estilo ponen a disposición del autor, este uso de las tablas poco a poco ha ido disminuyendo. La especificación de HTML recomienda no usar las tablas con esta finalidad por los problemas que puede presentar para medios de acceso a Internet no visuales. En todo caso, lo primero a tener en cuenta es que en las celdas de una tabla no solo se puede colocar texto sino casi cualquier cosa: texto, texto preformateado, imágenes, vídeos, enlaces, formularios, campos de formulario, otras tablas y otros objetos.

Los elementos básicos para construir una tabla son TABLE, TR, TD y TH. El par de etiquetas <TABLE> ... </TABLE> engloba toda la tabla. El par <TR> ... </TR> una fila. Y los pares <TD> ... </TD> y <TH ... </TH> una celda. Las celdas definidas mediante TD incluyen datos, las definidas mediante TH cabeceras. La etiqueta de fin de TR, TD y TH puede omitirse. Se va a mantener en los ejemplos para facilitar la lectura del código.



EJEMPLO 2.21

Observe en el siguiente ejemplo cómo los elementos TR delimitan las ocho filas de la tabla englobando cada una, excepto la octava, ocho celdas. Cada una de éstas últimas está definida por un elemento TH o TD. Tenga en cuenta que no todas las filas tienen porque tener el mismo número de celdas y que pueden definirse celdas vacías.

```
<html>
<head>
<title>Simple</title>
</head>

<body>

<table border="0" cellspacing="0" cellpadding="2">
    <caption>Distancias medias estimadas en los planetas con respecto al Sol</caption>
    <tr bgcolor="#008080">
        <td></td>
        <th>Copérnico</th>
        <th>Kepler</th>
        <th>Siglo XX</th>
    </tr>
    <tr bgcolor="#C0C0C0">
        <th>Mercurio</th>
        <td>0,3763</td>
        <td>0,389</td>
        <td>0,387</td>
    </tr>
    <tr bgcolor="#008080">
        <th>Venus</th>
        <td>0,7193</td>
        <td>0,724</td>
        <td>0,723</td>
    </tr>
    <tr bgcolor="#C0C0C0">
        <th>Tierra</th>
        <td>1,0000</td>
        <td>1,000</td>
        <td>1,000</td>
    </tr>
    <tr bgcolor="#008080">
        <th>Marte</th>
        <td>1,5198</td>
        <td>1,523</td>
        <td>1,524</td>
    </tr>
    <tr bgcolor="#C0C0C0">
        <th>Júpiter</th>
        <td>5,2192</td>
        <td>5,200</td>
    </tr>

```

```

<td>5,202</td>
</tr>
<tr bgcolor="#008080">
<th>Saturno</th>
<td>9,1743</td>
<td>9,510</td>
<td>9,539</td>
</tr>
<tr>
<td align="right" colspan="4">En unidades astronómicas</td>
</tr>
</table>
</body>
</html>

```

Mostrará una página web parecida a la siguiente:

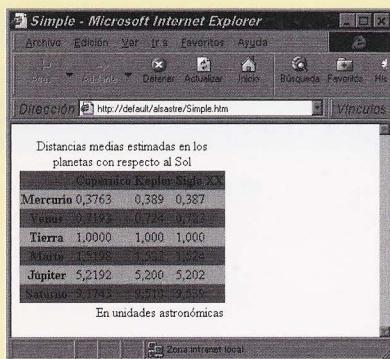


Figura 2.19

Observa que:

El elemento **TABLE** delimita toda la estructura.

El elemento **TR** delimita las filas actuando como un contenedor de celdas.

Las celdas se definen mediante los elementos **TD** y **TH**. TD define una celda que contiene datos. TH define una celda que contiene etiquetas de cabecera. El navegador presentará el contenido de estas últimas centrado y en negrita. Por defecto, el navegador presentará el contenido de los elementos TD alineado a la izquierda.

El navegador establecerá automáticamente el ancho total de la tabla y el de cada una de las celdas.

La última fila contiene una sola celda que se extiende a lo largo de las cuatro columnas de la estructura de la tabla. Esto se ha conseguido mediante el uso del atributo **colspan** cuyo valor es un entero que indica el número de columnas de la estructura de la tabla que ha de ocupar la celda. Si en lugar de extenderse por varias columnas, se quisiera extender la celda por varias filas se utilizaría el atributo **rowspan** aplicado a TD o TH.

La tabla tiene un título *Distancias medias estimadas en los planetas con respecto al sol*. El título de la tabla, que es opcional, se define mediante el elemento **CAPTION**. Este elemento solo puede ir

inmediatamente después de la etiqueta de inicio del elemento TABLE y solo puede haber un elemento CAPTION por tabla. Por defecto, el título se centrará en la parte superior pero admite el atributo **align** con los valores *top* (parte superior de la tabla), **bottom** (parte inferior de la tabla), *left* (izquierda de la tabla) y *right* (derecha de la tabla).

Se han utilizado tres atributos con el elemento TABLE que afectan a toda la tabla:

```
<table border="0" cellspacing="0" cellpadding="2">
```

El atributo **border** establece la anchura en píxeles del recuadro que rodea la tabla. En el ejemplo se ha puesto a 0 y, por tanto, no se visualizará el recuadro. Bastaría ponerlo a 1 para que éste se mostrara, así como la rejilla interior:

	Copérnico	Kepler	Sistema X
Mercurio	0,3763	0,389	0,387
Venus	0,7192	0,724	0,723
Tierra	1,0000	1,000	1,000
Marte	1,5192	1,523	1,524
Júpiter	5,2192	5,200	5,202
Saturno	9,1743	9,210	9,209

En unidades astronómicas

Figura 2.20

El atributo **cellspacing** controla el espacio, en píxeles, que el navegador debe mantener entre cada celda y entre los bordes de la tabla. En el ejemplo se ha puesto a 0 para hacer la tabla más compacta. Si el valor fuera 2 píxeles se vería de la manera siguiente:

	Copérnico	Kepler	Sistema X
Mercurio	0,3763	0,389	0,387
Venus	0,7192	0,724	0,723
Tierra	1,0000	1,000	1,000
Marte	1,5192	1,523	1,524
Júpiter	5,2192	5,200	5,202
Saturno	9,1743	9,210	9,209

En unidades astronómicas

Figura 2.21

El atributo **cellpadding** controla el espacio de relleno, en píxeles, entre el contenido y los bordes de la celda. Si en lugar de 2 se hubiera puesto su valor a 0, la tabla se vería de la manera siguiente:

A screenshot of a Microsoft Internet Explorer window displaying a table. The table has a header row 'Copérnico Kepler Siglo XX' and seven data rows for the planets: Mercurio, Venus, Tierra, Marte, Jupiter, Saturno, and Urano. The first seven rows have a dark gray background color. The last row, 'Urano', has a white background color. The table is titled 'Distancias medias estimadas en los planetas con respecto al Sol'. A note at the bottom right says 'En unidades astronómicas'.

Copérnico Kepler Siglo XX		
Mercurio	0,3763	0,389 0,387
Venus	0,7193	0,724 0,723
Tierra	1,0000	1,000 1,000
Marte	1,5198	1,523 1,524
Jupiter	5,2192	5,200 5,202
Saturno	9,1743	9,510 9,539
En unidades astronómicas		

Figura 2.22

En las siete primeras filas se ha utilizado el atributo **bgcolor** para establecer el color de fondo de todas las celdas de esas filas. Este atributo también puede utilizarse con el elemento TABLE para establecer un color de fondo para toda la tabla y con los elementos TD y TH para establecerlo para celdas individuales. Se han utilizado dos valores de color, en notación hexadecimal, para ir alternando el color de cada fila. Si no se hubieran establecido colores de fondo la tabla se vería de la manera siguiente:

A screenshot of a Microsoft Internet Explorer window displaying the same table as Figure 2.22, but without the alternating row colors. All rows, including the header and the last one, have a white background. The table structure is identical to Figure 2.22.

Copérnico Kepler Siglo XX		
Mercurio	0,3763	0,389 0,387
Venus	0,7193	0,724 0,723
Tierra	1,0000	1,000 1,000
Marte	1,5198	1,523 1,524
Jupiter	5,2192	5,200 5,202
Saturno	9,1743	9,510 9,539
En unidades astronómicas		

Figura 2.23

El contenido de todas las celdas TD se alinea horizontalmente, por defecto, a la izquierda. Para la última celda, que ocupa varias columnas, se ha querido controlar la alineación poniendo el contenido a la derecha de la celda. Para ello se ha utilizado el atributo **align** con el valor *right*. Para controlar la alineación vertical dentro de la celda se podría haber utilizado el atributo **valign**. Ambos atributos pueden utilizarse con los elementos TR (afectando a todas las celdas de la línea) y con TH y TD (afectando a celdas individuales).

ACTIVIDADES 2.28

- Cree una tabla basándose en el ejemplo 2.21, realice las modificaciones que considere y compruebe su funcionamiento.

2.2.16 MARCOS

Las **páginas de marcos** (traducción habitual del término **frames**), son una forma de organizar la presentación de los documentos HTML en la ventana del navegador. Esta se divide en varias regiones independientes, de manera que el explorador puede mostrar un documento distinto en cada una de ellas.



EJEMPLO 2.22

Considere el siguiente ejemplo:

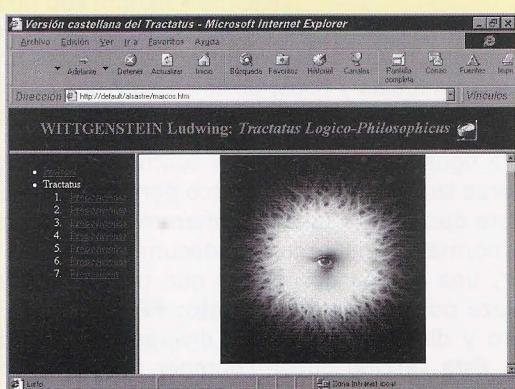


Figura 2.24

La ventana del navegador aparece dividida en tres marcos, regiones o subventanas. El marco superior contiene el autor y título de una obra y un pequeño ícono. Debajo de él aparecen otras dos regiones. La inferior izquierda contiene una lista con las secciones de la obra a las que se tiene acceso. En el marco derecho se presentarán esos contenidos. Cada una de esas regiones alberga un documento HTML.

Si se activa el enlace de hipertexto “1. Proposiciones” del marco de la izquierda, el contenido del marco de la derecha cambiará mostrando en su lugar el documento al que el atributo **href** de ese enlace hace referencia. En el ejemplo, se visualizaría lo siguiente:

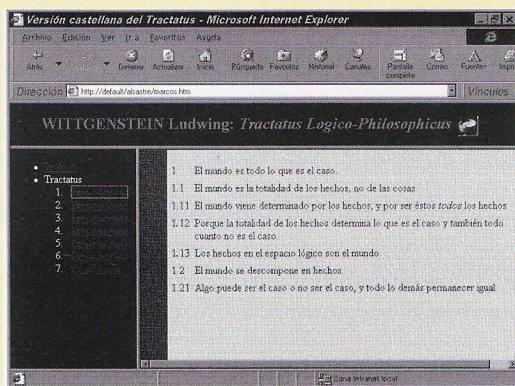


Figura 2.25

Al pulsar en el ícono del marco superior, en el marco inferior derecho se cargaría un documento HTML con la versión completa de la obra. Al activar cualquiera de los enlaces del marco izquierdo se cargaría en el marco derecho un nuevo documento con nuevas secciones.

En un documento normal, al activar un enlace el documento que lo contiene es reemplazado por el recurso a que el enlace hace referencia, es decir, un documento es reemplazado con otro en la ventana del navegador. En un documento con marcos, al activar uno de sus enlaces no es necesario que el documento que lo contiene sea reemplazado, el objetivo del enlace se puede mostrar en otro de los marcos de la ventana. Así sucede en el ejemplo, los documentos del marco superior y el izquierdo se mantienen cuando se activan los enlaces presentes en ellos y los documentos a los que apuntan esos hiperenlaces se cargan en el marco de la derecha.

Esta manera de organizar la presentación de los documentos puede ser idónea para determinados contenidos y, además, ofrece una manera de organizar la navegación en un sitio web. Este segundo uso de los marcos, que despierta fobias y filias, posiblemente sea desplazado por las posibilidades de emplazamiento de contenidos que ofrecen los lenguajes de hojas de estilo.

En el ejemplo de la primera figura están implicados cuatro documentos HTML. Tres se muestran en el marco superior, en el marco izquierdo y en el marco derecho, y un cuarto, que es el que establece la estructura de marcos. Este documento, al que llamaremos *página de marcos*, tiene una estructura diferente a los documentos normales sin marcos. Un documento normal tiene una sección de cabecera, *HEAD*, y un cuerpo, *BODY*; una página de marcos que tiene también una cabecera *HEAD* pero el elemento *BODY* se reemplaza por un nuevo elemento: **FRAMESET**. Es esta sección del documento la que establece el número y dimensiones de los diversos marcos que aparecerán en la ventana del navegador del usuario. Esta sección puede contener un elemento **NOFRAMES** para ofrecer un contenido alternativo a los navegadores más antiguos que no soportan marcos o a cualquier navegador para el que el usuario haya desactivado la presentación en forma de marcos.

El documento fuente de la *página de marcos* del ejemplo es el siguiente:

```
<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
           "http://www.w3.org/TR/REC-html40/frameset.dtd">
<head>
<title>Versión castellana del Tractatus</title>
</head>
<frameset rows="25%,*">
  <frame name="titular" scrolling="no" noresize src="encabezado.htm">
  <frameset cols="189,*">
    <frame name="contenido" src="Contenidos.htm">
    <frame name="principal" src="texto.htm">
  </frameset>
  <noframes>
    <body>
      <p>Esta página usa marcos, pero su explorador no los admite.</p>
    </body>
  </noframes>
</frameset>
</html>
```

Mediante el elemento **FRAMESET** se dibujan los marcos. Mediante el elemento **FRAME** se le asigna nombre, contenido y aspecto a cada marco individual. El ejemplo tiene tres marcos a pesar de que

solo se han utilizado dos elementos FRAMESET. Esto es posible porque el segundo elemento FRAMESET va anidado en el primero. Para mayor claridad se van a indicar otros ejemplos en los que no se utiliza anidamiento y después se explicará éste.

El elemento FRAMESET controla el diseño de los marcos mediante los atributos *rows* y *cols*. Con **rows** se define el número de regiones horizontales o filas. El atributo **cols** define el número de regiones verticales o columnas. Ambos atributos pueden ser utilizados simultáneamente para crear una rejilla. El valor del atributo es una lista de longitudes separadas por comas. El número de miembros de la lista se corresponde con el número de filas o columnas. Las longitudes pueden expresarse en píxeles, porcentajes o longitudes relativas.

Por ejemplo, una página con dos marcos verticales:

```
<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
 "http://www.w3.org/TR/REC-html40/frameset.dtd">

<head>
<title>Marcos verticales</title>
</head>

<frameset cols="*, *">
  <frame name="izquierdo" src="mizquierdo.htm">
  <frame name="derecho" src="mderecho.htm">
  <noframes>
    <body>
      <p>Esta página usa marcos, pero su navegador no los admite.</p>
    </body>
  </noframes>
</frameset>
</html>
```

En este ejemplo, la línea `<frameset cols="*, *">` dibuja dos marcos verticales (dos columnas). La anchura de las mismas se expresa mediante longitudes relativas indicándose al programa navegador que divide el espacio horizontal disponible en dos partes iguales. Se podría haberlo expresado con longitudes expresadas en porcentajes (`<frameset cols="50%,50%">`) o con longitudes expresadas en píxeles (para una pantalla con resolución de 800 x 600: `<frameset cols="400,400">`).

Los dos elementos FRAME definen el contenido de cada marco individual. Ambos reciben un nombre mediante el atributo **name** y para ambos se indica el documento inicial que ha de cargarse en cada marco mediante el atributo **src** cuyo valor es el URI correspondiente al documento. Para todo marco ha de indicarse el documento inicial a cargar en el marco que luego puede ser sustituido por cualquier otro para el que se indique, de la manera que se verá, que ha de mostrarse en ese marco.

Ahora un ejemplo en el que se dibujan dos marcos horizontales:

```
<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
 "http://www.w3.org/TR/REC-html40/frameset.dtd">

<head>
<title>Marcos horizontales</title>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
</head>
```

```

<frameset rows="*, *">
    <frame name="superior" src="msuperior.htm">
    <frame name="inferior" src="minferior.htm">
    <noframes>
        <body>
            <p>Esta página usa marcos, pero su navegador no los admite.</p>
        </body>
    </noframes>
</frameset>
</html>

```

Ahora el elemento **FRAMESET** se modifica con el atributo **rows** para dibujar dos filas del mismo tamaño expresado en longitudes relativas. Los dos elementos *FRAME* permiten asignar nombre a cada marco y establecer el *URI* del documento de inicio que ha de mostrarse en cada uno.

También se pueden utilizar simultáneamente ambos atributos en un elemento FRAMESET (e, incluso, los documentos que se carguen en alguno de los marcos pueden ser a su vez páginas de marcos).

El siguiente documento fuente se visualizaría como una rejilla con cuatro marcos todos del mismo tamaño:

```

<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
           "http://www.w3.org/TR/REC-html40/frameset.dtd">
<head>
<title>Rejilla</title>
</head>
<frameset rows="50%,50%" cols="50%,50%">
    <frame name="superior izquierdo" src="msi.htm">
    <frame name="superior derecho" src="msd.htm">
        <frame name="inferior izquierdo" src="mii.htm">
        <frame name="inferior derecho" src="mid.htm">
</frameset>
<noframes>
<body>
<p>Esta página usa marcos, pero su navegador no los admite.</p>
</body>
</noframes>
</html>

```

Cuando no se utiliza el atributo **rows**, cada columna se extiende por todo lo largo de la página. Si es **cols** el que no se utiliza, cada fila se extenderá por todo lo ancho. Si ninguno de los atributos se utiliza, el marco ocupará exactamente el tamaño de la página.

Los marcos se crean de izquierda a derecha para las columnas y de arriba a abajo para las filas. Cuando se utilizan simultáneamente ambos atributos, las regiones se crean de izquierda a derecha en la primera fila, de izquierda a derecha en la segunda y así sucesivamente.

Ahora que ha quedado todo más claro, se va a volver al primer ejemplo en el que un elemento **FRAMESET** se anidaba dentro de otro.

El primer elemento FRAMESET, puesto que tiene asignado el atributo *rows*, dibuja dos filas o regiones verticales. A la primera se le reservan 64 píxeles y para la segunda se indica al navegador mediante una longitud relativa que le asigne el resto del espacio:

```
<frameset rows="64,*">
    <frame name="titular" scrolling="no" noresize target="contenido" src="encabezado.htm">
```

Inmediatamente después, el primer elemento **FRAME** asigna un nombre y un documento inicial a cargar para el marco de la primera fila (recuerde que los marcos horizontales se crean de arriba a abajo).

Inmediatamente después (antes de introducir la etiqueta de cierre del primer *frameset*), aparece otro elemento **FRAMESET** que afecta al marco de la segunda fila creado con el **FRAMESET** anterior:

```
<frameset cols="25%,*>
    <frame name="contenido" target="principal" src="Contenidos.htm">
    <frame name="principal" src="texto.htm">
</frameset>
```

Este **FRAMESET** divide la segunda fila en dos columnas, a la primera le reserva el 25% del espacio horizontal y a la segunda, el resto de la anchura disponible. Se les asigna nombre y contenido (de izquierda a derecha). Ahora una etiqueta de cierre delimita este segundo *frameset*.

Solo queda introducir el elemento **NOFRAMES** para aquellos navegadores que no soporten marcos:

```
<noframes>
    <body>
        <p>Esta página usa marcos, pero su navegador no los admite.</p>
    </body>
</noframes>
```

Y por último, introducir la etiqueta de cierre del primer **FRAMESET**: `</frameset>`

Suponga ahora que en lugar de crear dos columnas en la segunda fila se quisiera crearlas para la primera y no dividir en columnas la segunda fila. El código se reescribiría así:

```
<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
           "http://www.w3.org/TR/REC-html40/frameset.dtd">
<head>
<title>Versión castellana del Tractatus</title>
</head>
<frameset rows="64,*">
    <frameset cols="25%,*>
        <frame name="contenido" src="Contenidos.htm">
        <frame name="principal" src="texto.htm">
    </frameset>
    <frame name="titular" scrolling="no" noresize src="encabezado.htm">
</frameset>
<noframes>
    <body>
        <p>Esta página usa marcos, pero su explorador no los admite.</p>
    </body>
</noframes>
</frameset>
</html>
```

Evidentemente ahora el diseño no es adecuado para el contenido:

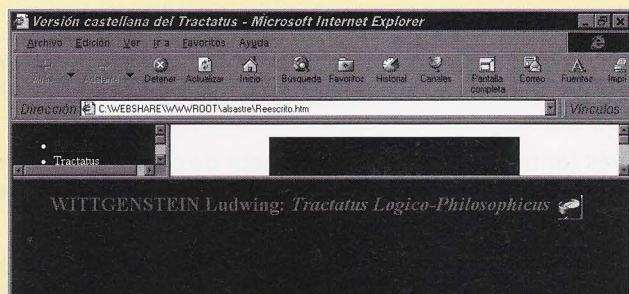


Figura 2.26

El elemento **FRAME** no solo permite identificar y dar un contenido inicial a cada marco definido por **FRAMESET**, también permite controlar su aspecto mediante los atributos:

- **Frameborder.** Permite establecer si el navegador dibujará o no un borde para el marco. Valores posibles:
 - "1": El navegador dibujará un borde separador entre el marco actual y cada marco inmediato
 - "0": El navegador no dibujará un borde entre el marco actual y los inmediatos. Advierta que puede aparecer un borde al lado de este marco si así se especifica para otros marcos adyacentes.
- **Marginheight.** Indica la cantidad de espacio que el navegador debe dejar entre los contenidos del marco y sus márgenes superior e inferior. El valor debe ser mayor que un píxel. El valor por defecto depende del navegador.
- **Marginwidth.** Indica la cantidad de espacio que el navegador debe dejar entre los contenidos del marco y sus márgenes derecho e izquierdo. El valor debe ser mayor que un píxel. El valor por defecto depende del navegador.
- **Noresize.** Es un atributo *booleano*, si se usa se indica que el marco no es redimensionable por el usuario.
- **Scrolling.** Controla la utilización de barras de desplazamiento con el marco. Valores posibles:
 - "auto": el navegador presenta barras de desplazamiento, o un dispositivo análogo, cuando es necesario. Es el valor por defecto.
 - "yes": indica al navegador que utilice siempre barras de desplazamiento o un dispositivo análogo.
 - "no": indica al navegador que no utilice nunca barras de desplazamiento.

En el primer ejemplo se han utilizado los atributos **noresize** para impedir el redimensionamiento del marco horizontal, y se ha puesto el atributo **scrolling** a *no* para que el navegador no cree barras de desplazamiento en ese marco. En el marco de la columna izquierda **scrolling** toma el valor por defecto: *auto*. Puesto que todo el marco es visible, el navegador no dibujará las barras de desplazamiento. Lo que sí ocurre para el marco de la columna derecha.

Hasta ahora, solo se ha mostrado el código fuente del documento para la página de marcos. Sin embargo, en el ejemplo están implicados otros tres documentos, es decir, los que constituyen el contenido inicial de cada marco. Ahora, se va a echar una ojeada su código:

El documento del marco horizontal superior es *encabezado.htm* y su código fuente es:

```
<html>
<head>
<title>Tractatus Logico-Philosophicus</title>
<base target="principal">
</head>
<body bgcolor="#000000" text="#FFFFFF">
<h2 align="center"><font color="#919100">WITTGENSTEIN Ludwig: <em>Tractatus Logico-Philosophicus</em></font><a href="textoc.htm" target="principal"></a></h2>
</body>
</html>
```

El documento contiene una imagen *png* que funciona como enlace al documento *textoc.htm* y se muestra en el código siguiente extrayéndolo del código anterior:

```
<a href="textoc.htm" target="principal"></a>
```

Pues bien, ¿en cuál de los tres marcos se mostraría ese documento al activar el enlace? Observe que se ha utilizado con el elemento **A** y el atributo *target*. Su valor es el nombre de uno de los marcos que se han creado. Mediante esta utilización de *target* se indica que el recurso enlazado ha de mostrarse en el marco *principal* de la página de marcos.

El documento inicial del marco vertical de la izquierda es *Contenidos.htm* y su código fuente es:

```
<html>
<head>
<title>Contenidos</title>
<base target="principal">
</head>

<body bgcolor="#000000" text="#FFFFFF">

<ul>
<li><a href="prologo.htm">Prólogo</a></li>
<li>Tractatus<ol>
    <li><a href="nivel_1.htm">Proposiciones</a></li>
    <li><a href="nivel_2.htm">Proposiciones</a></li>
    <li><a href="nivel_3.htm">Proposiciones</a></li>
    <li><a href="nivel_4.htm">Proposiciones</a></li>
    <li><a href="nivel_5.htm">Proposiciones</a></li>
    <li><a href="nivel_6.htm">Proposiciones</a></li>
    <li><a href="nivel_7.htm">Proposición</a></li>
</ol>
</li>
</ul>
</body>
</html>
```

Observe que cada ítem de la lista es un hiperenlace. Ahora no se ha utilizado el atributo **target** con cada elemento A. ¿En qué marco se cargarán los documentos enlazados? Observe la línea:

```
<base target="principal">
```

Mediante el uso del elemento **BASE** con el atributo **target** se ha establecido un marco de destino predeterminado para todos los enlaces existentes en este documento, el marco *principal*" del ejemplo. Así se evita tener que utilizar el atributo *target* para cada uno de los ítems de la lista del ejemplo.

El documento que constituye el contenido inicial del marco vertical de la derecha es *texto.htm* y su código fuente es:

```
<html>
<head>
<title>Texto</title>
</head>
<body>
<p align="center"><a href="http://default/contact/" target="_top"></a></p>
</body>
</html>
```

En este documento se vuelve a encontrar el atributo *target* modificando a un elemento A. Pero ahora, su valor no es el nombre de ninguno de los marcos que se han creado en la página de marcos, sino *_top*:

```
<a href="http://default/contact/" target="_top">
```

¿En qué marco se mostrará? En ninguno, el documento enlazado ocupará la ventana completa. Esto es así porque en HTML existen varios nombres reservados de marcos que pueden utilizarse con el atributo *target*. Dichos nombres reservados son:

- **_blank**. El navegador deberá cargar el documento enlazado en una nueva ventana.
- **_self**. El navegador cargará el recurso enlazado en el mismo marco desde el que es invocado.
- **_parent**. El navegador cargará el documento en el juego de marcos padre del marco actual. Se utiliza cuando un documento mostrado en un marco es a su vez una página de marcos.
- **_top**. El navegador cargará el documento en ventana completa cancelando la estructura de marcos.

Así pues el atributo *target* es el que permite hacer operativa la estructura de marcos. Permite que cualquier recurso enlazado se cargue en un marco al que se le ha dado nombre mediante el atributo *name* y también permite ubicarlo mediante estos nombres predefinidos.

El atributo *target* puede ser utilizado con los elementos que permiten crear algún tipo de enlace: *A*, *LINK*, *AREA* y *FORM*.

El navegador deberá atenerse a un orden de precedencia para determinar en qué marco ha de marcar un recurso enlazado. Este orden es el siguiente:

- Si un elemento tiene establecido el atributo *target* a un marco conocido, cuando el elemento sea activado, el recurso debe cargarse en ese marco.
- Si un elemento no tiene especificado un atributo *target* pero en el documento del que forma parte se ha utilizado el elemento *BASE*, el atributo *target* de *BASE* determinará el marco
- Si no ocurre ni lo uno ni lo otro, el recurso deberá ser cargado en el mismo cargo que contiene al elemento desde el que es invocado.
- Si un atributo *target* se refiere a un marco desconocido *X*, el navegador deberá crear una nueva ventana y marco, asignará el nombre *X* al marco y cargará el recurso en este nuevo marco.

ACTIVIDADES 2.29



- Cree unas páginas web con marcos basándose en el ejemplo 2.22 y compruebe su funcionamiento.
- Haga las modificaciones que considere en las páginas del ejercicio anterior y pruebe su funcionamiento.

2.2.17 CAPAS

Las **capas** se pueden definir como páginas que se pueden incrustar dentro de otras. Los atributos de una capa (posición, visibilidad, etc.), como los de cualquier otro elemento HTML, pueden definirse dentro de una hoja de estilo. Su contenido, en cambio, siempre deberá ser especificado dentro de la parte principal de la página. Como se puede ver, se está siguiendo la filosofía de separar el contenido y la forma de representarlo.

Sin duda, lo más importante de las capas es la posibilidad que presentan de ser movidas y modificadas desde un lenguaje de *script*. Desgraciadamente, las implementaciones de los distintos navegadores pueden ser incompatibles entre sí, por lo que resulta complicado escribir código que funcione en distintas plataformas.

La única manera común de definir capas para distintos navegadores es mediante hojas de estilo.

La definición de una capa sigue la misma estructura que la que se usaba para decidir las características de una etiqueta con el atributo *id*:

```
<STYLE TYPE="text/css">
  #lacapa {position:absolute; top:80px; left:10px;}
</STYLE>
```

Esto colocaría a la capa que se ha denominado *lacapa* a 10 píxeles de la izquierda de la página y a 80 píxeles del comienzo de la misma. Pero, ¿dónde se escribirá lo que se desea que contenga la capa? Para ello, se utilizará el elemento **ID**:

```
<DIV ID="lacapa">
  <H1>El título de la capa</H1>
  <P>Aquí es donde iría el texto.
  ...
</DIV>
```

Otra forma de insertar capas es utilizar el elemento **SPAN** en lugar del elemento **DIV** (**SPAN** se utiliza del mismo modo que **DIV**, y es compatible con un mayor número de navegadores).

Las capas que se han definido se colocan en una posición determinada de la página. Pero existe otro tipo de capas llamadas flotantes cuya colocación depende, en cambio, de la posición dentro del código fuente de la página donde se las haya colocado. Pueden definirse de la manera siguiente:

```
<STYLE TYPE="text/css">  
    #flotante {position: relative; left: 20px; top: 100px; }  
</STYLE>
```

ACTIVIDADES GLOBALES



- ▶ Escriba un documento HTML que contenga elementos presentados a lo largo de los epígrafes de HTML.
- ▶ Valide el documento usando, por ejemplo, el validador de W3C (<http://validator.w3.org/check>).

2.3 XHTML

En sus inicios, el lenguaje HTML fue ampliamente adoptado por la comunidad, lo que dio lugar a que el HTML empezaría a crecer incorporando etiquetas nuevas, pero sin demasiado control. Esto provocó que determinados contenidos solo fueran accesibles desde determinados navegadores. Por tanto, esta falta de estandarización dio lugar a que el HTML dejara de ser tan sencillo como era en un principio.

Es en ese momento, cuando XML hace aparición y se postula como alternativa al caos generado por el crecimiento de HTML. Sin embargo, XML está orientado al intercambio de información más que a la presentación de la misma. Como solución se propone utilizar una combinación de XML y HTML y de esta forma y de un modo natural surge **XHTML (eXtensible HyperText Markup Language)**, como dialecto de XML pero con las características orientadas a presentación de información de HTML.

Por tanto, una de las principales diferencias es que HTML proviene de SGML mientras que XHTML proviene de XML, lo que condiciona su sintaxis, entre otras características.

2.3.1 SINTAXIS

Una de las diferencias de XHTML frente a HTML es que los documentos que genera son documentos bien formados de XML. Esto es, un documento XHTML deberá cumplir las mismas normas que un documento XML. A continuación, se destacan algunas:

- Debe existir un único elemento raíz llamado `<html>`
- Se debe incluir una instrucción de procesamiento XML para indicar la versión de XML y la codificación de caracteres usada.

```
<?xmlversion="1.0" encoding="UTF-8" ?>
```

- Debe existir una declaración de DOCTYPE para poder validar el documento frente al DTD correspondiente. Por ejemplo:

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

- Las etiquetas de apertura deberán tener su correspondiente par de cierre teniendo en cuenta los posibles anidamientos de etiquetas.
- Como XML diferencia entre mayúsculas y minúsculas las etiquetas válidas en XHTML deberán ir siempre en minúsculas.
- Los valores de los atributos se escriben siempre entre comillas dobles.
- Elementos vacíos que en HTML son correctos como
, en XHTML deben escribirse como
 o como
</br>.

2.3.2 VERSIONES

El lenguaje XHTML surgió a partir de la versión HTML 4.0 alrededor del año 2000 y se denominó XHTML 1.0. A partir de entonces se empezaron a desarrollar algunas nuevas versiones como el XHTML 1.1 (en 2001). Posteriormente, se desarrollaron borradores de nuevas versiones como la 1.2 o la 2.0 que nunca llegaron a estandarizarse y a usarse ampliamente. En la actualidad, se está trabajando sobre XHTML 5.0.

Para la versión 1.0 existen varias versiones dependiendo de lo restrictivo que se quiere que sea el DTD correspondiente. Existen las siguientes versiones:

- **Transitional**: en este caso se permite el uso de algunos elementos que están en desuso o desaconsejados (del inglés *deprecated*) como por ejemplo, <center> o .
- **Strict**: este, sin embargo, no permite incluir los elementos marcados por la W3C como *deprecated*.
- **Frameset**: permite escribir documentos XHTML usando marcos.

Por otro lado, la versión 1.1 surge para dotar a la versión 1.0 de modularidad. De esta forma, se permite el uso de módulos externos combinados con XHTML 1.1. Una segunda edición de la versión 1.1 se publicó por parte del W3C en 2010 para corregir algunas erratas en la especificación original e incluir el uso de XML Schema.

ACTIVIDADES 2.30



- Reescriba el documento HTML de la actividad global anterior usando XHTML.
- De nuevo, compruebe que el documento sea válido usando, por ejemplo, el validador del consorcio W3C.

2.4 HOJAS DE ESTILO

Cuando surgió HTML, los elementos que poseía no estaban pensados realmente para dar formato o estilo a un documento sino que los elementos estaban pensados para dar estructura al documento. En posteriores revisiones del estándar, algunos elementos fueron añadidos para, por ejemplo, cambiar la fuente o el color de determinados elementos. Esto realmente provocó un problema ya que, por un lado, la información de estructura se entremezcla con la información de estilo y por otro, al desarrollar sitios web de gran tamaño, cambiar el estilo del sitio requería ir uno a uno cambiando cada una de los elementos de estilo.

Por estas dos razones, el consorcio W3C desarrollo lo que denominaron **CSS** (*Cascade Style Sheets*). CSS son hojas de estilo en cascada y permiten definir el estilo de cada uno de los elementos presentes en un documento HTML (o XHTML), pudiendo realizar agrupaciones de elementos por tipos, de forma que modificar el estilo de todos los elementos de un tipo requiera simplemente un único cambio. Por otro lado, se dice que son en cascada ya que se pueden incluir diversas hojas de estilos pudiendo modificar las nuevas hojas estilos definidos en hojas previas, lo que permite una gran versatilidad.

2.4.1 SINTAXIS DE CSS

La sintaxis básica de una hoja de estilos CSS consta de dos partes. La primera es un selector para indicar qué elementos se van a ver afectados por el estilo. La segunda parte es el estilo propiamente dicho y puede escribirse en una línea o varias para facilitar la lectura.

```
selector { propiedad: valor; propiedad2: valor; }
```

A continuación, se muestra un ejemplo en el que se define el color y el tamaño de fuente para los párrafos (es decir para las etiquetas <p>).

```
p {  
    color: #f00;  
    font-size: 10px;  
}
```

El selector se puede definir de diferentes formas dependiendo de los elementos a los que se desee cambiar el estilo:

- **Tipo de elemento:** como en el ejemplo anterior, se puede escribir como selector el nombre de una etiqueta haciendo que todos los elementos asociados a esa etiqueta adquieran un determinado estilo.
- **Selector por identificador:** para poder dar estilo a un único elemento se puede usar el atributo “id” dentro de HTML (o XHTML) de forma que se puede definir un estilo para un determinado valor de “id” usando el carácter “#” al empezar el selector. Por ejemplo, en el código HTML podría aparecer:

```
<p id="parrafo">Párrafo de ejemplo</p>
```

Y mediante el siguiente estilo CSS se podría dar estilo únicamente a dicho elemento.

```
#parrafo {color: #c00;}
```

- **Clase de elemento:** para poder dar estilo a un grupo de elementos se usa el atributo *class* de HTML. Para ello, en el documento CSS se definirá el selector empezando por el carácter “.”. Por ejemplo, se podrían definir los siguientes elementos en HTML:

```
<p class="clase1">Párrafo de ejemplo</p>
<p class="clase1">Otro párrafo de ejemplo</p>
<ul class="clase1">
<li>Item 1</li>
<li>Item 2</li>
</ul>
```

Y mediante la definición CSS del estilo de la clase 1 se podría dar estilo a los tres de forma conjunta:

```
.clase1 {color: #c00;}
```

Además, se puede hacer una combinación de selectores mezclando el tipo de elemento con la clase. De esta forma si se definen dos párrafos y una lista no ordenada en HTML de la siguiente manera:

```
<p class="clase1">Párrafo de ejemplo</p>
<p>Otro párrafo de ejemplo</p>
<ul class="clase1">
<li>Item 1</li>
<li>Item 2</li>
</ul>
```

Se podría dar estilo únicamente a los párrafos de la clase “clase1” usando el siguiente selector:

```
p.clase1 {color: #c00;}
```

Las características existentes en CSS a las que se puede dar estilo son muy numerosas por lo que no pueden describirse en este libro. Se recomienda acudir al consorcio W3C para una revisión completa de las mismas.

2.4.2 USO DE CSS DENTRO DE HTML Y XHTML

Una vez que se ha definido la hoja de estilos, es necesario incorporarla desde el documento HTML o XHTML.

Se pueden utilizar tres métodos:

- Usar el elemento `<link>` dentro de la sección `<head>` del documento:
- ```
<head>
<link rel="stylesheet" type="text/css" href="est.css" />
</head>
```
- Definir la hoja de estilos dentro del propio documento HTML usando la etiqueta `<style>` dentro de la sección `<head>` del documento:

```
<head>
<style type="text/css">
body {background-color: #f32;}
h1 {Font-size: 120%; }
p {margin:20px;}
</style>
</head>
```

- Definir el estilo directamente en las propias etiquetas de HTML usando el atributo *style*. El estilo definido de esta forma tiene precedencia sobre los definidos de las dos maneras anteriores.

```
<p style="color: black; margin:10px">Ejemplo de párrafo</p>
```

Las tres formas pueden utilizarse para un mismo documento. La elección de una, otra o varias depende de la planificación con que el autor organice su trabajo y de las características del propio documento.

En general, el primer método, utilizar una hoja de estilo externa al documento, es el más racional pues permite dar una apariencia homogénea a todas las páginas que se integran en un sitio web y permite realizar todos los cambios necesarios sin reescribir el código fuente de cada página.

Si se opta por incrustar la información de estilo en la cabecera del documento, su alcance estará limitado al cuerpo de ese mismo documento. Si se introduce la información de estilo en línea, es decir, aplicándola a elementos puntuales del cuerpo de un documento, el alcance estará limitado a ese elemento de ese documento.

Sea cual sea el método o métodos utilizados, el autor deberá especificar el lenguaje de estilo por defecto para la información de estilo asociada al documento. Esto puede hacerse mediante la utilización del elemento **META**, caso más frecuente, o mediante cabeceras HTTP si se tiene control del servidor.

Por ejemplo, para especificar que el lenguaje de estilo por defecto es CSS el autor deberá incluir en la cabecera del documento la siguiente declaración:

```
<META http-equiv="Content-Style-Type" content="text/css">
```

Si se utiliza una cabecera HTTP, la anterior declaración equivaldría a:

Content-Style-Type: text/css

En todo caso, según la especificación, los navegadores deberán determinar el lenguaje de hoja de estilo atendiendo, en orden de prioridad, a:

1. Una declaración **META** en la cabecera del documento.
2. Una cabecera *http*.
3. Si no se especifica nada de lo anterior el navegador asumirá que el lenguaje por defecto es *text/css*.

#### 2.4.3 INFORMACIÓN DE ESTILO EN EL CUERPO DEL DOCUMENTO

Para ello se utiliza el atributo **style** que es admitido por casi todos los elementos. Por ejemplo, se podría asociar información de estilo a una determinada cabecera de un documento:

```
<H1
style="font-family: Verdana, fantasía; font-size: 12pt; background-color: rgb(128,0,0);
color: rgb(0,0,128); font-weight: 900; text-decoration: none; letter-spacing: 10px;
line-height: 25px; text-transform: capitalize; border: 5px dotted rgb(0,0,0)">Esta
cabecera incorpora información de estilo mediante el uso del atributo style</p>
```

Observe cómo las reglas de estilo adoptan la forma *propiedad:valor* y van separadas por un punto y coma “;”. Todo ello se encierra entre comillas como con cualquier otro atributo.

Toda la información de estilo del ejemplo solo sería aplicable a esa instancia determinada del elemento **H1**. Esta manera de incluir la información de estilo es la menos flexible de todas por ser la menos reutilizable.

#### 2.4.4 INFORMACIÓN DE ESTILO EN LA CABECERA DEL DOCUMENTO

Ahora no se recurrirá al atributo *style* sino al elemento **STYLE**. Se puede colocar cualquier número de elementos **STYLE** en la cabecera del documento. Si un navegador no soporta hojas de estilo o no soporta el lenguaje de hojas de estilo utilizado en **STYLE**, no deberá mostrar el contenido del elemento. Algunos lenguajes de hojas de estilo soportan recursos que permiten ocultar al navegador la información de estilo si éste no la soporta.



#### EJEMPLO 2.23

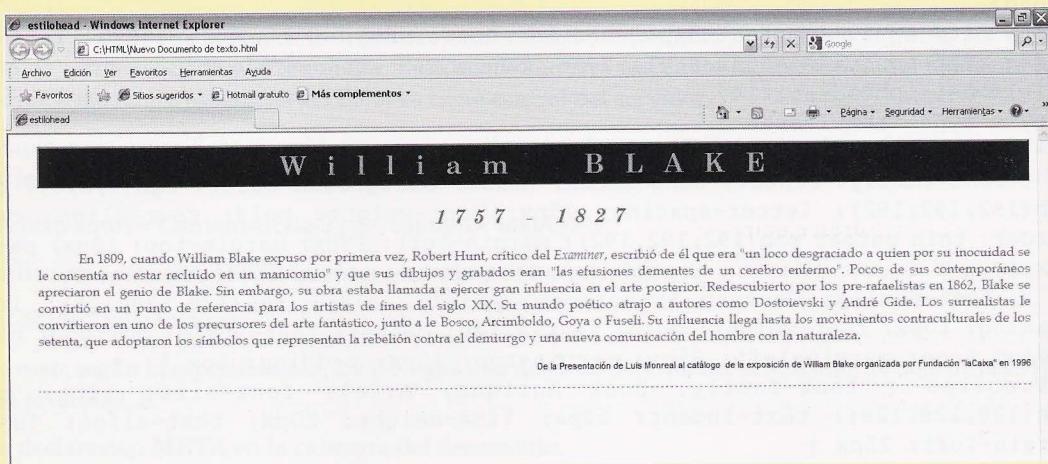
Considere el siguiente ejemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Content-Style-Type" content="text/css">
<title>estilohead</title>
<style type="text/css">
<!--
H1 { font-family: Century Schoolbook, Arial; background-color: rgb(0,0,0); color:
rgb(192,192,192); letter-spacing: 25px; font-weight: bold; text-align: center;
border: thin outset rgb(192,192,192); margin-left: 25px; margin-top: 10px; padding:
5px }
H2 { font-family: Century Schoolbook, Arial; color: rgb(128,128,128); letter-
spacing: 10px; font-weight: bold; text-align: center; font-style: oblique; border:
medium none; margin-left: 25px; margin-top: 10px; padding: 5px }
DIV.cuerpo { font-family: Book Antiqua, Arial; font-size: 12pt; color:
rgb(128,128,128); text-indent: 50px; line-height: 20px; text-align: justify;
margin-left: 25px }
SPAN.cursiva { font-style: oblique }
DIV_fuente { font-family: Arial; font-size: 8pt; text-align: right; border-bottom:
1px dashed rgb(128,128,128); margin-left: 25px }
-->
</style>
</head>
<body>
<h1>William BLAKE</h1>
<h2>1757 - 1827</h2>
<div class="cuerpo">
<p>En 1809, cuando William Blake expuso por primera vez, Robert Hunt, crítico del
Examiner, escribió de él que era "un loco desgraciado
a quien por su inocuidad se le consentía no estar recluido en un manicomio";
y que sus dibujos y grabados eran "las efusiones dementes de un cerebro
enfermo". Pocos de sus contemporáneos apreciaron el genio de Blake. Sin
embargo, su obra estaba llamada a ejercer gran influencia en el arte posterior.
Redescubierto por los pre-rafaelistas en 1862, Blake se convirtió en un punto de
referencia para los artistas de fines del siglo XIX. Su mundo poético atrajo a
autores como Dostoievski y André Gide. Los surrealistas le convirtieron en uno de
```

los precursores del arte fantástico, junto a le Bosco, Arcimboldo, Goya o Fuseli. Su influencia llega hasta los movimientos contraculturales de los setenta, que adoptaron los símbolos que representan la rebelión contra el demiurgo y una nueva comunicación del hombre con la naturaleza.</p>

```
</div><div class="fuente">
<p>De la Presentación de Luis Monreal al catálogo de la exposición de William Blake organizada por Fundación "laCaixa" en 1996 </p>
</div>
</body>
</html>
```

que el navegador mostrará así:



**Figura 2.27**

Observe como el contenido del elemento **STYLE** se ha delimitado como si se tratara de un comentario para ocultarlo a los navegadores que no soportan hojas de estilo. Las reglas de estilo siguen teniendo la forma *propiedad:valor* pero ahora se encierran entre corchetes '{ }' no entre comillas como sucedía con el atributo *style*. Todo ello, separado por un espacio, se adjunta a expresiones como *H1*, *H2*, *DIV*, *cuerpo*, *DIV.fuente* o *SPAN.cursiva* que en CSS son selectores, es decir, engarzan la información de estilo con el documento HTML seleccionando los elementos a que es aplicable. Es decir, la regla de estilo:

```
H1 { font-family: Century Schoolbook, Arial; background-color: rgb(0,0,0); color: rgb(192,192,192); letter-spacing: 25px; font-weight: bold; text-align: center; border: thin outset rgb(192,192,192); margin-left: 25px; margin-top: 10px; padding: 5px }
```

se ha seleccionado para ser aplicada a todas las ocurrencias del elemento **H1** en el cuerpo del documento. Lo mismo ocurre con la regla de estilo para *H2*, todas las instancias de este elemento en el cuerpo del documento se verán afectadas por esta regla de estilo definida en la cabecera. El alcance de la información de estilo utilizando el elemento **STYLE** es ahora mucho más amplia que cuando se modificaba la apariencia de una instancia determinada de un elemento mediante el atributo **style** como en el primer ejemplo. La información es ahora reutilizable en todo el cuerpo del documento.

En general, con CSS pueden declararse reglas de estilo en un elemento *STYLE* para:

- Todas las instancias de un elemento (es decir, todas las ocurrencias de elementos *P*, *H1*, etc.).
- Todas las instancias de un elemento HTML pertenecientes a una determinada clase (por ejemplo, cuando se utiliza el atributo *class* con algún valor).
- Instancias individuales de un elemento HTML (es decir, cuando se utiliza el atributo *id* con algún valor).

Las reglas:

```
DIV.cuerpo { font-family: Book Antiqua, Arial; font-size: 12pt; color: rgb(128,128,128);
text-indent: 50px; line-height: 20px; text-align: justify; margin-left: 25px }
Y
SPAN.cursiva { font-style: oblique }
```

son ejemplos del segundo caso. Observe como en el cuerpo del documento ambas se invocan mediante el uso del atributo *class* modificando los elementos *DIV* y *SPAN*:

```
<div class="cuerpo"> ... </div> en un caso y
Examiner en el otro
```

Más adelante se presentarán ejemplos del uso del atributo **id** como selector.

Observe que en la regla de la cabecera se ha definido a voluntad un estilo, el estilo *cuerpo* mediante una serie de pares *propiedad:valor*, además, se ha seleccionado el elemento al que afecta separándolo de su nombre mediante un punto (*DIV.cuerpo*). Para invocar ese estilo en el cuerpo del documento bastará calificar el elemento utilizando el atributo **class** con el nombre declarado en la regla encerrado entre comillas (*<DIV class="cuerpo">*). Es decir, esa ocurrencia del elemento *DIV* perteneciente a la clase *cuerpo* deberá recibir la información de estilo definida en la regla para *DIV.cuerpo*.

En el ejemplo se han utilizado los elementos *DIV* y *SPAN* para asignarlos a una clase y recibir la información de estilo definida para la misma, pero se podría haber utilizado cualquier otro elemento. **DIV** y **SPAN** tienen la ventaja de que no tienen por si mismos ningún significado que afecte a la presentación aparte de caracterizar a su contenido como de nivel de bloque o de nivel de línea respectivamente.

La combinación de hojas de estilo y el uso de los atributos *class* e *id* con los elementos HTML facilitan al autor una poderosa herramienta para ampliar las capacidades de HTML.

#### 2.4.5 INFORMACIÓN DE ESTILO EN HOJAS EXTERNAS

Insertar las reglas de estilo en la cabecera del documento las hace reutilizables para todo un documento. Pero el grado más alto de flexibilidad y eficacia se logra mediante las hojas de estilo externas, es decir, definiendo las reglas de estilo en un archivo externo al documento. De este modo puede cambiarse la apariencia de todo un conjunto de páginas con solo cambiar la hoja de estilo externa.



## EJEMPLO 2.24

Por ejemplo, se podría copiar la información de estilo utilizada en el ejemplo anterior y guardarla en un archivo de solo texto con la extensión .css al que se llamará *miestilo.css*:

```
<!--
H1 { font-family: Century Schoolbook, Arial; background-color: rgb(0,0,0); color: rgb(192,192,192); letter-spacing: 25px; font-weight: bold; text-align: center; border: thin outset rgb(192,192,192); margin-left: 25px; margin-top: 10px; padding: 5px }
H2 { font-family: Century Schoolbook, Arial; color: rgb(128,128,128); letter-spacing: 10px; font-weight: bold; text-align: center; font-style: oblique; border: medium none; margin-left: 25px; margin-top: 10px; padding: 5px }
DIV.cuerpo { font-family: Book Antiqua, Arial; font-size: 12pt; color: rgb(128,128,128); text-indent: 50px; line-height: 20px; text-align: justify; margin-left: 25px }
SPAN.cursiva { font-style: oblique }
DIV_fuente { font-family: Arial; font-size: 8pt; text-align: right; border-bottom: 1px dashed rgb(128,128,128); margin-left: 25px }
-->
```

Se pueden mantener las marcas de comentario pero se prescindirán de las etiquetas *<STYLE>* y *</STYLE>* que ya no son necesarias en la hoja de estilo externa.

La nueva versión del ejemplo 2.20, que incluye un enlace en su cabecera al fichero de hoja de estilo se reescribiría así:

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Content-Style-Type" content="text/css">
<link href="miestilo.css" rel="stylesheet" type="text/css">
<title>Estilo externo</title>
</head>

<body>

<h1>William BLAKE</h1>

<h2>1757 - 1827</h2>
<div class="cuerpo">

<p>En 1809, cuando William Blake expuso por primera vez, Robert Hunt, crítico del
Examiner, escribió de él que era "un loco desgraciado
a
quien por su inocuidad se le consentía no estar recluido en un manicomio"; y
que sus
dibujos y grabados eran "las efusiones dementes de un cerebro enfermo". Pocos de
```

sus contemporáneos apreciaron el genio de Blake. Sin embargo, su obra estaba llamada a ejercer gran influencia en el arte posterior. Redescubierto por los pre-rafaelistas en 1862, Blake se convirtió en un punto de referencia para los artistas de fines del siglo XIX. Su mundo poético atrajo a autores como Dostoievski y André Gide. Los surrealistas le convirtieron en uno de los precursores del arte fantástico, junto a le Bosco, Arcimboldo, Goya o Fuseli. Su influencia llega hasta los movimientos contraculturales de los setenta, que adoptaron los símbolos que representan la rebelión contra el demiurgo y una nueva comunicación del hombre con la naturaleza.

</div><div class="fuente">

<p>De la Presentación de Luis Monreal al catálogo de la exposición de William Blake organizada por Fundación "laCaixa" en 1996 </p>

</div>

</body>

</html>

Y se visualizaría en el navegador exactamente igual que antes.

Es la etiqueta `<link href="miestilo.css" rel="stylesheet" type="text/css">` la que invoca la información de estilo mediante un enlace (elemento LINK) en la cabecera del documento.

Mediante el valor `miestilo.css` del atributo **href** (un URI relativo en este caso, téngalo en cuenta si reproduce el ejemplo) se indicará dónde se ubicará la hoja de estilo externa. Mediante el valor `stylesheet` del atributo **rel** se definirá la relación existente entre el recurso invocado, la hoja de estilo, y el documento HTML. Mediante el valor `text/css` del atributo **type** se indicará el lenguaje de la hoja de estilo vinculado, en los ejemplos siempre CSS.

Además, el autor y el lector del documento no están obligados a utilizar una única hoja de estilo. HTML permite asociar cualquier número de hojas de estilo con un documento.

Así, el autor del documento puede especificar un determinado número de hojas de estilo mutuamente excluyentes denominadas hojas de estilo alternativas. Si el programa navegador le ofrece esa posibilidad, el lector podrá elegir de entre ellas su favorita, por ejemplo, en función de la resolución de su pantalla. Si se ofrecen **hojas de estilo alternativas** el autor deberá especificar de, entre ellas, la **hoja de estilo preferida**. A no ser que el usuario opte por otra, está sería la que el navegador debe utilizar. El autor también podrá especificar una **hoja de estilo persistente**, es decir, que el navegador deberá aplicar junto a una hoja de estilo alternativa. Incluso los autores podrán agrupar varias hojas de estilo alternativas (inclusive la declarada preferida por el autor), utilizando un solo nombre de estilo. Si el navegador permite al usuario seleccionar el nombre de un estilo, deberá también poder aplicar todas las hojas de estilo con ese nombre.

Además, si se diseñan hojas de estilo pensadas para diferentes medios, por ejemplo, para cuando el documento es presentado en monitor gráfico, para cuando el documento se muestra por televisión, para su impresión, para el uso

con dispositivos que ofrecen síntesis de voz, dispositivos táctiles basados en braille, etc.) HTML dispone del atributo **media** para que los navegadores carguen y apliquen la información de estilo de forma selectiva.

Las posibilidades que ofrece la separación entre el contenido del documento y las instrucciones acerca de cómo debe presentarse al usuario son enormes cuando se utilizan hojas de estilo.

Para especificar que una hoja de estilo es *persistente* se deberá utilizar el atributo **rel** con el valor *stylesheet* y no utilizar el atributo *title*:

```
<link href="miestilo.css" rel="stylesheet" type="text/css">
```

Para establecer que una hoja de estilo es *preferida*, se deberá utilizar *rel* con el valor *stylesheet* y nombrarla mediante el atributo *title*:

```
<link href="graficoa.css" title="Alta resolucion" rel="stylesheet" type="text/css">
```

Para especificar una hoja de estilo *alternativa* el atributo *rel* deberá establecerse a *alternate stylesheet* e identificarla mediante el atributo *title*.

```
<link href="graficob.css" title="Baja resolucion" rel="alternate stylesheet" type="text/css">
```

La hoja de estilo preferida también puede especificarse mediante el elemento **META** y mediante cabeceras HTTP. Por ejemplo, para establecer como preferida la hoja *Alta resolucion* con **META**, se introducirá la línea siguiente en la cabecera del documento:

```
<META http-equiv="Default-Style" content="Alta resolucion">
```

equivalente a la siguiente cabecera HTTP:

```
Default-Style: "Alta resolucion"
```

## ACTIVIDADES 2.31

- Cree una página web con hoja de estilo basándose en el ejemplo 2.23 y compruebe su funcionamiento.
- Cree una página web con hoja de estilo basándose en el ejemplo 2.24 y compruebe su funcionamiento.

## ACTIVIDADES GLOBALES

- Prepare una hoja de estilos para el documento HTML y para el documento XHTML definidos en las actividades globales anteriores.
- Compruebe que la hoja de estilos sea válida usando, por ejemplo, el validador de hojas de estilo del consorcio W3C (<http://jigsaw.w3.org/css-validator/>).



## RESUMEN DEL CAPÍTULO



En este capítulo se ha realizado una breve descripción del modelo de objetos del documento.

Se han introducido los lenguajes de marcas usados para representar información y específicamente los que se usan en entornos web.

Concretamente, se ha hecho una introducción a HTML y a los elementos y atributos básicos que se usan habitualmente.

Posteriormente, se ha mostrado el estándar XHTML así como sus diferencias con HTML y las versiones existentes a lo largo del tiempo.

Por último, se ha presentado una introducción a las hojas de estilos y su uso en conjunción con HTML y XHTML.



## EJERCICIOS PROPUESTOS



Dado el siguiente documento HTML base:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML
4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Documento HTML Lorem Ipsum</title>
</head>
<body>
<h1>Texto Lorem ipsum</h1>
<p>
```

  Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetur signiferumque eu per. In usu latine

equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

```
</p>
```

```
<p>
```

  Ius id vidit volumus mandamus, vide veritus democratum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vobis deserunt quaestio ei.

```
<a href="http://es.wikipedia.org/wiki/Lorem_
ipsum">Texto original
```

```
</p>
```

```
</body>
```

```
</html>
```

- 1. Incorpore una imagen con ancho de 200 píxeles.
- 2. Cree una nueva cabecera `<h1>`.
- 3. Incorpore un nuevo enlace a una página web.
- 4. Bajo esa cabecera cree dos párrafos.
- 5. Incorpore una lista ordenada.
- 6. Cree una hoja de estilos e incorpórela en el documento HTML. Dar los siguientes estilos:
  - Cambiar el tamaño de la fuente de los párrafos a 9 puntos.
- Añadir un borde a la imagen de 5 px de color blanco (`border: 2px solid #fff`).
- Asignar un “id” a un párrafo y modificar únicamente el estilo de éste cambiando el color de la fuente a rojo (`color: red`).
- Crear una clase nueva y asignar dos párrafos a esa clase y cambiar el estilo del mismo modificándolo a negrita (`font-weight: bold`).
- 7. Valide todos los cambios usando un validador online como el de W3C.



## TEST DE CONOCIMIENTOS



**1** ¿Qué etiqueta sirve en HTML para definir un párrafo?

- a) `<img>`.
- b) `<br>`.
- c) `<p>`.
- d) `<b>`.

**2** En HTML, ¿qué atributo de una imagen se usa para indicar la URL de la imagen?

- a) Img.
- b) url.
- c) Source.
- d) Src.

**3** ¿Qué significa XHTML?

- a) eXtra Huge Templates Markup Language.
- b) eXtended Huge Templates Markup Language.
- c) eXtra HyperText Markup Language.
- d) eXtended HyperText Markup Language.

**4** En XHTML, ¿cuál es el nombre del elemento raíz?

- a) `<head>`.
- b) `<html>`.
- c) `<xhtml>`.
- d) No hay un único elemento raíz.

**5** ¿Cuál de las siguientes etiquetas es válidas en XHTML?

- a) `<head></Head>`.
- b) `<br/>`.
- c) `<imgsrc=imagen.jpg>`.
- d) `<p><br></p></br>`.
- e) No hay un único elemento.

**6** En CSS, ¿para qué se usa el selector que comienza con `#`?

- a) Seleccionar varios elementos de una clase.
- b) Seleccionar un único elemento.
- c) Seleccionar todos los elementos de un tipo.
- d) Un selector nunca debe comenzar con `#`.

# 3

# Lenguajes para el almacenamiento y transmisión de información

## OBJETIVOS DEL CAPÍTULO

- ✓ Conocer los tipos de lenguajes para el almacenamiento y transmisión de información.
- ✓ Aprender la sintaxis básica y los posibles elementos de XML.
- ✓ Diferenciar entre documentos bien formados y documentos válidos.
- ✓ Conocer qué son y para qué se usan los espacios de nombres en XML.

## 3.1 TIPOS DE LENGUAJES

Dentro de los lenguajes para el almacenamiento y transmisión de la información se pueden encontrar los tipos siguientes:

- **De marcas. XML** (*eXtended Markup Language*). Es un metalenguaje extensible de etiquetas desarrollado por el W3C que permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

```
view plain copy to clipboard print ?
01. <?xml version="1.0" encoding="UTF-8" ?>
02. <poblaciones>
03. <poblacion id="0">Alcobendas</poblacion>
04. <poblacion id="1">Miraflores de la Sierra</poblacion>
05. <poblacion id="2">San Fernando de Henares</poblacion>
06. </poblaciones>
```

Figura 3.1. Documento XML



XML es una simplificación y adaptación del SGML.

- **De listas. JSON** (*JavaScript Object Notation*). Es un formato ligero para el intercambio de datos.

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX.



**AJAX** (*Asynchronous JavaScript And XML*, JavaScript Asíncrono y XML) es una técnica de desarrollo web para crear aplicaciones interactivas o **RIA** (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma, es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad de las aplicaciones.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador semántico de JSON que de XML. En JavaScript, un texto JSON se puede analizar fácilmente usando el procedimiento `eval()`, lo que ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

Si bien es frecuente ver JSON posicionado contra XML, también es frecuente el uso de JSON y XML en la misma aplicación. Por ejemplo, una aplicación de cliente que integra datos de *Google Maps* con datos meteorológicos en SOAP hacen necesario soportar ambos formatos.

El término JSON está altamente difundido en los medios de programación, sin embargo, es un término mal descrito ya que en realidad es solo una parte de la definición del estándar **ECMA-262** en que está basado JavaScript. De ahí que ni Yahoo, ni Google emplean JSON, sino **LJS** (*Literal Javascript*). Una de las cualidades intrínsecas de LJS facilita el flujo de datos e, incluso, de funciones. Todo lo referente a transferencia de datos en todos sus tipos no requiere de la función `eval()`, y es precisamente en eso en donde supera por mucho JavaScript al XML, si se utiliza el LJS y no la incorrecta definición de JSON.

```

view plain copy to clipboard print ?
01. {"poblaciones": [
02. {"poblacion": { "@id": "0", "#text": "Alcobendas" }},
03. {"poblacion": { "@id": "1", "#text": "Miraflores de la Sierra" }},
04. {"poblacion": { "@id": "2", "#text": "San Fernando de Henares" }}
05.
06.
07.]
08. }

```

*Figura 3.2. Equivalencia del documento XML de la figura 3.1 en JSON*

## ACTIVIDADES 3.1



- Vaya a la página <http://www.json.org> y obtenga más información sobre JSON.
- Busque información en la web sobre JSON.
- Busque información en la web sobre AJAX.

## 3.2 DEFINICIÓN DE XML

XML se puede ver como un subconjunto de SGML mucho más simple, hasta el punto de que la especificación de XML es de una décima parte que SGML. Por otro lado, el lenguaje XML se puede definir como un metalenguaje, esto es, XML puede ser usado para definir otros lenguajes, al igual que el SGML. Por ejemplo, el lenguaje XHTML, ampliamente usado hoy en día en la *web*, se ha construido usando como base XML.

A continuación, se detallará como se construyen los documentos XML y las características principales de los mismos.

## 3.3 ESTRUCTURA Y SINTAXIS DE XML

Un documento XML está formado, en principio, por lo que se conoce como “texto plano”, esto es, texto en el cual todos los caracteres se representan visualmente, sin existir caracteres no visibles exceptuando los de salto de línea, tabulador o espacio.

Tal y como ya sabe el lector los documentos escritos usando XML contendrán marcas para separar la información que estructura el documento de la información que se quiere almacenar. Para construir dichas marcas, en XML se usan los caracteres “<” y “>” para delimitar el texto que se desea marcar, mientras que el carácter “/” sirve para indicar la etiqueta de finalización del marcado. Un posible ejemplo de documento XML sería el siguiente:

<nombre>Luis</nombre>

Esta construcción se denomina habitualmente “elemento” y constituye la base principal de los documentos XML. Además de los elementos, un documento XML puede contener otros tipos de información. A continuación, se especificarán los más relevantes.

### 3.3.1 ETIQUETAS, ELEMENTOS Y ATRIBUTOS

Las etiquetas son el componente de XML que permite definir los elementos que conformarán un documento de la siguiente forma:

```
<etiqueta>Valor</etiqueta>
```

Como se puede observar los elementos se formarán usando una etiqueta de inicio, otra de fin, delimitadas mediante los caracteres ">" y "<", y que comparten el identificador textual pero añadiendo el carácter "/" al principio. En medio de las etiquetas de inicio y fin del elemento se representará el contenido que se desee almacenar en ese elemento. Este contenido puede a su vez englobar otros elementos. Por otro lado, los elementos pueden no tener ningún valor, pero en ese caso se deberá usar solamente la etiqueta de finalización.

Se considera que en el elemento XML engloba todo lo que se encuentra entre las correspondientes etiquetas de inicio y de fin y pueden contener tanto otros elementos como simplemente texto o una combinación de ambos.

A continuación, se muestra un ejemplo de XML para representar la ficha de un alumno sería sintácticamente correcto:



#### EJEMPLO 3.1

```
<alumno>
 <nombre>Pablo</nombre>
 <apellido>Pérez</apellido>
 <telefono>9155555</telefono>
 <direccion></direccion>
 </varon>
</alumno>
```

Los nombres de los elementos deberán empezar por una letra o bien por el carácter "\_" o ":" siempre y cuando el principio no contenga la palabra "xml" en cualquier combinación posible de mayúsculas y minúsculas. Además, los nombres son *case sensitive* y solo podrán contener letras, números y los caracteres "\_", ".", "-", ":" Teniendo en cuenta estas reglas, los siguientes ejemplos serían incorrectos:



#### ¿SABÍAS QUE...?

*Case sensitive* (del inglés, literalmente *sensible a las mayúsculas/minúsculas*) es una expresión usada en jerga informática que se aplica a los textos en los que tiene alguna relevancia escribir un carácter en mayúsculas o minúsculas.

```
<nombre>Pablo</finNombre>
</apellido>Pérez<apellido>
<varon>
<xMl_direccion></xMl_direccion>
<!notas></!notas>
```

Los elementos pueden asimismo contener atributos, los cuales se especificarán en la etiqueta de inicio del elemento. El objetivo de los atributos es poder proporcionar una información adicional sobre un elemento concreto. La sintaxis para representar los atributos consiste en especificar el nombre del atributo dentro de la etiqueta de inicio, a continuación un símbolo “=” y finalmente el valor del atributo delimitado por comillas dobles o por comillas simples:

```
<elem1 atrib1="val1" atrib2="val2"> Valor </elemento>
<elem2 atrib1="val3" atrib3="val3"> Valor </elemento>
```

Siguiendo con el ejemplo presentado previamente, el sexo del alumno anteriormente se había representado usando un elemento mientras que en el siguiente ejemplo se utiliza un atributo para denotar dicha característica. Además se ha añadido el atributo “fechaNacimiento” para el elemento alumno y el atributo “tipo” para el elemento teléfono.

### EJEMPLO 3.2

```
<alumno sexo="varon" fechaNacimiento="5/6/1990">
 <nombre>Pablo</nombre>
 <apellido>Pérez</apellido>
 <telefono tipo="movil">9155555</telefono>
 <direccion>Ronda de Segovia 111</direccion>
</alumno>
```

Se puede observar que generalmente se pueden usar tanto atributos como nuevos elementos para representar información. Sin embargo, el uso de un número excesivo de atributos puede provocar que el documento XML sea menos legible, más difícil de mantener y difícilmente extensible. Además hay que tener en cuenta que los atributos no pueden contener información en forma de árbol, esto es, no pueden contener otros elementos o atributos tal y como sucede con los elementos. De forma general, se puede establecer la recomendación no usar atributos en exceso y dejarlos casi exclusivamente para representación de metadatos.

Siguiendo esta recomendación, en el ejemplo tanto el atributo “sexo” como el atributo “fechaNacimiento” se pueden convertir en elementos. Además en este ejemplo se ha añadido un identificador del alumno como atributo:

### EJEMPLO 3.3

```
<alumno id="532">
 <nombre>Pablo</nombre>
 <apellido>Pérez</apellido>
 <fechaNacimiento>
 <dia>5</dia>
 <mes>6</mes>
 <año>1990</año>
 </fechaNacimiento>
 <sexo>varón</sexo>
 <telefono tipo="movil">9155555</telefono>
 <direccion>Ronda de Segovia 111</direccion>
</alumno>
```

## ACTIVIDADES 3.2



► En el ejemplo siguiente:

```
<persona sexo="hombre" id="ricky">
<nombre>Ricky Martin</nombre>
<email>ricky@puerto-rico.com</email>
<relacion amigo-de="leatitia">
</persona>
```

Indique los elementos y los atributos que haya.

### 3.3.2 CARÁCTERES ESPECIALES

En XML algunos símbolos son reservados del lenguaje por lo que para poder representarlos es necesario usar unos códigos. Estos se definen usando el símbolo & seguido de una palabra clave y terminados por punto y coma. Estas construcciones son entidades predefinidas (las entidades se verán más adelante). A continuación, se detallan algunos de los más relevantes:

Código	Carácter
&quot;	"
&amp;	&
&apos;	'
&lt;	<
&gt;	>

### 3.3.3 INSTRUCCIONES DE PROCESAMIENTO

Más allá de los propios datos contenidos en los ficheros XML y las etiquetas de marcado en un fichero XML se pueden encontrar instrucciones especiales llamadas instrucciones de procesamiento. Éstas comienzan con <? Y terminan con ?>. Una de las instrucciones de procesamiento más habituales es la que se usa para indicar qué versión de XML se va a usar y cuál es la codificación de caracteres que se va a usar. Por ejemplo, si se usa XML 1.0 y UTF-8 la instrucción de procesamiento sería la siguiente:

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

### 3.3.4 COMENTARIOS Y SECCIONES CDATA

Dentro de un documento XML se puede añadir información que no pertenezca ni al marcado ni la información contenida documento y que sirve para documentarlo en forma de comentarios internos. La sintaxis de un comentario consta de un texto delimitado por una marca inicial “!--” y una marca final “-->”

```
<!-- Comentario valido en XML -->
```

Los comentarios son elementos especiales y no necesita ninguna marca de cierre. Además hay que tener en cuenta que dentro de un comentario no se pueden usar dos guiones seguidos “--”.

Además en XML se encuentran disponibles las secciones CDATA, que permiten marcar un texto para que éste no sea procesado por el *parser*, es decir, no serán analizadas sintácticamente. CDATA proviene de “Character DATA” (Datos de carácter) en contraposición a datos de marcado. La sintaxis de estas secciones se basa en la etiqueta de inicio “<![CDATA[“ y la etiqueta de fin “]]>”.

En el siguiente ejemplo las dos definiciones del elemento 4 serían interpretadas de la misma forma:



### EJEMPLO 3.4

```
<alumnos>
<![CDATA[
 <alumno id="321">
 <nombre>Luis</nombre>
 </alumno>
]]
</alumnos>

<alumnos>
 <alumno id="321">
 <nombre>Luis</nombre>
 <alumno>
</alumnos>
```



### ACTIVIDADES 3.3



- ▶ Visite la web oficial de XML: <http://www.w3.org/XML/>.
- ▶ Busque por Internet documentos XML que contengan secciones CDATA.

## 3.4 DOCUMENTOS XML BIEN FORMADOS

Una vez visto los elementos que pueden formar parte de un documento XML y sus características el siguiente paso será establecer cuando un documento es correcto. En este sentido, en XML se puede hablar de documentos “bien formados” y documentos “válidos”.

Los documentos bien formados son aquellos que son sintácticamente correctos, es decir, que cumplen las reglas expuestas en los apartados previos. Sin embargo, los documentos válidos son aquellos que, además de estar bien formados, cumplen los requisitos de una definición de estructura.



Las definiciones de estructura se presentarán en el Capítulo 4.

Además de las reglas expuestas en los apartados anteriores, se pueden destacar algunos otros aspectos en los que no se ha incidido de forma directa:

- Un documento XML debe contener un único elemento raíz.
- Los elementos son *case-sensitive*, por lo que las etiquetas de inicio y fin de un elemento deben concordar en mayúsculas y minúsculas.
- El documento solo contendrá caracteres válidos dependiendo del tipo de codificación del documento.
- Los caracteres “<”, “>” y “&” solo deben aparecer para delimitar las etiquetas de los elementos y para usar caracteres especiales.

## 3.5 ESPACIOS DE NOMBRES

De forma general, los documentos XML se suelen combinar con otros documentos XML existentes, esto es, es habitual que se desee usar uno o varios módulos desarrollados previamente por terceras personas. Esta modularidad es una característica esencial de XML y permite al desarrollador poder reutilizar código existente que en muchos casos ya ha sido depurado y ampliamente probado.

El problema que surge en estos casos es la posible colisión que se puede producir en los nombres de los elementos que conformen los módulos que se quieren usar. Para solucionar este problema XML proporciona un mecanismo denominado “espacio de nombres”, que permite asignar nombres extendidos a los elementos de forma que se puedan evitar las colisiones.

### 3.5.1 DECLARACIÓN DE ESPACIOS DE NOMBRES

Un espacio de nombres se define como una referencia URI (*Uniform Resource Identifier*), que servirá para identificar los elementos que pertenecen a dicho espacio de nombres. Otra forma de verlo es que los elementos tendrán un nombre compuesto por dos partes: una primera con su nombre y una segunda con el nombre de espacio de nombres. Este nombre compuesto permitirá identificar de forma única al elemento en cuestión y de esta forma conocer siempre a qué elemento se está refiriendo el documento.

La construcción de estos nombres extendidos se realiza uniendo el nombre el espacio de nombres y el nombre del elemento o atributo usando como conector el símbolo “:”. Sin embargo, las referencias URI pueden ser largas, lo

que va en detrimento de la legibilidad y claridad del documento, además de propiciar que se cometan errores más fácilmente. Además las URIs pueden contener caracteres no válidos en XML. Para solucionar este problema, en XML se puede asignar un sinónimo corto al espacio de nombres de forma que este sinónimo corto sea el que se use a lo largo del documento. El sinónimo se asigna usando el separador ":" y la etiqueta "xmlns". En realidad, "xmlns" es un atributo reservado (recuerde que los atributos no pueden comenzar por "xml" en ninguna combinación de mayúsculas y minúsculas). El ejemplo siguiente representa el uso de un espacio de nombres:



### EJEMPLO 3.5

```
<elementoej xmlns:enej="http://dominioej.com/rutaej">
 <enej:elemento1>Texto 1</nsej:elemento1>
 <enej:elemento2>Texto 2</nsej:elemento2>
</elementoej>
```

Se puede observar que el sinónimo corto del espacio de nombres de ejemplo es "enej" y que su uso resulta más adecuado que el nombre completo del espacio de nombres "http://dominioej.com/rutaej". Además se puede observar que los elementos "elemento1" y "elemento2" pertenecen al espacio de nombres "enej" al estar calificados con el sinónimo de dicho espacio.

## ACTIVIDADES 3.4



- Busque un documento XML por Internet que disponga de un espacio de nombres.

### 3.5.2 ESPACIOS DE NOMBRES POR DEFECTO

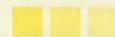
Si un espacio de nombres se declara sin su sinónimo correspondiente esto indicará que todos los elementos (incluido el elemento que declara el espacio de nombres) que contenga pertenecerán a dicho espacio de nombres. Esto será así siempre que los elementos no tengan el prefijo de otro espacio de nombres. Por tanto, sería como definir un espacio de nombres por defecto para los elementos que no tengan espacio de nombres asignado.

Otro uso de los espacios de nombres que puede resultar de gran utilidad es dejar su declaración en blanco ("xmlns=""") , lo que indicaría que los elementos y atributos contenidos, por defecto, no pertenecen a ningún espacio de nombres.

Por último, hacer énfasis en que cuando se declara un espacio de nombres por defecto (o sin espacio de nombres por defecto, dejando el atributo *xmlns* vacío) pero un elemento contiene un prefijo de un espacio de nombres, el espacio de nombres que prevalecerá será éste último.



## RESUMEN DEL CAPÍTULO



En este capítulo se ha presentado una introducción a los lenguajes de marcas así como la motivación de su origen y la evolución que han seguido desde su nacimiento.

Por otro lado, se ha introducido la sintaxis básica de XML y se han definido los elementos básicos que pueden aparecer en un documento XML.

A continuación, se ha expuesto qué es un documento XML bien formado y cuándo es válido, dejando este último punto para su profundización en el Capítulo 4.

Por último, se han introducido el concepto de espacios de nombres y su utilidad dentro de un sistema que maneja múltiples documentos XML.



## EJERCICIOS PROPUESTOS



- 1. Definir un documento XML que permita representar un libro. Deberá contener los atributos típicos como “título”, “autores”, “editorial”, “fecha de publicación”, “isbn”, etc.
- 2. A partir de la definición anterior escribir un documento XML que al menos contenga 10 entradas de libros.
- 3. Buscar un validador de XML *online* (por ejemplo el de W3C <http://validator.w3.org/>), introducir el documento generado en el ejercicio anterior y comprobar que el documento esté bien formado.
- 4. Crear un espacio de nombres ficticio e introducirlo en el XML del ejercicio 2 y comprobar que el documento XML sigue estando bien formado.



## TEST DE CONOCIMIENTOS



**1** ¿Qué relación tienen el SGML y el XML?

- a) El SGML es un derivado del XML.
- b) El XML es un derivado del SGML.
- c) Son el mismo lenguaje.
- d) Ninguna de las anteriores es correcta.

**2** ¿Cuál de las siguientes líneas de XML es correcta?

- a) <nombre>Pablo</nombre>.
- b) <persona nombre="Pablo">.
- c) <persona nombre="Pablo"/>.
- d) <Nombre="Pablo"></nombre>.

**3** Una sección CDATA sirve para:

- a) Poner datos adicionales sobre el tipo de fichero XML.
- b) Sirve para añadir metadatos adicionales en la cabecera XML.
- c) Escribir un contenido que no va a ser interpretado como XML sino como solo texto.
- d) No es un campo válido en XML.

**4** ¿Cuál de las siguientes afirmaciones es correcta?

- a) Un fichero XML puede tener uno o más nodos raíz.
- b) Un fichero XML puede tener un único nodo raíz.
- c) Los elementos en XML no distinguen entre mayúsculas y minúsculas.
- d) Los caracteres <, > y & se pueden usar sin problema en el contenido de los elementos de un fichero XML.

**5** ¿Cuál es la función principal de un espacio de nombres?

- a) Poder distinguir elementos con el mismo nombre que provienen de distinta fuente.
- b) Asegurar que un documento está bien formado.
- c) Asegurar que un documento es válido.
- d) No tiene una función específica.

# 4

# Definición de esquemas y vocabularios en XML

## OBJETIVOS DEL CAPÍTULO

- ✓ Describir la estructura de un documento XML con DTD.
- ✓ Conocer los elementos de los que se compone una DTD.
- ✓ Definición de entidades en una DTD.
- ✓ Creación y asignación de atributos a un elemento desde una DTD.
- ✓ Asignación de una DTD a un documento XML.
- ✓ Describir la estructura de un documento XML con un esquema.
- ✓ Conocer los elementos de los que se compone un esquema.
- ✓ Creación y asignación de atributos a un elemento desde un esquema.
- ✓ Asignación de un esquema a un documento XML.
- ✓ Tipos básicos en los elementos de un esquema.
- ✓ Asignación de elementos hijos con modificación de ocurrencias.
- ✓ Distinguir entre documento bien formado y documento válido.
- ✓ Conocer que herramientas de validación existen vía web o aplicación local.

XML ha sido propuesto como un estándar en el intercambio de información, independientemente de la plataforma en la que se genere o se utilice. Esta premisa, en principio, es válida mientras que los documentos XML no cambien entre los intercambios. Si en los intercambios se permite la modificación, adición o supresión de elementos de información, se ha de tener especial cuidado de no modificar la estructura del documento. No siempre es posible y puede llegar el caso a situaciones en las que el mismo documento pueda leerse en una aplicación y en otra no.

No solo es imprescindible que el documento esté bien formado (con las etiquetas de apertura y cierre bien ubicadas, que la codificación sea correcta, que los elementos cumplan la sintaxis de XML, etc.), sino que ambos actores (emisor y receptor) se ciñan a un formato de fichero definido previamente. Para evitar estos casos, se ha de definir una estructura fija del documento que conozcan las partes que intercambian la información.

A continuación, se mostrarán dos maneras de especificar formatos de comunicación en XML, de los que destacan las DTD y los XML Schema.

## 4.1 DTD

**DTD** (*Document Type Definition*) es la definición de cómo se construye un documento XML para que se ajuste a las necesidades previamente analizadas. Es decir, establece qué elementos son aceptados y en qué posiciones deben estar dentro de un documento XML.

Cuando se define una DTD y se referencia dentro de un documento XML, se establece una relación de:

- ✓ Qué léxico es el que se espera.
- ✓ Qué reglas sintácticas debe cumplir nuestro lenguaje.

El significado semántico se proporciona en el uso del DTD y del documento XML al generar las aplicaciones que lo utilizan.

Por tanto, antes de crear un documento XML se deberá analizar qué elementos y etiquetas se van a necesitar para la aplicación que se quiere crear. Para esto sirve una DTD.

¿Por qué resulta importante la creación de DTD? Principalmente, cuando se analiza un almacenamiento de información en XML es porque esa información es importante compartirla. Pero aunque la compartición de la información pueda resultar interesante, que la información esté sujeta a una serie de reglas para que todos los que lo utilicen (en consultas, inserciones, modificaciones, borrados, etc.); se asegure que esté bien formado y resulte válido para el uso por terceras aplicaciones. Si en el proceso de acceso y modificación del documento XML se incumple su estructura, nos encontraremos con un documento no válido y con un futuro problema en su futura utilización. Por tanto una DTD permitirá asegurar que la información que contiene es válida y cumple los requisitos de intercambio de información entre terceras (personas con la misma DTD).

La creación de una DTD resulta muy simple y se verá más adelante pero resulta interesante como ubicar estas declaraciones:

1. La DTD se puede ubicar dentro del propio documento XML.
2. La DTD se ubica fuera del documento XML (en un fichero externo).

Ambas realizan la misma función pero resulta más cómoda utilizar un fichero externo para almacenar la DTD y realizar un enlace en el documento XML para que la use. Además de la comodidad, existe otra ventaja fundamental:

no se almacena en cada documento XML la DTD. Si tuviéramos muchos ficheros XML que se basan en la misma especificación DTD, se tendría que cambiar en todos los fichero la declaración de la DTD, cosa que si estuviera enlazado en un fichero externo, se cambiaría una única vez y todos los documentos XML lo aprovecharían. Además se estaría almacenando la misma información en todos los documentos, lo que aumentaría el tamaño del documento XML.

Para continuar, establezcamos un ejemplo en el que utilizar una DTD. Se quiere almacenar los mensajes de móviles que se envían a un servidor. Los datos que se guardarán son los siguientes:

- ✓ Número de teléfono del usuario.
- ✓ Fecha de envío.
- ✓ Hora de envío.
- ✓ Contenido del mensaje.

Un ejemplo de documento XML que guarde estos mensajes SMS podría ser el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE BDsms SYSTEM "BDsms.dtd">
<BDsms>
 <sms>
 <telefono>955 55 66 55</telefono>
 <fecha>1/7/2011</fecha>
 <hora>23:55</hora>
 <mensaje>Juego1: Tetris</mensaje>
 </sms>
 <sms>
 <telefono>745 15 56 11</telefono>
 <fecha>22/9/2011</fecha>
 <hora>15:05</hora>
 <mensaje>Juego2: Arkanoid</mensaje>
 </sms>
 <sms>
 <telefono>842 35 22 00</telefono>
 <fecha>10/11/2011</fecha>
 <hora>09:22</hora>
 <mensaje>Juego3: Comecocos</mensaje>
 </sms>
</BDsms>
```

La DTD que permite establecer el formato de intercambio de estos mensajes SMS en el documento XML, podría ser la siguiente (fichero "BDsms.dtd"):

```
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (telefono, fecha, hora, mensaje)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```

Notad que el fichero *BDsms.dtd* está referenciado en el documento XML. Si se quisiera incluir todo en el mismo documento XML, entonces quedaría:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE BDsms
[
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (telefono, fecha, hora, mensaje)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
]>

<BDsms>
<sms>
<telefono>955 55 66 55</telefono>
<fecha>1/7/2011</fecha>
<hora>23:55</hora>
<mensaje>Juego1: Tetris</mensaje>
</sms>
<sms>
<telefono>745 15 56 11</telefono>
<fecha>22/9/2011</fecha>
<hora>15:05</hora>
<mensaje>Juego2: Arkanoid</mensaje>
</sms>
<sms>
<telefono>842 35 22 00</telefono>
<fecha>10/11/2011</fecha>
<hora>09:22</hora>
<mensaje>Juego3: Comecocos</mensaje>
</sms>
</BDsms>

```

#### 4.1.1 BLOQUES PARA CONSTRUIR UNA DTD

Viendo el ejemplo de la base de datos de SMS, se puede observar una serie de elementos definidos en una DTD:

- **Elemento:** Es el bloque principal con el que se construye los documentos XML.
- **Atributo:** Es una manera de añadir más información a un elemento.
- **Entidad:** En XML existen algunos caracteres que tienen un significado especial. La aparición de estos caracteres como datos almacenables hace que se puedan confundir con entidades propias de un documento XML. Son entidades las siguientes:

- &nbs; => ‘ ‘ // Espacio en blanco
- &gt; => >
- &lt; => <
- &quot; => “
- &apos; => ‘
- &amp; => &

Se pueden usar dentro del propio documento XML y tras parsear el documento, tomarán el valor indicado a la derecha. También el usuario puede definir entidades propias para que, después de analizar el documento XML, se sustituyan por el valor indicado. Es como una especie de definición preestablecida. Un par de ejemplos son los siguientes:

```
<!ENTITY pi "3.141592"> => π
<!ENTITY textField SYSTEM "fichero.txt"> => &textFile
<!ENTITY miURL SYSTEM "http://www.as.com"> => &miURL
```

- **PCData:** su significado en inglés es *Parsed Character Data*. Indica que entre la etiqueta de apertura y cierre de ese elemento, se almacenarán caracteres como texto y serán analizados por un *parser*. Como es lógico, al analizarse mediante el *parser* el contenido del texto para encontrar entidades y elementos, si es solo texto no podrá encontrarse ninguno de los caracteres anteriormente mencionados (pues podría confundir al *parser* a la hora de analizar el documento XML).
- **CDATA:** su significado en inglés es *Character Data*. A efectos es prácticamente igual que PCData pero el contenido de ese elemento, entre sus etiquetas de apertura y cierre, no se analizará por el *parser* de análisis de entidades y elementos XML.

Los elementos se declaran de una de las dos siguientes maneras:

```
<!ELEMENT nombre_del_elemento categoria>
o
<!ELEMENT nombre_del_elemento (nodos_hijos)
```

En el segundo caso se pueden definir el *nombre\_del\_elemento* como el nodo padre del que cuelgan un conjunto de *nodos\_hijos* (separados por comas).

Si existiera algún elemento que deba estar vacío, existe la palabra reservada **EMPTY** que puesta en la zona de “categoría” permite indicar este hecho. Un ejemplo de uso es la etiqueta *<br>* de HTML. Esta etiqueta permite establecer un salto de línea pero debería estar indicado como *<br></br>*. Si se declara como **EMPTY**, entonces se podría simplificar el salto de línea de la siguiente manera: *<br/>*. Resulta a todos modos más cómodo.

```
<!ELEMENT saltoLinea EMPTY> => <saltoLinea/>
```

Dentro de los elementos se puede indicar que si no están compuestos por nodos hijos, o son vacíos (EMPTY), o contienen información (**PCDATA**, **CDATA**, **ANY**).

```
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT hora (#CDATA)>
```

En el caso concreto de ANY, permite indicar que cualquier combinación de elementos conocidos serían válidos en ese caso.

```
<!ELEMENT hora (#ANY)> => Podría valer teléfono, fecha, mensaje, ...
```

No se ha comentado nada, hasta el momento, sobre los atributos. La manera de declararlos en un DTD es de esta forma (sintaxis):

```
<!ATTLIST nombre_elemento nombre_atributo
 tipo_atributo valor_por_defecto>
```

- **nombre\_elemento:** es el elemento al que se le quiere añadir el atributo.

- **nombre\_atributo:** es el nombre del atributo que se quiere añadir.
- **tipo\_atributo:** existen muchos tipos de atributos, destacan:

- **CDATA:** es un texto. Podrá tener cualquier carácter.
- **ID:** es un identificador que permite identificar al elemento de manera única en todo un documento XML.
- **IDREF:** es un identificador de otro elemento del propio documento XML.
- **IDREFS:** es una lista de identificadores a otros elementos.
- **(tipo1 | tipo2 | tipo3 | ...):** el valor es uno de los indicados en esta lista enumerada (definida por el usuario).
- **NMTOKEN:** es un texto que solo podrá tener letras, dígitos, guión “-”, subrayado “\_”, punto “.” y dos puntos “:”. Es lo que se llama nombres válidos XML.
- **NMTOKENS:** es una lista de nombres XML válidos. Es como un *NMTOKEN* pero se incluyen los espacios en blanco “ “, tabuladores o retornos de carro.
- **ENTITY:** el tipo del atributo es una entidad que se ha declarado anteriormente.
- **ENTITIES:** Es una lista de entidades.

Los atributos podrán ser declarados como obligatorios **#REQUIRED**, optativos **#IMPLIED** o fijos **#FIXED**. Este último obliga al usuario que el atributo sea siempre el mismo, sin que pueda cambiarlo (si se cambia no estaría conforme a la DTD declarada y devolvería un error).

Como no se ha declarado ningún atributo en el ejemplo de los SMS, cambiaremos el elemento *hora* para añadir la zona o franja horaria (de manera obligatoria). Se quiere tener un documento XML de este tipo:

...  
 <hora zona="GMT+1">09:22</hora>  
 ...

por lo que el DTD debe añadir la siguiente línea:

<!ATTLIST hora zona CDATA “GMT+1” #REQUIRED>

#### 4.1.2 SECUENCIAS DE ELEMENTOS: ESTRUCTURA CON HIJOS

Podemos ver que el elemento *sms* está compuesto por *telefono*, *fecha*, *hora* y *mensaje*. Esta es una relación padre-hijos. Imaginemos que una vez definida una DTD, nos damos cuenta que, por cada SMS recibido se pueden almacenar varios mensajes. En un primer momento se puede pensar que para salir del paso la solución sería ésta:

```
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (telefono, fecha, hora, mensaje, mensaje2)>
<!ELEMENT telefono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
<!ELEMENT mensaje2 (#PCDATA)>
```

¿Por qué no es conveniente? La razón fundamental es que estamos haciendo cambios en la DTD que obligarán a los *sms* que quizás no tengan “mensaje2”. Además estamos particularizando un caso concreto en vez de fomentar la generalidad de la definición del lenguaje DTD. ¿Cómo resolverlo?

En este caso debería cambiar la ocurrencia de la aparición de los elementos, justo cuando los indicamos en el elemento padre. Las ocurrencias que pueden aparecer se indican con los siguientes operadores:

- “”: indica que aparece obligatoriamente una vez. Es el caso inicial en cuanto se declara un hijo al elemento *sms*.
- ‘+’: indica que puede haber una o más ocurrencias del elemento indicado.  
`<!ELEMENT sms (teléfono, fecha, hora, mensaje+)>`
- ‘\*’: indica que puede haber cero o más ocurrencias del elemento indicado.  
`<!ELEMENT sms (teléfono, fecha, hora, mensaje*)>`
- ‘?’ indica que puede haber cero o una ocurrencia del elemento indicado (también llamado opcionalidad).  
`<!ELEMENT sms (teléfono, fecha, hora, mensaje)>`

No solo se pueden cambiar las ocurrencias de los hijos declarados, sino que podemos indicar la opcionalidad de aparición de hijos. En este caso podríamos definir que queremos almacenar o la hora o el mensaje pero no las dos simultáneamente pero sí obligatoriamente una de las dos, por lo que quedaría así:

```
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (telefono, fecha, (hora | mensaje)>
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```



### ¿SABÍAS QUE...?

IBM inventó, allá por los años 60, un lenguaje llamado *GML* (*General Markup Language*) que permitía definir la estructura de un documento formateado y utilizarlo para almacenar toda la información que generaba la empresa. La Organización Internacional de Estandarización (*ISO*) vio las posibilidades de *GML* y se puso a trabajar en una versión estandarizada del mismo. En 1986 lanzó *SGML* (*Standard Generalized Markup Language*). A partir de *SGML* nació en 1989 *HTML* (*HyperText Markup Language*) y poco después *XML* (*eXtensible Markup Language*) en 1998.

## ACTIVIDADES 4.1



- Se quiere crear una biblioteca de libros en un documento XML. Para ello se necesita crear una DTD que almacene los siguientes campos de un libro:
- Código de libro.
  - Título.
  - Editorial.
  - Edición.
  - ISBN.
  - Número de páginas.
  - Autor.

## 4.2 ESQUEMAS

Hasta el momento se ha hablado de cómo generar un lenguaje XML con la definición de las reglas que lo componen (DTD). Otra manera de formalizar esas reglas es con un lenguaje llamado **XML Schema** (también llamado **XSD** o **XML Schema Definition**). Se puede decir que un XSD (o esquema de ahora en adelante), es la evolución natural de un DTD. Permite expresar con mayor potencia, gramáticas más complejas utilizando la misma sintaxis de XML (lo que facilita enormemente el trabajo). Nació en el 1998 y se recomendó el uso en el 2001 por el W3C (*World Wide Web Consortium*). Las características principales de un esquema son las siguientes:

- ✓ Define qué elementos pueden aparecer en un documento XML.
- ✓ Define qué atributos pueden aparecer en un documento XML.
- ✓ Define qué elementos son compuestos, indicando qué elementos hijos deben aparecer y en qué orden.
- ✓ Define qué elementos pueden ser vacíos o que pueden incluir texto asociado.
- ✓ Define los tipos que pueden utilizarse en cada elemento o atributo.
- ✓ Define la obligatoriedad, la optatividad de elementos y/o atributos.

Si se analiza con un DTD, prácticamente permite realizar las mismas funcionalidades. Entonces, ¿qué diferencia un DTD de un esquema XSD si sirven para lo mismo?

La principal ventaja de XSD es que al estar basado en XML, es fácilmente extensible a las futuras modificaciones o necesidades que se identifiquen. Además no es necesario aprender un nuevo lenguaje (en contraposición con los DTD). También permite definir de manera muy clara los tipos de datos y los espacios de nombres que se soportan. Esto es realmente importante puesto que si un elemento se sabe que es de tipo *double* y, finalmente, se escribe como *string* o *integer*, no se admitirá la validez de un documento XML hasta que se corrija ese error. Es más, se pueden definir los tipos exactamente igual que si fuese una base de datos, facilitando la conversión de uno a otro y las transferencias de información. Más adelante hablaremos de la validez de los documentos.

Como en la sección anterior, partiremos de un documento XML y definiremos el esquema. En la base de datos de SMS, teníamos mensajes almacenados como los siguientes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BDsms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.gmrv.es BDsmsSchema.xsd">
<sms>
 <telefono>955 55 66 55</telefono>
 <fecha>1/7/2011</fecha>
 <hora>23:55</hora>
 <mensaje>Juego1: Tetris</mensaje>
</sms>
<sms>
 <telefono>745 15 56 11</telefono>
 <fecha>22/9/2011</fecha>
 <hora>15:05</hora>
 <mensaje>Juego2: Arkanoid</mensaje>
</sms>
<sms>
 <telefono>842 35 22 00</telefono>
 <fecha>10/11/2011</fecha>
```

```
<hora>09:22</hora>
<mensaje>Juego3: Comecocos</mensaje>
</sms>
</BDsms>
```

Si el esquema está en un fichero local, en vez de usar *xsi:schemaLocation* indicando una ubicación en la red, podemos utilizar esta otra forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BDsms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BDsmsSchema.xsd">
```

Antes teníamos una DTD que permitía validarla. Ahora definiremos el esquema en los mismos términos que el anterior (fichero *BDsmsSchema.xsd*):

```
<?xml version="1.0" ?>
<xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
 version="0.1" xml:lang="es">
 <xselement name="BDsms">
 <xsccomplexType>
 <xsssequence>
 <xselement name="sms" maxOccurs="unbounded">
 <xsccomplexType>
 <xsssequence>
 <xselement name="telefono" type="xss:string"/>
 <xselement name="fecha" type="xss:string"/>
 <xselement name="hora" type="xss:string"/>
 <xselement name="mensaje" type="xss:string"/>
 </xsssequence>
 </xsccomplexType>
 </xselement>
 </xsssequence>
 </xsccomplexType>
 </xselement>
</xsschema>
```

En negrita están destacados los elementos mínimos que componen un esquema.

#### 4.2.1 ELEMENTO RAÍZ

Un esquema se puede componer de muchos elementos enlazados. El elemento principal es el elemento raíz, y deberá estar siempre.

```
<?xml version="1.0" ?>
<xsschema>
...
</xsschema>
```

Este elemento *raíz* puede contener algunos atributos interesantes. En el ejemplo anterior se vio que se indicaba el espacio de nombres, una versión y un lenguaje predefinido. Entre los atributos más importantes se encuentra el siguiente:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

Indica cuál es el espacio de nombres en el que se basa para saber qué elementos y tipos de datos son los soportados en el esquema.

#### 4.2.2 ELEMENTOS SIMPLES

El resto de los elementos son los que el usuario puede definir para declarar un lenguaje XML. Llamamos elemento simple a aquel que puede contener información y que no tiene otros elementos hijos asociados a él. Tampoco puede tener atributos en su interior. En argot de árbol jerárquico, serían en los nodos hoja en los que solo habrá texto.

Para definir un elemento nuevo, la sintaxis a seguir es la siguiente:

```
<xs:element name="nombre_elemento" type="tipo_elemento">
```

Destaca que el nombre puede ser el que desee el usuario (mientras que no esté repetido). Sin embargo, el tipo tiene que ser uno de los siguientes:

■ **xs:string**

```
<xs:element name="nombre" type="xs:string">
<nombre>Javier Toledano</nombre>
```

■ **xs:date**

```
<xs:element name="fechaNacimiento" type="xs:date">
<fechaNacimiento>1979-02-04</fechaNacimiento>
<fechaNacimiento>1979-02-04+01:00</fechaNacimiento>
```

■ **xs:time**

```
<xs:element name="hora" type="xs:date">
<hora>23:55:15</hora>
<hora>23:55:15+01:00</hora>
<hora>23:55:15.1</hora>
```

■ **xs:dateTime**

```
<xs:element name="fecha" type="xs:date">
<fecha>1979-02-04T23:55:15</fecha>
<fecha>1979-02-04T23:55:15+01:00</fecha>
<fecha>1979-02-04T23:55:15.1</fecha>
```

■ **xs:decimal**

```
<xs:element name="precio" type="xs:decimal">
<precio>1205.74</precio>
<precio>-1205.74</precio>
```

### ■ **xs:integer**

```
<xs:element name="vueltas" type="xs:integer">
<vueltas>1205</vueltas>
<vueltas>-1205</vueltas>
```

### ■ **xs:boolean**

```
<xs:element name="pagado" type="xs:boolean">
<pagado>true</pagado>
<pagado>false</pagado>
```

### ■ **xs:hexBinary**

```
<xs:element name="imagen" type="xs:hexBinary">
<xs:element name="foto" type="xs:base64Binary">
```

<!-- entre las etiquetas va una codificación binaria en hexadecimal o base64. Se utiliza para almacenar documentos o formatos binarios. -->

Imaginemos que queremos establecer un conjunto de elementos simples para almacenar la información de un alumno de una escuela. Los datos a almacenar serían:

- ✓ Código de alumno.
- ✓ Nombre.
- ✓ Apellido1.
- ✓ Apellido2.
- ✓ D.N.I.
- ✓ Fecha de nacimiento.
- ✓ Curso.
- ✓ Cuota pagada (*booleano*).

Podríamos definir los siguientes elementos:

```
<xs:element name="codigo" type="xs:integer"/>
<xs:element name="nombre" type="xs:string"/>
<xs:element name="apellido1" type="xs:string"/>
<xs:element name="apellido2" type="xs:string"/>
<xs:element name="dni" type="xs:string"/>
<xs:element name="fechaNacimiento" type="xs:date"/>
<xs:element name="curso" type="xs:integer"/>
<xs:element name="cuotaPagada" type="xs:boolean"/>
```

Imaginemos que se quiere indicar, por defecto, que todo alumno tiene la cuota sin pagar. Cuando se define el elemento *cuotaPagada* podemos añadir ese valor por defecto de la siguiente forma:

```
<xs:element name="cuotaPagada" type="xs:boolean"
default="false"/>
```

En cambio, si lo que se quiere es que tenga un valor fijo siempre, el cambio sería el siguiente:

```
<xs:element name="cuotaPagada" type="xs:boolean"
fixed="false"/>
```

### 4.2.3 ATRIBUTOS

Los atributos son complementos de información que se pueden asignar a un elemento previamente declarado. La sintaxis de un atributo es la siguiente:

```
<xs:attribute name="nombre_atributo" type="tipo_atributo">
```

En los campos *name* y *type* se aplica exactamente lo mismo que lo visto en los elementos simples.

Siguiendo con el ejemplo de los alumnos de una escuela, nos piden añadir un atributo al elemento curso puesto que en un colegio puede darse el caso de tener varios cursos concurrentes y no pertenecer a la misma clase (por ejemplo, 1º letra A, 1º letra B...).

```
<curso letra="A">1</curso>
```

Debemos modificar lo siguiente en el esquema:

```
<xs:element name="curso" type="xs:integer"/>
<xs:attribute name="letra" type="xs:string"/>
```

Para terminar, es necesario indicar que valores por defecto, fijos o valores opcionales puede tomar un atributo. A continuación se ven ejemplos de los tres casos:

```
<xs:attribute name="letra" type="xs:string" default="A"/>
<xs:attribute name="letra" type="xs:string" fixed="A"/>
<xs:attribute name="letra" type="xs:string" use="required"/>
```

### 4.2.4 RESTRICCIONES

En algunos momentos es importante establecer un rango de valores en los que un elemento puede moverse. Hasta el momento no hemos podido realizar ese tipo de condicionantes por lo que se mostrará un ejemplo de cómo realizarlo. Imaginemos que los alumnos anteriores deben cumplir la condición de estar entre los 16 y los 24 años para poder hacerse el carnet joven. Si no está en ese rango, no podrían solicitarlo. En primer lugar estableceremos un nuevo atributo llamado edad (aunque podría calcularse la edad por la fecha de nacimiento y la actual). A continuación, indicamos la restricción:

```
<xs:element name="edad">
 <xs:simpleType>
 <xs:restriction base="xs:integer">
 <xs:minInclusive value="16"/>
 <xs:maxInclusive value="24"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Las restricciones no son solo útiles para los elementos sino que se pueden aplicar a los atributos también. En este caso vamos a declarar una restricción sobre el atributo "letra", para que tome uno de los valores determinados en una lista ("A", "B", "C", "D").

```
<xs:attribute name="letra">
 <xs:simpleType>
```

```

<xs:restriction base="xs:string">
 <xs:enumeration value="A"/>
 <xs:enumeration value="B"/>
 <xs:enumeration value="C"/>
 <xs:enumeration value="D"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>

```

También sería una solución válida la siguiente:

```

<xs:attribute name="letra" type="letrasCursos"/>
<xs:simpleType name="letrasCursos"/>
 <xs:restriction base="xs:string">
 <xs:pattern value="[A-D]"/>
 </xs:restriction>
</xs:attribute>

```

En este último caso se ha definido un tipo simple llamado *letrasCursos* que podrá ser reutilizado en otras partes del esquema. Cabe destacar que en vez de realizar una enumeración, se ha definido un patrón en forma de expresión regular, lo que permite tener mayor potencia para declarar valores predeterminados. También hubiera válido cualquiera de las siguientes expresiones regulares:

```

<xs:pattern value="[ABCD]"/>
<xs:pattern value="A | B | C | D"/>

```

También hay posibilidad de establecer restricciones sobre la longitud de lo contenido por un elemento. El siguiente ejemplo clarifica bastante esta restricción:

```

<xs:element name="dni">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:length value="10"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>

```

- **xs:length**: establece una longitud fija.
- **xs:minLength**: establece un mínimo en la longitud.
- **xsmaxLength**: establece un máximo en la longitud.

Existe otro tipo de restricciones que van asociadas al contenido que almacena el elemento. En concreto son importantes aquellas que permiten definir el comportamiento que debe tener el procesador XML cuando se encuentra un espacio en blanco. Imaginemos que se quiere almacenar la dirección de vivienda habitual del alumno. Es claro que deberíamos crear un nuevo elemento:

```

<xs:element name="direccion">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:whiteSpace value="preserve"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>

```

En el caso de los espacios en blanco nos encontramos con las siguientes opciones:

- **preserve**: mantiene los espacios en blanco que almacene entre sus etiquetas de apertura y cierre (realmente mantiene los espacios, tabuladores, saltos de línea y retornos de carro).
- **collapse**: borra todos los espacios en blanco que encuentre entre sus etiquetas de apertura y cierre (espacios, tabuladores, saltos de línea y retornos de carro).
- **replace**: sustituye todos los espacios en blanco, tabuladores, saltos de línea y retornos de carro por un único espacio en blanco.

Hemos visto las más representativas a la hora de definir un esquema XML, pero existen más, cuyo nombre define su funcionalidad perfectamente:

- **xs:maxExclusive**: es el máximo valor. Serían válidos todos aquellos que sean menores que el indicado.
- **xs:minExclusive**: es el mínimo valor. Serían válidos todos aquellos que sean mayores que el indicado.
- **xs:totalDigits**: indica el número exacto de cifras permitidas en el elemento o atributo. Debe ser mayor que cero.
- **xs:fractionDigits**: indica el número máximo de decimales que se permiten. Debe ser mayor o igual a cero.

#### 4.2.5 ELEMENTOS COMPLEJOS

En contraposición a los elementos simples, los elementos complejos son todos aquellos que se componen de otros elementos y/o que puedan contener atributos propios. Podemos identificar como elementos complejos los siguientes:

- ✓ Elementos que contienen otros elementos en su interior.
- ✓ Elementos que contienen atributos en su interior.
- ✓ Elementos que contienen otros elementos y atributos en su interior.

Un ejemplo de elemento complejo es el siguiente (por tener un atributo):

```
<curso letra="A">1</curso>
```

En el ejemplo de la base de datos de SMS, un elemento complejo, por tener otros elementos en su interior, podría ser el elemento *BDsms*:

```
<xs:element name="BDsms">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="sms" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="telefono" type="xs:string"/>
 <xs:element name="fecha" type="xs:string"/>
 <xs:element name="hora" type="xs:string"/>
 <xs:element name="mensaje" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

Podemos ver que el elemento *BDsms* se compone de elementos *sms*. Y un *sms* se compone de elementos de tipo *telefono*, *fecha*, *hora* y *mensaje*. A la hora de definir un elemento complejo en el esquema, hay que indicarlo de alguna manera. Esta manera es añadiendo, a continuación, un elemento llamado `<xs:complexType>`. Justo después de añadirlo, se inserta otro elemento llamado `<xs:sequence>` que permite indicar que los siguientes elementos a definir se declaran como hijos del elemento inicial (en esa determinada secuencia). Así se define un elemento con hijos como subelementos.

#### 4.2.6 SECUENCIA DE ELEMENTOS

En el punto anterior, se hablaba del elemento `<xs:sequence>`, que permite indicar una determinada secuencia de hijos en un elemento complejo. Pero no se ha hablado de los indicadores que permiten, por ejemplo, marcar cuántas veces deben aparecer determinados elementos. Es lo que se llama **indicadores**.

Un indicador permite determinar los siguientes patrones:

- ✓ El orden en el que se establecen los elementos hijos.
- ✓ La ocurrencia con la que aparecen los elementos hijos.
- ✓ El grupo al que pertenecen los elementos hijos.

Los **indicadores de orden** permiten saber cómo se pueden ubicar los elementos hijos. Dentro de este tipo de indicador podemos encontrarnos tres:

1. `<xs:all>`: especifica que todos los elementos hijos pueden aparecer en cualquier orden determinado, siempre que solo aparezcan una única vez.
2. `<xs:Choice>`: especifica que de entre los elementos hijos solo pueden aparecer o uno u otro.
3. `<xs:sequence>`: ya lo conocemos y permite indicar el orden específico en el que deben aparecer los elementos hijos.

Los **indicadores de ocurrencia** permiten conocer cuantas veces pueden repetirse cada elemento hijo. Podemos distinguir los siguientes:

1. `<xs:maxOccurs>`: indica el máximo número de veces que un elemento hijo puede aparecer.
2. `<xs:minOccurs>`: indica el mínimo número de veces que un elemento hijo puede aparecer.

En estos dos indicadores, los valores por defecto son una vez. Si pudiera concurrir un número ilimitado de veces un elemento hijo, se indicaría de la siguiente forma:

```
<xs:element name="sms" maxOccurs="unbounded">
```

Los **indicadores de grupo** facilitan la manera de establecer un conjunto de elementos asociados entre si. La mejor manera de ver su funcionalidad es con un ejemplo. Imaginemos que queremos representar las características de un coche. Nos gustaría almacenar lo siguiente:

- ✓ Marca.
- ✓ Modelo.
- ✓ Caballos.

Como cualquier automóvil tiene esas características, la definición del esquema podría quedar así:

```
<xs:group name="grupoCoche">
<xs:sequence>
<xs:element name="marca" type="xs:string"/>
<xs:element name="modelo" type="xs:string"/>
<xs:element name="caballos" type="xs:integer"/>
</xs:sequence>
</xs:group>
<xs:element name="coche" type="TipoCoche"/>
<xs:complexType name="TipoCoche">
<xs:sequence>
<xs:element name="codigo" type="xs:integer"/>
<xs:group ref="grupoCoche"/>
<xs:element name="combustible" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

El mismo concepto se puede seguir con grupos de atributos:

```
<xs:attributeGroup name="grupoCoche">
<xs:element name="marca" type="xs:string"/>
<xs:element name="modelo" type="xs:string"/>
<xs:element name="caballos" type="xs:integer"/>
</xs:attributeGroup>
```

Los grupos permiten modularizar los elementos en pequeños trozos y reutilizarlos si fuera necesario en otras partes del esquema que se define.

Para finalizar, existe un elemento que permiten extender la funcionalidad de un documento XML sin tener que especificar lo que debe venir en el esquema. Es como la utilización de tipos de objetos *void* en C++, es decir, podemos esperarnos cualquier tipo de objeto (en nuestro caso, elemento). En este caso el elemento se llama *<xs:any>*. Con los indicadores de ocurrencia podemos indicar cuantos elementos genéricos podemos encontrarnos en ese punto en el esquema. Suele dar flexibilidad a los documentos pero desvirtua ligeramente el cometido principal de los esquemas (que es tener bien definido todo lo que te puedes encontrar en un documento XML). A efectos, el elemento *<xs:anyAttribute />* permite realizar lo mismo pero a nivel de atributos.



## ¿SABÍAS QUE...?

**RELAX NG (REgular LAnguage for XML Next Generation)** es una nueva manera de especificar un patrón para la estructura y el contenido de un documento XML. A efectos es como declarar una DTD o un XML Schemas. La principal ventaja es que es más sencillo de utilizar y bastante más intuitivo.



## ¿SABÍAS QUE...?

*Schematron* es un lenguaje de validación basado en reglas (en vez de gramáticas). Se suele utilizar para extender la funcionalidad de DTD, XML Schema o RELAX NG.

## ACTIVIDADES 4.2



- Se quiere crear una biblioteca de libros en un documento XML. Para ello se necesita crear un esquema que almacene los siguientes campos de un libro:
- Código de libro.
  - Título.
  - Editorial.
  - Edición.
  - ISBN.
  - Número de páginas.
  - Autor.

## 4.3 VALIDACIÓN DE DOCUMENTOS XML

La validación de los documentos XML es una serie de comprobaciones que permiten saber si el documento XML está bien formado y si se ajusta a una estructura previamente definida (ya sea DTD o esquemas). Cuando se habla de documento bien formado se quiere decir que se siguen las normas y reglas básicas que se establecen en todos los documentos XML. Cuando se habla de un documento válido, quiere decir que además de estar bien formado, su estructura es acorde con las normas que se dictan en la definición del tipo de documento o esquema asociado.

En resumen, ¿por qué son necesarios los procesos de validación?

- ✓ Porque nos permite asegurar que en una transferencia de información XML entre un emisor y un receptor, ambos interpretarán su contenido de igual manera.
- ✓ Porque nos permite asegurar que el documento contiene toda la información declarada como obligatoria.
- ✓ Porque los datos vendrán en un formato conocido y correcto.

Se hablará a continuación de ambos casos.

### 4.3.1 DOCUMENTOS BIEN FORMADOS

Todo documento XML que quiera serlo, debe cumplir una serie de normas básicas:

- Al principio del documento se debe declarar una cabecera (en formato etiqueta) que indique lo siguiente:
  - Versión del documento XML. Actualmente se puede indicar la versión 1.0 ó 1.1
  - Tipo de codificación utilizada. De esta manera se sabe que juego de caracteres se están utilizando. Lo normal es utilizar una codificación UTF-8, ISO-8859-1 o ISO-8859-15.
  - Si está basado en un documento de estructura externo (DTD o esquema). Si es así, se indicaría en el atributo *standalone="yes"*. Por defecto es que no.
- <?xml version="1.0" encoding="UTF-8" standalone="no"?>
- Todas las entidades, elementos y atributos deben tener una sintaxis correcta:
  - XML distingue entre mayúsculas y minúsculas.
  - Los nombres de los elementos pueden ser alfanuméricos pero empezando obligatoriamente por letra.
  - Elemento que se abra <*etiq*>, deberá cerrarse </*etiq*>.
  - Los valores que se indiquen en los atributos deberán ir entre comillas dobles o simples. <*etiq* *atributo*="1">.
  - Si hay un elemento vacío *EMPTY*, deberá autocerrarse. <*etiq*>.
- En la estructura del documento XML, se encadenarán los elementos en un formato jerárquico, en el que solamente habrá un elemento raíz.
- Si se usan entidades definidas por el usuario dentro del documento XML, en la DTD deberán estar declaradas.

### 4.3.2 DOCUMENTOS VÁLIDOS

Un documento XML válido es aquel que, además de estar bien formado, está acorde con la especificación de un DTD o esquema. Cuando se valida un documento XML se comprueban las siguientes cuestiones:

- Solo se pueden utilizar elementos o atributos definidos en el DTD o esquema. Cualquier elemento no declarado hará que el documento no pase el proceso de validación (análisis léxico).
- Que los elementos y atributos estén en el orden definido en el DTD o esquema. Además se comprueba que los atributos o elementos obligatorios estén presentes dentro del documento (análisis sintáctico).
- Si se declaran valores que deben ser únicos, como por ejemplo, atributos ID, debe asegurarse que en todo el documento XML se cumple esta condición.
- Si se define un tipo concreto para los atributos o elementos, los valores que tomen estarán tipados a la definición anterior. No es lo mismo declarar un tipo entero que un tipo *string*.

### 4.3.3 HERRAMIENTAS PARA VALIDAR

Aunque se puede validar echando un simple vistazo en el documento XML, cuando los tamaños a manejar son grandes no es la mejor manera. En la actualidad existen muchas herramientas que nos permiten comprobar que se tienen documentos bien formados y que cumplen los criterios de validez. A continuación enumeraremos algunas herramientas bastante conocidas que permitirán realizar esta tarea de manera fácil y sencilla:

- **Vía web** (la mayoría permite insertar en un formulario el documento XML con las DTD o esquemas):
  - [validator.w3.org](http://validator.w3.org)<sup>5</sup>.
  - [xmlvalidation.com](http://www.xmlvalidation.com)<sup>6</sup>: permite insertar en un formulario un fichero XML con las DTD incluidas o con un esquema externo y verifica la validez del documento.
  - [validome.org](http://www.validome.org/xml/)<sup>7</sup>.

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below that, there are three tabs: "Validate by URI", "Validate by File Upload", and "Validate by Direct Input". The "Validate by Direct Input" tab is selected. It has a text area labeled "Enter the Markup to validate:" containing the following XML code:

```

<?ELEMENT ?DENT !PCDATA>
[<ELEMENT mensaje !PCDATA>]
>

<bDoms>
<?xml version="1.0" encoding="UTF-8"?>
<telefono>953 55 66 55</telefono>
<fecha>22/09/2011</fecha>
<hora>21:55</hora>
<mensaje>Juego1: Tetris</mensaje>
<sms>
<telefono>745 15 55 11</telefono>
<fecha>22/9/2011</fecha>
<hora>22:00</hora>
<mensaje>Juego2: Arkano</mensaje>
<sms>
<telefono>842 35 22 00</telefono>
<fecha>18/11/2011</fecha>
<hora>09:22</hora>
<mensaje>Juego3: Comecocos</mensaje>
</sms>
</bDoms>

```

Below the code, there's a "More Options" link and a "Check" button.

Figura 4.1. Formulario validación W3C

The screenshot shows the W3C Markup Validation Service interface after a successful validation. At the top, it says "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below that, there are two tabs: "Jump To:" and "Notes and Potential Issues". The "Notes and Potential Issues" tab is selected. It displays the message "This document was successfully checked as XML!". Below this, it shows the XML code from Figure 4.1. Further down, there are fields for "Result:" (Passed), "Source:", "Encoding:" (utf-8), "Doctype:" (XML), and "Root Element:" (bDoms). At the bottom, there's a note: "The W3C validators rely on community support for hosting and development. Donate and help us build better tools for a better web." and a "1533" link.

Figura 4.2. Resultados validación W3C

5 <http://validator.w3.org>

6 <http://www.xmlvalidation.com>

7 <http://www.validome.org/xml/>

## ■ Aplicaciones:

- **Serna Free**<sup>8</sup>: editor de documentos XML que permite analizar la validez y aplicar los estilos previamente definidos.
- **TotalEdit**<sup>9</sup>: es un editor multilenguaje que puede ser utilizado también para tratar los documentos XML.
- **XML Notepad**<sup>10</sup>: editor de Microsoft que facilita el trabajo con documentos XML. Algunos lo denominan el *Visual Studio para XML*.
- **<oXygen/>**<sup>11</sup>: anteriormente llamado *XMLSpy*. Posiblemente el mejor editor de XML del momento.

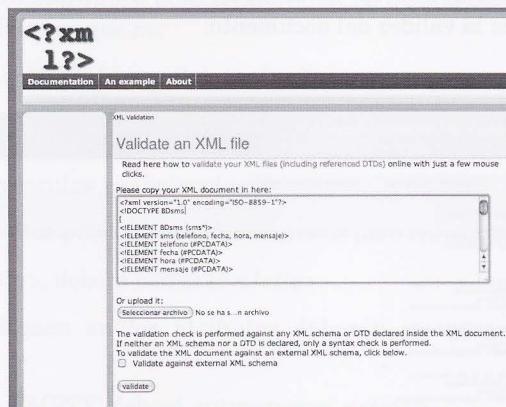
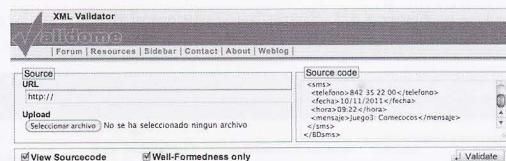


Figura 4.3. Formulario validación [xmlvalidation.com](http://xmlvalidation.com)



### Validator for XML Documents

Allows you to check your XML documents on conformance to W3C specifications for XML 1.0. The validator provides documentation, including the most important points to:

Please note: When expected to personalized (but still XML conform) grammars, we check the document against the grammar (DTD or Schema) you pointed to within document. Please specify an available URL to your XML DTD or Schema, so Validome can process validation.

This service provides XML-validation only. In order to validate other documents, please use the appropriate services.

#### Other Validators:

[HTML / XHTML](#) Validates HTML, XHTML and WML documents. > [WML / XHTML / HTML Validator](#) <

[DTD / Schema](#) Checks XML grammar only. > [DTD / Schema Validator](#) <

[RSS / Atom](#) Check your feeds with the > [RSS and Atom Validator](#) <

[Google Sitemaps](#) The > [Google Sitemap\(s\) Validator](#) helps you checking XML conformity and other specific requirements

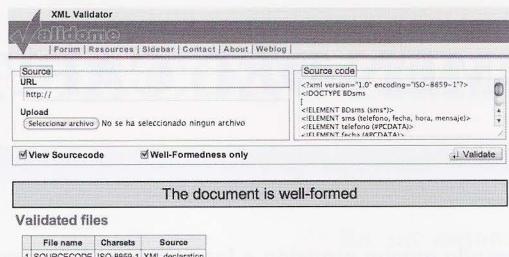
Figura 4.4. Formulario validación [validome.org](http://validome.org)

8 <http://www.syntext.com/products/serna/>

9 <http://www.codertools.com>

10 <http://www.microsoft.com>

11 <http://www.oxygenxml.com>



*Figura 4.5. Resultados validación validome.org*

Podrá encontrar más información en el capítulo de herramientas.

## ACTIVIDADES 4.3

- ▶ Crear un documento XML que contenga la información de al menos 10 libros de la biblioteca que se definieron en la DTD y el esquema de las actividades anteriores.
- ▶ Verificar que el documento XML está bien formado y es válido para la definición DTD y para el esquema. Se recomienda utilizar los validadores web para encontrar los errores.

## 4.4 CASO PRÁCTICO

Una comunidad de propietarios, en su Junta General Ordinaria, decide crear un sistema de almacenamiento de la información para almacenar datos de la Comunidad. De esta manera podrán realizar las gestiones de manera más ágil y con un estándar de comunicación con el resto de acreedores y deudores de la Comunidad. En principio los campos a almacenar son los siguientes:

- Código de vecino.
- Nombre.
- Apellidos.
- Portal.
- Piso y letra.
- Código de Cuenta Corriente (CCC).
- Cargo (Presidente, Vicepresidente, Secretario, Vocal, Ninguno).

Se pide lo siguiente:

- Generar una DTD que defina esta estructura de información.
- Generar un esquema que defina esta estructura de información.
- Generar un documento XML, con al menos 10 vecinos. Dos de ellos tendrán el cargo de Presidente y Vicepresidente (al menos).



## RESUMEN DEL CAPÍTULO



En este capítulo se han prestado mucha atención a las ventajas de tener documentos bien formados y válidos. Se ha clarificado la diferencia entre ellos (un documento válido es, de por sí, un documento bien formado).

Se han comentado las dos formas más comunes de definir la estructura de un documento XML para poder validarlos:

- Un DTD (*Document Type Definition*).
- XSD (*XML Schema Definition*) o esquemas.

En cada uno de ellos se ha explicado como declarar los elementos más destacados, incluyendo atributos y restricciones que permiten cambiar su ocurrencia, su secuencia o incluso su tipo de datos.

Finalmente se nombraron una serie de herramientas web y/o aplicaciones locales que aseguran que un documento XML está bien formado y es válido con respecto a una gramática determinada. Estas herramientas permitirán comprobar que el trabajo realizado en un documento XML y asociado a una DTD o esquema es correcto o, por el contrario, indicará donde se han cometido los errores para solucionarlos.



## EJERCICIOS PROPUESTOS



- 1. Crear una DTD que permita definir el almacenamiento de coches de un concesionario de coches de segunda mano. Para ello, los campos a almacenar serán los siguientes:
  - Código de coche.
  - Marca.
  - Modelo.
  - Matrícula.
  - Potencia (caballos).
  - Plazas.
  - Número de puertas.
- 2. Crear un esquema para el mismo ejemplo anterior.
- 3. Crear un documento XML que almacene un conjunto de coches. Asignar a dicho documento la DTD y comprobar su validez. Una vez hecho, realizar lo mismo pero con el esquema anteriormente declarado.



## TEST DE CONOCIMIENTOS



**1** ¿Qué permite definir una DTD?

- a) Define cómo se construye un documento XML válido. Para ello únicamente es necesario establecer una serie de reglas léxicas.
- b) Define cómo se construye un documento XML válido. Para ello únicamente es necesario establecer una serie de reglas sintácticas.
- c) Define cómo se construye un documento XML válido. Para ello únicamente es necesario establecer una serie de reglas léxicas y sintácticas.
- d) Ninguna de las anteriores.

**2** El elemento `<!ELEMENT BDsms (sms*)>`:

- a) Se permite de uno a infinito, el número de ocurrencias del elemento *sms*.
- b) Se permite de cero a infinito, el número de ocurrencias del elemento *sms*.
- c) Se permite de cero a uno, el número de ocurrencias del elemento *sms*.
- d) Ninguna de las anteriores.

**3** El elemento `<!ELEMENT BDsms (sms+)>`:

- a) Se permite de uno a infinito, el número de ocurrencias del elemento *sms*.
- b) Se permite de cero a infinito, el número de ocurrencias del elemento *sms*.
- c) Se permite de cero a uno, el número de ocurrencias del elemento *sms*.
- d) Ninguna de las anteriores.

**4** El elemento `<!ELEMENT BDsms (sms?)>`:

- a) Se permite de uno a infinito, el número de ocurrencias del elemento *sms*.
- b) Se permite de cero a infinito, el número de ocurrencias del elemento *sms*.
- c) Se permite de cero a uno, el número de ocurrencias del elemento *sms*.
- d) Ninguna de las anteriores.

**5** En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado `<xs:all>`, significa:

- a) Que el orden de todos los hijos es indiferente siempre que solo aparezcan una vez (en el caso de no cambiar la ocurrencia).
- b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
- c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
- d) Ninguna de las anteriores.

**6** En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado `<xs:Choice>`, significa:

- a) Que el orden de todos los hijos es indiferente siempre que solo aparezcan una vez (en el caso de no cambiar la ocurrencia).
- b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
- c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
- d) Ninguna de las anteriores.

**7** En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado `<xs:sequence>`, significa:

- a) Que el orden de todos los hijos es indiferente siempre que solo aparezcan una vez (en el caso de no cambiar la ocurrencia).
- b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
- c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
- d) Ninguna de las anteriores.

**8** En un esquema, si nos encontramos <xs:whiteSpace value="preserve"/>, significa:

- a) Que en el texto que se almacene podrá sustituir todos los espacios en blanco, tabuladores, saltos de línea y retornos de carro; por un único espacio en blanco.
- b) Que en el texto que se almacene se mantendrá inalterado, aún cuando se tengan multitud de espacios en blanco, tabuladores, saltos de línea y retornos de carro.
- c) Que en el texto que se almacene se eliminarán todos los espacios en blanco, tabuladores, saltos de línea y retornos de carro.
- d) Ninguna de las anteriores.

**9** Un documento está bien formado:

- a) Si tiene una cabecera que indique que el documento es XML.
- b) Si tiene una sintaxis correcta todas las entidades, elementos y atributos.

c) Si se tiene una estructura jerárquica en la que solo puede haber un nodo raíz.

d) Si se han definido correctamente todas las entidades, elementos y atributos en una DTD o esquema.

e) Todas las anteriores.

f) Ninguna de las anteriores.

**10** Un documento es válido:

- a) Si está bien formado.
- b) Si en el análisis de validez del documento con respecto a una DTD o esquema se cumplen todas las reglas sintácticas que en él se definen.
- c) A y B, en su conjunto definen un documento válido.
- d) A y si se tiene una estructura jerárquica con multinodo raíz.
- e) Todas las anteriores.
- f) Ninguna de las anteriores.

# 5

# Conversión y adaptación de documentos XML

## OBJETIVOS DEL CAPÍTULO

- ✓ Conocer XSL como herramienta de transformación de documentos XML.
- ✓ Saber qué elementos básicos pueden utilizarse en la definición de estilos XSL.
- ✓ Utilización de plantillas para facilitar el proceso de conversión y transformación de documentos XML.
- ✓ Conocer qué operadores permiten discriminar los datos para aplicar estilos diferentes.
- ✓ Realizar ejercicios prácticos que afiancen los conceptos de la transformación a través de XSL.

Tal y como se ha explicado en capítulos anteriores, la tecnología XML permite separar de manera efectiva los datos a almacenar, la estructura o semántica en la que se organizan y la presentación de los mismos.

Aunque podemos visualizar un fichero XML con un simple editor de texto, no es la manera más amigable ni profesional para presentar los datos que están almacenados en su interior. Es ahí donde debemos utilizar alguna herramienta que nos permita convertir y transformar los datos en el formato que deseemos. Esta herramienta se llama XSL (*Extensible Stylesheet Language*).

XSL es a XML, lo que las hojas de estilo en cascada (CSS) a HTML. XSL permite tomar pleno control sobre los datos, pudiendo establecerse criterios como qué datos ver, en qué orden visualizarlos, estableciendo filtros y definiendo formatos de salida para su representación. Es, por tanto, una herramienta de procesado muy potente que hay que conocer en profundidad.

Para entrar dentro del contexto de la conversión y transformación de ficheros XML, se define el siguiente ejemplo con código XML que permitirá almacenar un conjunto de libros en una librería:



### EJEMPLO 5.1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<libreria>
 <libro>
 <titulo>Physics-based animation</titulo>
 <autor>Kenny Erleben</autor>
 <editor>Charles River Media</editor>
 <isbn>978-1584503804</isbn>
 <precio>50.06</precio>
 </libro>
 <libro>
 <titulo>Principios de seguridad informática para usuarios</titulo>
 <autor>Carlos Garre</autor>
 <editor>Dykinson</editor>
 <isbn>978-84-9849-998-8</isbn>
 <precio>13.00</precio>
 </libro>
 <libro>
 <titulo>Ejercicios complementarios de lógica digital</titulo>
 <autor>Alberto Sánchez</autor>
 <editor>Dykinson</editor>
 <isbn>978-84-9849-703-8</isbn>
 <precio>12.00</precio>
 </libro>
</libreria>
```

Puede verse claramente que la librería puede contener un conjunto de libros. Cada libro tiene una serie de atributos propios como puede ser el título, autor o ISBN. En el ejemplo anterior se han almacenado tres libros y se quiere visualizarlos de una manera más amigable. Se podría definir una XSL como la siguiente:



## EJEMPLO 5.2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
 <h1>Mi biblioteca</h1>
 <table>
 <tr bgcolor="#887788">
 <th>Título</th>
 <th>Autor</th>
 </tr>
 <xsl:for-each select="libreria/libro">
 <tr>
 <td><xsl:value-of select="titulo"/></td>
 <td><xsl:value-of select="autor"/></td>
 </tr>
 </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Esta hoja de estilo XSL define que por cada libro almacenado en la librería, se añade una entrada en una tabla HTML con los valores del título y el nombre del autor. El resultado de vincular al XML una hoja de estilo XSL permite obtener un resultado como el siguiente:

### Mi biblioteca

Título	Autor
Physics-based animation	Kenny Erleben
Principios de seguridad informática para usuarios	Carlos Gámez
Ejercicios complementarios de lógica digital	Alberto Sánchez

*Figura 5.1. Aplicación de un XSL básico*

Este es un pequeño ejemplo de conversión. En principio se puede transformar un documento XML a otro XML o cualquier otro distinto que pueda ser reconocido por un navegador web (HTML o XHTML).

Si se quiere utilizar correctamente esta tecnología de representación de documentos XML, se deben seguir los siguientes pasos:

1. Tener bien definido el documento XML.
2. Crear una hoja de estilo XSL bien formada.
3. Vincular al documento XML la hoja de estilo XSL.

Los dos primeros puntos asumimos que se cumplen al estar determinados por los DTDs o esquemas definidos anteriormente. Además se indica sobre qué espacio de nombres se trabaja por lo que es fácil cumplir las dos condiciones iniciales. Para vincular en un documento XML una hoja de estilo XSL cualquiera, por ejemplo “tablaBiblioteca.xsl”, es tan simple como añadir al fichero XML lo siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xmlstylesheet type="text/xsl" href="tablaBiblioteca.xsl"?>
<libreria>
...
</libreria>
```

Cumplidos los pasos anteriores, el procesador de documentos XSL comienza a recorrer el árbol del documento XML. Se inicia por el nodo raíz y se irá recorriendo todo el árbol haciendo que cada parte del documento XML pueda tener un formato específico (árbol de resultados). Pero, ¿cómo determinar qué formatos tendrán cada una de las partes del documento?

La respuesta a la pregunta es la definición de plantillas en la hoja de estilos. Una plantilla es un patrón que cuando se cumple puede generar resultados de formateo en el árbol final. Es decir, con un documento de origen que se recorre por completo buscando concordancias con las plantillas definidas, de manera que si se cumple alguna, se aplica el formato definido y se genera en el árbol del documento final.

## ACTIVIDADES 5.1



- ▶ Insertar varios libros más en el fichero XML indicado cuyo coste sea inferior a 10.50 €.
- ▶ Insertar el dato número de páginas en cada libro contenido en el fichero XML inicial. Por ejemplo, el elemento será <numPaginas>150<numPaginas>.
- ▶ Cambiar el título de la tabla a “Mi biblioteca personal”.
- ▶ Cambiar el color de la tabla en el fichero XSL.
- ▶ Añadir una columna más al final en la que se muestre el precio del libro.
- ▶ Añadir una columna más al principio en la que se muestre el ISBN.

## 5.1 TRANSFORMACIÓN DE DOCUMENTOS (XSL)

XSL es un lenguaje que nos permite definir un conjunto de reglas que, aplicados sobre un documento XML, permite transformarlo en un resultado formateado más adecuado a nuestros intereses. Para ello, se necesita almacenar esas reglas dentro de un fichero llamado hoja de estilo XSL.

El fichero que almacena la hoja de estilo XSL, como fichero XML que es, debe estar bien formado. Usando el ejemplo anterior de los libros, se puede ver que el fichero XSL posee una declaración inicial que determina su contenido como XML y su codificación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

A continuación se indica la etiqueta que identifica al fichero XSL y el espacio de nombres en el que se basa. En este caso tenemos dos opciones:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

O

```
<xsl:transform version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Las dos maneras están soportadas en el estándar establecido en el consorcio W3C y se han elevado a recomendaciones de uso. Se puede observar que, independientemente de la que se utilice, ambas hacen referencia a un espacio de nombres común en su versión 1.0.

La siguiente etiqueta interesante es la que nos permite definir un elemento plantilla dentro del XSL. Todo lo que quede entre las siguientes etiquetas:

```
<xsl:template match="/">
 ...
</xsl:template>
```

será lo que permitirá generar la salida formateada. Cabe destacar que con esta etiqueta se puede indicar sobre qué parte del documento XML se quiere actuar, utilizando el atributo “match” para ello. En este caso se ha querido actuar sobre la raíz del propio documento XML.

## 5.2 ELEMENTOS BÁSICOS

### 5.2.1 XSL:FOR-EACH

En este momento tenemos en el fichero XML un conjunto de libros bien identificados. Es posible que al usuario final le interese recorrerse todos y cada uno de los libros y extraer los datos para darles un nuevo formato. En este caso, la siguiente etiqueta indica que se recorran todo el conjunto de elementos XML que sean libros:

```
<xsl:for-each select="libreria/libro">
 ...
</xsl:for-each>
```

### 5.2.2 XSL:VALUE-OF

Finalmente queda extraer el contenido del libro. Para ello la etiqueta que extrae la información de ese elemento XML seleccionado es la siguiente:

```
<xsl:value-of select="titulo"/>
```

Esta es la manera más sencilla de realizar un listado completo, formateado y con estilos adecuados, de todos los libros almacenados en la biblioteca. Aunque este ejemplo es muy simple, la potencia de las hojas de estilo XSL no acaba con estos elementos.

### 5.2.3 XSL:SORT

En este momento el lector podría indicar que tiene tantos libros almacenados que resultaría muy interesante mostrarlos de manera ordenada (y no tal y como se salvaron en el fichero XML). Si lo que interesa es ordenarlos por el título del libro, tras la etiqueta “*for-each*”, incluimos ésta:

```
<xsl:sort select="titulo"/>
```

#### Mi biblioteca

Título	Autor
Ejercicios complementarios de lógica digital	Alberto Sánchez
Physics-based animation	Kenny Erleben
Principios de seguridad informática para usuarios	Carlos Garre

Figura 5.2. Aplicar ordenación por título

Si por el contrario, el orden adecuado es por autor, entonces se debería incluir la siguiente:

```
<xsl:sort select="autor"/>
```

#### Mi biblioteca

Título	Autor
Ejercicios complementarios de lógica digital	Alberto Sánchez
Principios de seguridad informática para usuarios	Carlos Garre
Physics-based animation	Kenny Erleben

Figura 5.3. Aplicar ordenación por autor

Lo realmente interesante es que se puedan realizar búsquedas con determinados patrones preestablecidos. En este caso, parece lógico pensar que en vez de recorrerse todos los elementos, deberíamos filtrar por ese patrón (independientemente del orden que establezcamos). Si, por ejemplo, queremos extraer todos los libros del autor “Kenny Erleben”, se podría cambiar lo siguiente:

```
<xsl:for-each select="libreria/libro[autor='Kenny Erleben']">
...
</xsl:for-each>
```

#### Mi biblioteca

Título	Autor
Physics-based animation	Kenny Erleben

Figura 5.4. Búsqueda por autor “Kenny Erleben”

Si, por el contrario, lo que interesa es realizar una búsqueda con otro operador, como por ejemplo, todos los libros en los que el autor no es “*Carlos Garre*”, ordenado por autor, el cambio sería tan simple como éste:

```
<xsl:for-each select="libreria/libro[autor!='Carlos Garre']">
<xsl:sort select="autor"/>
...
</xsl:for-each>
```

## Mi biblioteca

Título	Autor
Ejercicios complementarios de lógica digital	Alberto Sánchez
Physics-based animation	Kenny Erleben

Figura 5.5. Búsqueda por autor distinto a “*Carlos Garre*”

## ACTIVIDADES 5.2



- Sin ningún patrón de búsqueda establecido, ordenar los resultados por precio.
- Insertar en el fichero XML que se indica inicialmente, un nuevo libro cuyo autor sea el mismo a uno ya incluido anteriormente.
- Buscar los libros del autor anterior y comprobar que salen todos sus libros (ordenados por precio).
- Buscar todos los libros que no son del autor anterior. Comprobar que en los resultados faltan sus libros.

## 5.3 OPERADORES EN XSL

### 5.3.1 ELEMENTO XSL:IF

Hasta este momento, se ha visto operadores de igualdad y desigualdad para cambiar el patrón de búsqueda pero no son los únicos que existen. Podrían usarse también los siguientes:

Los operadores lógicos que se pueden utilizar para cambiar el patrón de búsqueda o filtro son los siguientes:

- ✓ Operador de igualdad (=): “=”
- ✓ Operador de desigualdad (≠): “!=”
- ✓ Operador menor que (<): “&lt;”
- ✓ Operador mayor que (>): “&gt;”

En ocasiones se necesita mayor potencia a la hora de establecer un patrón de búsqueda. Los operadores lógicos vistos anteriormente son efectivos para filtros simples pero operadores lógicos no es suficiente. Existe la posibilidad de indicar, con el elemento “`<xsl:if>`”, condiciones más complejas en la evaluación del fichero XML. La sintaxis de este nuevo elemento es la siguiente:

```
<xsl:if test="expresión">
...
</xsl:if>
```

Imaginemos que queremos mostrar todos aquellos libros cuyo coste sea superior a 12 euros. Para ello, solo modificaremos las siguientes líneas del fichero XSL:

```
<xsl:for-each select="libreria/libro">
<xsl:if test="precio > 12">
...
</xsl:if>
</xsl:for-each>
```

### Mi biblioteca

Título	Autor
Physics-based animation	Kenny Erleben
Principios de seguridad informática para usuarios	Carlos Garre

Figura 5.6. Búsqueda por coste de libro mayor que 12 €

## ACTIVIDADES 5.3

- ▶ Añadir al fichero XML inicial (ejemplo 5.1), dos libros de un nuevo autor. Sus precios serán 10.50 € y 12.50 € respectivamente
- ▶ Realizar un fichero XSL que busque todos aquellos libros cuyo coste sea superior a 10 €. Asegurarse que aparece los libros del autor nuevo.
- ▶ En el mismo fichero XSL anterior, añadir una condición en la que solo se muestren aquellos cuyo autor sea el último que se ha definido.
- ▶ Cambiar la condición del coste del libro. Ahora solo se quiere aquellos libros cuyo coste sea superior a 12 €. Comprobar que solo aparece un libro de los dos anteriores añadidos.

### 5.3.2 ELEMENTO XSL:CHOOSE

El elemento “`<xsl:if>`” no es la única manera de condicionar los resultados. Existe otro elemento que nos permite establecer múltiples condiciones dentro del recorrido en el árbol XML. El elemento “`<xsl:choose>`” es muy similar a la instrucción “switch” de C/C++ o “case” de Pascal. Se pueden establecer tantas expresiones condicionales como se quieran mediante los elementos “`<xsl:when>`”. Si se quiere establecer una condición por defecto, el elemento a utilizar sería “`<xsl:otherwise>`”. La sintaxis quedaría como sigue:

```
<xsl:choose>
 <xsl:when test="expresion">
 ...
 </xsl:when>
 <xsl:when test="expresion">
 ...
 </xsl:when>
 <xsl:otherwise>
 ...
 </xsl:otherwise>
</xsl:choose>
```

Siguiendo con el ejemplo inicial, establezcamos una condición nueva en la que si el libro es menor de 12.50 euros muestre en color rojo todos sus datos. Si es mayor de 25.50 euros, en color verde. Y si no se cumple ninguna de las anteriores, en color azul. Esto se realizaría de la siguiente manera:

```
<xsl:for-each select="libreria/libro">
 <tr>
 <xsl:choose>
 <xsl:when test="precio < 12.50">
 <td bgcolor="#ff0000">
 <xsl:value-of select="titulo"/>
 </td>
 <td bgcolor="#ff0000">
 <xsl:value-of select="autor"/>
 </td>
 </xsl:when>
 <xsl:when test="precio > 25.50">
 <td bgcolor="#00ff00">
 <xsl:value-of select="titulo"/>
 </td>
 <td bgcolor="#00ff00">
 <xsl:value-of select="autor"/>
 </td>
 </xsl:when>
 <xsl:otherwise>
 <td bgcolor="#0000ff">
 <xsl:value-of select="titulo"/>
 </td>
 <td bgcolor="#0000ff">
 <xsl:value-of select="autor"/>
 </td>
 </xsl:otherwise>
 </xsl:choose>
 </tr>
</xsl:for-each>
```

## Mi biblioteca

Título	Autor
Physics-based animation	Kenny Erleben
Principios de seguridad informática para principiantes	Carlos García
Ejercicios complementarios de lógica digital	Alberto Sánchez

Figura 5.7. Cambio de estilo condicional con xsl:choose

## ACTIVIDADES 5.4



- Añadir al fichero XML inicial (ejemplo 5.1), dos libros de un nuevo autor. Sus precios serán 10.50 € y 60.50 € respectivamente
- Comprobar con el fichero XSL indicado que los nuevos libros se pintan del color indicado.
- Añadir las columnas precio al final e ISBN al principio de la tabla de resultados.
- Cambiar los colores para que si el coste es mayor que 25 €, las filas sean de color rojo y verde si es menor que 25 €.

## 5.4 LAS PLANTILLAS

### 5.4.1 ELEMENTO XSL:TEMPLATE

Hasta el momento solo hemos utilizado una única plantilla para todo el documento XML. Se habló del elemento “`<xsl:template>`” como etiqueta que permitía definir sobre qué parte del documento XML se quería actuar. Si se observan con detenimiento todos los ejemplos anteriores, el atributo “`match`” referenciaba siempre la raíz del documento XML, no estableciendo distinción entre otras ramas o elementos que lo componen. A continuación se muestra un cambio radical en el fichero XSL para que, usando distintas plantillas, podamos ver el nombre del autor en un guis claro y el título del libro en otro más oscuro (sin ningún elemento condicional por simplicidad del ejemplo):

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
 <html> <body>
 <h1>Ejemplo Plantillas</h1>
 <xsl:apply-templates/>
 </body> </html>
</xsl:template>

```

```

<xsl:template match="libreria">
 <h2>Mi biblioteca</h2>
 <table>
 <tr bgcolor="#887788">
 <th>Título</th> <th>Autor</th>
 </tr>
 <xsl:apply-templates select="libro"/>
 </table>
</xsl:template>

<xsl:template match="libro">
 <tr>
 <td><xsl:apply-templates select="titulo"/></td>
 <td><xsl:apply-templates select="autor"/></td>
 </tr>
</xsl:template>

<xsl:template match="titulo">
 <td bgcolor="#DDEEDD"><xsl:value-of select=". "/></td>
</xsl:template>

<xsl:template match="autor">
 <td bgcolor="#AABBAA"><xsl:value-of select=". "/></td>
</xsl:template>

</xsl:stylesheet>

```

## Ejemplo Plantillas

### Mi biblioteca

Título	Autor
Physics-based animation	Kenny Erleben
Principios de seguridad informática para usuarios	Carlos Garre
Ejercicios complementarios de lógica digital	Alberto Sánchez

*Figura 5.8. Aplicación de plantilla*

En este ejemplo se puede observar que se han establecido cinco plantillas personalizadas para los elementos “/”, “libreria”, “libro”, “titulo” y “autor”. Cuando se llame a esta hoja de estilos XSL desde un documento XML, se intentará reconocer los patrones de las plantillas, sustituyendo en la salida los contenidos que se indiquen por cada plantilla.

Se debe destacar una serie de elementos que no se han explicado anteriormente y que permiten la aplicación directa de la plantilla en el documento final:

- **<xsl:apply-templates />**: Indica que se apliquen el resto de las plantillas definidas en cuanto se cumplan el patrón “match” de cada una.
- **<xsl:apply-templates select="XXXX"/>**: Indica que se aplique en ese momento, el contenido de la plantilla “XXXX” definida en el documento XSL.

- `<xsl:value-of select="." />`: En el nodo XML actual de la plantilla actual, indica que se copie el valor que contenga ese nodo. Por ejemplo, si el nodo actual es de tipo “*título*”, insertará como resultado el título del libro actual.

El proceso seguido cuando se recorre un documento XML desde una transformación XSL, siempre es el mismo. Intuitivamente comienza a recorrerse el fichero XML desde su nodo raíz, aplicando en ese momento la plantilla asociada. En el ejemplo anterior se utilizará plantilla “/”, generando un esqueleto de documento HTML válido. El elemento “`<xsl:apply-templates />`” que está situado en la plantilla “/”, indica que se recorra todos y cada uno de los nodos definidos en el fichero XML aplicando las plantillas definidas. Si en ese recorrido del documento XML se encuentra el elemento “*librería*”, aplicará la plantilla del mismo nombre definida en el fichero XSL.

Es importante reseñar que cuando se aplica la plantilla librería, se establecen las propiedades y etiquetas HTML necesarias para crear una tabla de datos. Esta tabla contendrá datos en su interior por lo que el elemento “`<xsl:apply-templates select="libro" />`” indicará que, cuando se reconozca un nodo XML “libro”, se aplique la plantilla definida para ello. La misma explicación tienen la aplicación de las plantillas “*título*” y “*autor*”.

Finalmente, si en el recorrido del documento XML se encuentra sobre un nodo “*título*” o “*autor*”, se aplicará su plantilla correspondiente, con la salvedad que en esos nodos se requiere la extracción de la información para el documento final. Esta funcionalidad vendrá dada por el elemento “`<xsl:value-of select="." />`”, copiando el valor de ese nodo y enviándolo al documento HTML final.

Podemos ver, por tanto, el recorrido del documento XML como un conjunto de llamadas a “procedimientos” (plantillas), que según el caso, integrarán datos de salida en el documento final.

## ACTIVIDADES 5.5



- Insertar el dato número de páginas en cada libro contenido en el fichero XML inicial. Por ejemplo, el elemento será `<numPaginas>150</numPaginas>`.
- Añadir las columnas precio y número de páginas al final de la tabla de resultados.
- Añadir la columna ISBN al principio de la tabla de resultados.
- Añadir al fichero XSL una plantilla nueva para cambiar el estilo del elemento “*numPaginas*” para que se muestre en rojo si el libro tiene más de 150 páginas.
- Comprobar los resultados obtenidos comparándolo con los datos almacenados en el fichero XML.

## 5.5 CASO PRÁCTICO

Se propone crear un fichero XML nuevo que almacene datos de un videoclub. El videoclub almacenará los siguientes datos de una película:

- ✓ Título.
- ✓ Nacionalidad.
- ✓ Productor.
- ✓ Director.
- ✓ Año.
- ✓ Duración.
- ✓ Género.
- ✓ Sinopsis.
- ✓ Foto.
- ✓ URL.

Se deben crear al menos 20 películas distintas en las que algunas sean de la misma nacionalidad y del mismo director.

El valor del elemento foto será el nombre de un fichero .jpg o .png que deberá estar descargado en el mismo sitio donde está el fichero XML.

Se pide:

- Generar un fichero XSL (“p1.xsl”) en el que se muestre una tabla con todos los datos de las películas (salvo la foto) que el videoclub tiene almacenado.
- Generar un fichero XSL (“p2.xsl”) que muestre título, director y la foto (no el nombre del fichero sino la imagen en sí) de todas las películas almacenadas en el videoclub.
- Generar un fichero XSL (“p3.xsl”) igual que el anterior pero realizado con plantillas y que al hacer clic en la foto, nos lleve a la URL almacenada.



## RESUMEN DEL CAPÍTULO



En este capítulo se han explicado los conceptos básicos para la transformación de documentos XML mediante el lenguaje de transformación XSL. XSL es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio. Esto ha permitido saber qué elementos básicos son los que componen cualquier documento XSL (plantillas, elementos para recorrer los nodos, elementos para extraer la información, para ordenar los resultados, de tipo condicional, operadores, etc.).

También se ha proporcionado un conjunto de ejemplos y actividades pequeñas que permiten comprobar la funcionalidad de las transformaciones de manera fácil y sencilla. Se ha propuesto un caso práctico, de mayor complejidad, en el que se deben poner en práctica la práctica totalidad de las transformaciones mostradas anteriormente, fijando los conceptos como la generación de distintos documentos XSL para mostrar distintos resultados y la aplicación de las plantillas.



## EJERCICIOS PROPUESTOS

- 1. Busque por Internet cuál es la última recomendación del W3C de las transformaciones XSL.
- 2. Busque por Internet cuándo utilizar CSS y cuándo utilizar XSL (pista: la W3C responde esta pregunta).
- 3. Crear un documento XML que almacene una lista de CDs de música. Cada CD deberá almacenar la siguiente información:
  - Título del album.
  - Artista.
  - Títulos de las canciones con el tiempo por canción.
  - Sello discográfico.
  - Año de publicación.
- 4. Con el documento XML anterior, se pide:
  - Generar un fichero XSL (“cd\_p1.xsl”) en el que se muestre una tabla con todos los datos de los discos de música.
  - Elegir un artista cualquiera y generar un fichero XSL (“cd\_p2.xsl”) en el que se muestre una tabla con todas las canciones de ese artista.
  - Elegir un sello cualquiera y generar un fichero XSL (“cd\_p3.xsl”) en el que se muestre una tabla con todas las canciones de ese sello discográfico.
  - Elegir una duración máxima de canción y generar un fichero XSL (“cd\_p4.xsl”), en el que se muestre una tabla con todas las canciones que tienen una duración inferior a la elegida.

Al menos insertar 10 CDs.



## TEST DE CONOCIMIENTOS

### 1 ¿Qué es XSL?

- a) Es un lenguaje de estilos extensible para documentos HTML.
- b) Es un lenguaje de estilos extensible para documentos XML.
- c) Es un lenguaje de estilos extensible para documentos CSS.
- d) Ninguna de los anteriores.

### 2 XSL es a XML:

- a) Lo que las hojas de estilo en cascada (CSS) a HTML.
- b) Lo que las hojas de estilo en cascada (CSS) a XML.
- c) Lo que las hojas de estilo extensible (XSL) a HTML.
- d) Lo que las hojas de estilo extensible (XSL) a XML.

**3** Un fichero XSL debe:

- a) Guardarse en la misma ubicación que el fichero XML.
- b) Guardarse en la misma codificación que el fichero XML.
- c) Estar bien formado, pues es un fichero XML como otro cualquiera.
- d) Ninguna de las anteriores.

**4** El elemento “xsl:for-each”:

- a) Permite recorrerse todos los nodos indicados en su atributo “select”.
- b) Permite extraer la información de cada uno de los nodos que se recorran.
- c) Permite recorrerse los nodos dejándolos ordenados en todos los casos, según el criterio del programador.
- d) Ninguna de las anteriores.

**5** El elemento “xsl:value-of”:

- a) Permite recorrerse todos los nodos indicados en su atributo “select”.
- b) Permite extraer la información de cada uno de los nodos que se recorran.
- c) Permite ordenar los nodos según el criterio del programador.
- d) Ninguna de las anteriores.

**6** El elemento “xsl:sort”:

- a) Permite recorrerse todos los nodos indicados en su atributo “select”.
- b) Permite extraer la información de cada uno de los nodos que se recorran.
- c) Permite ordenar los nodos según el valor que indique su atributo "select".
- d) Ninguna de las anteriores.

**7** El elemento “xsl:if”:

- a) Permite establecer una condición basada en una expresión indicada por el programador. Esa condición está en el documento XSL.

**b)** Permite extraer la información de manera condicional.

**c)** Permite establecer una condición basada en una expresión indicada por el programador. Esa condición está en el documento XML.

**d)** Ninguna de las anteriores.

**8** El elemento “xsl:case”:

- a) Permite establecer una condición basada en una expresión indicada por el programador. Esa condición está en el documento XSL.
- b) Permite extraer la información de manera condicional.
- c) Permite establecer una condición basada en una expresión indicada por el programador. Esa condición está en el documento XML.
- d) Ninguna de las anteriores.

**9** El elemento “xsl:choose”:

- a) Permite condicionar los resultados, al igual que el elemento “xsl:if”.
- b) Permite extraer la información de manera condicional, al igual que el elemento “xsl:value-of”.
- c) Permite establecer una condición basada en una expresión indicada por el programador. Esa condición está en el documento XML.
- d) Ninguna de las anteriores.

**10** El elemento “xsl:template”:

- a) Solo puede haber un elemento de este tipo en un documento XSL.
- b) Solo puede haber un elemento de este tipo en un documento XML.
- c) Pueden usarse múltiples elementos de este tipo en un documento XSL, pero al menos uno tiene que referenciar al raíz del documento (“/”).
- d) Pueden usarse múltiples elementos de este tipo en un documento XSL y puede haber varios que refieran a la misma parte del documento, por ejemplo “/”.

# 6

# Almacenamiento de información

## OBJETIVOS DEL CAPÍTULO

- ✓ Buscar a través de Internet las necesidades de almacenamiento que tienen las grandes empresas tecnológicas.
- ✓ Descubrir, buscando qué es el método Fermi, cómo estimar cualquier necesidad de almacenamiento (sin datos suficientes).
- ✓ Conocer los tipos de bases de datos existentes.
- ✓ Descubrir cómo unificar un conjunto de bases de datos y administrarlos a través de los Sistemas de Gestión de Bases de Datos (SGBD).
- ✓ Mostrar la importancia de las transacciones y conocer que características deben implementar los SGBD para conseguirlas (ACID).
- ✓ Buscar y comparar distintos SGBD.
- ✓ Conocer qué es SQL y cómo XQuery permite realizar las mismas funciones pero en BD XML nativas.
- ✓ Convertir una BD relacional a un conjunto de documentos XML equivalentes.
- ✓ Utilización de Qizz Studio como motor de BD XML nativo.
- ✓ Utilizar expresiones FLWOR en XQuery para realizar consultas, inserciones, modificaciones y borrados de la BD. También para generar documentos XML o XHTML.
- ✓ Exportación de BD a ficheros XML y la importación en la BD de documentos XML.
- ✓ Conocer otras librerías y APIs que permiten tratar los documentos XML a través de lenguajes de programación como Java o C/C++.

Eric Schmidt, CEO de Google hasta enero del 2011, realizó la siguiente afirmación en la conferencia de Lake Tahoe en California (EE.UU.):

*“Cada dos días creamos tanta información como la que se generó desde los albores de la humanidad hasta el 2003”.*



**Figura 6.1.** Eric Schmidt, antiguo CEO de Google



Para obtener más información, vea las páginas <http://www.isidoroporquicho.com/eric-schmidt-every-2-days-we-create-as-much-i> y <http://techcrunch.com/2010/08/04/schmidt-data/>.

Aún cometiendo algún error numérico en la estimación de Schmidt, parece claro que de manera constante y diaria producimos ingentes cantidades de información que debemos almacenar y procesar. Esta afirmación es tremadamente reveladora en la era de la información en la que vivimos, pero:

- ✓ ¿Cómo es posible que podamos generar toda esa información en tan poco tiempo?
- ✓ ¿Cómo es posible que además podamos almacenarla?

Un ejemplo de generación masiva de información puede ser el acelerador de partículas llamado Gran Colisionador de Hadrones. Éste se encuentra ubicado en la Organización Europea para la Investigación Nuclear (CERN) de Ginebra-Suiza. La red Física-Matemática-Informática que se ha desplegado permitirá capturar características, velocidades y trayectorias de las partículas subatómicas que se conocen e incluso descubrir nuevas partículas. Para ello han desarrollado un sistema llamado *Grid-Computing* en el que se manejarán la enorme cantidad de datos producidos por el acelerador. Estiman que generarán en torno a 40 Terabytes de datos por día, enviándose copias de los datos a todas las instituciones académicas de índole mundial que trabajan en el proyecto. Llegan a afirmar que ellos solos podrán generar 15 petabytes al año (y el tamaño sigue creciendo).

Sin duda las novedades que se presentan en los congresos tecnológicos sobre las tecnologías de la información, almacenamiento y procesamiento hacen pensar que no solo podemos generar y almacenar la información, sino también tratarla y realizar operaciones con ella. Pero debe existir alguna manera de seleccionar, catalogar, ordenar la información relevante y desechar aquella que esté duplicada o sea superflua, sin que el rendimiento en el acceso penalice los tiempos. Ese sistema se llama Sistema de Gestión de Bases de Datos (SGBD).



## ¿SABÍAS QUE...?

El CERN fue fundado en 1954 y está financiado por 20 estados miembros de la Unión Europea. Para el período 2011-2015 tiene un presupuesto de unos 3800 millones de euros. La mayor contribución del CERN a la humanidad fue la creación del protocolo *World Wide Web* (www), creada por Tim Berners-Lee para acceder a la información del centro a través de los sistemas de comunicaciones internos.



## ¿SABÍAS QUE...?

Google y Microsoft utilizan, entre otras muchas, una técnica de selección de personal llamada "Cuestiones de Fermi". Con esta técnica se permite medir las capacidades de abstracción de los aspirantes ante problemas complejos por la insuficiencia de información suficiente.

### ACTIVIDADES 6.1



- Un Kilobit son  $2^{10}$  bits. Un Megabit son  $2^{20}$  bits. Un Gigabit son  $2^{30}$  bits. ¿Qué nombres tienen las siguientes capacidades?
- $2^{40}$  bits
  - $2^{50}$  bits
  - $2^{60}$  bits



Para obtener más información, vea la página <http://es.wikipedia.org/wiki/Petabyte>.

- Buscar por Internet cuanta información está actualmente almacenada en los servidores de Google. ¿Y en los servidores de Facebook/Tuenti?
- ¿Qué es un problema de Fermi? Buscar por Internet dos ejemplos en los que se utilice esta metodología.



Para obtener más información, vea la página [http://es.wikipedia.org/wiki/Problema\\_de\\_Fermi](http://es.wikipedia.org/wiki/Problema_de_Fermi).

- ¿Cómo puede ayudar una estimación de Fermi para almacenar datos en un SGBD?
- Estimar, con el método de Fermi, qué tamaño de base de datos deberíamos tener para almacenar en imágenes toda la vida de una persona en España.

## 6.1 SISTEMAS DE ALMACENAMIENTO DE LA INFORMACIÓN

Una Base de Datos (BD) es una herramienta que permite organizar los datos que tienen algún tipo de relación entre sí, de manera que son almacenados y utilizados a través de un interfaz de comunicación que facilita su gestión dentro de una entidad u organización. Existen muchas maneras de clasificar los distintos tipos de bases de datos pero podemos resumirlo según la variabilidad de los datos que se almacenen en ellas:

- **Datos estáticos:** Datos que nunca cambiaran en el tiempo (solo lectura).
- **Datos dinámicos:** Datos que pueden modificarse de cualquier manera, a lo largo del tiempo.

Salvo contadas excepciones (datos históricos, estadísticas, análisis clínicos que nunca cambiarán desde que se tomaron), las bases de datos que se utilizan hoy en día son dinámicas. Las bases de datos estáticas suelen utilizarse cuando se requiere un alto rendimiento en las consultas, quedando muy penalizadas si tuviera que actualizarse algún dato posteriormente.

Los datos que se almacenan en ellas, deben seguir un modelo lógico adecuado para cumplir su función de manera eficiente. Es decir, de alguna manera hay que indicar a la base de datos qué cosas quiere almacenar, enviando una **descripción** de los datos de interés. Solo así será posible tener un modelo de datos adecuado al problema que se quiera dar solución. Dependiendo de cómo se haya hecho esta descripción de los datos, así cambiarán los procedimientos de acceso, almacenaje y recuperación de los contenidos. Por tanto, dependiendo del modelo de datos utilizado podemos distinguir:

- **BD Jerárquica:** Utilizan árboles para almacenar la información (nodo raíz, nodos padre/hijos, nodos hoja), de manera que la extracción de los datos se realiza mediante un recorrido del árbol y eligiendo la rama por la que debe continuar su búsqueda. Este tipo de estructuras son muy eficientes en las búsquedas pero no pueden reflejar eficientemente la redundancia de datos.
- **BD de Red:** Muy parecida a las jerárquicas pero donde un nodo puede tener varios padres simultáneamente (en contraposición de las bases de datos jerárquicas).
- **BD Relacional:** Es una base de datos que permite resolver el problema que tenían las bases de datos jerárquicas (las redundancias). Mientras que las jerárquicas la posición del dato en el árbol era importante, en las relacionales es irrelevante. Un conjunto de datos compacto y con sentido propio se almacena en algo llamado **tupla**. Una tupla no es más que una fila de una tabla en la que se almacena dicha información compacta. El conjunto de tuplas con información semántica igual determina una tabla. Y un conjunto de tablas pueden ser relacionadas con otras tablas (o tuplas), mediante tablas intermedias que reflejan las relaciones entre cada una de ellas. Esto es lo que soluciona la redundancia de información y son las más utilizadas actualmente.
- **BD Transaccional:** Utilizan el concepto de transacción (que veremos más adelante) que permite asegurar una serie de características importantes en los sistemas informáticos de hoy en día. Estas características se incluyeron en la mayoría de las bases de datos relacionales de hoy en día.
- **BD Multidimensional:** A efectos sería una base de datos Relacional en el que en vez de tener tablas (2D), tendríamos estructuras con N dimensiones para almacenar la información (cubos si fuese 3D).
- **BD Orientadas a Objetos:** Difiere de las bases de datos relacionales en que no se almacena la información por tuplas sino por objetos que contienen la información y los métodos que son necesarios para tratarlas. Es llevar la programación orientada a objetos al mundo de las bases de datos generando características como la herencia, la encapsulación o el polimorfismo.

Como puede observarse, dependiendo del modelo de datos que se quiera almacenar, se deberá utilizar una base de datos u otra.

Independientemente del modelo usado, cuando se tiene un gran conjunto de bases de datos, lo común es utilizar un Sistema de Gestión de Bases de Datos o SGBD, que unifiquen las bases de datos entre sí y se mejoren los flujos de distribución de la información en diferentes entornos y/o formatos.



Para obtener más información, vea la página <http://www.maestrosdelweb.com/principiantes/que-son-las-bases-de-datos/>.

Básicamente se deja de utilizar los almacenes de información de las aplicaciones (los ficheros) para utilizar un interfaz de comunicación único (el SGBD y una BD concreta). Su utilización permite:

- ✓ Evitar la redundancia de datos que puede producirse por la duplicación de los mismos ficheros de datos utilizados en distintas aplicaciones y/o ubicaciones.
- ✓ Permiten la abstracción de los datos independientemente del sistema hardware/software utilizado y del soporte utilizado para su almacenamiento.
- ✓ Evitar la inconsistencia de los datos cuando diferentes programas intentan actualizar un dato que es compartido.
- ✓ Evitar que el cambio o adición de nueva información no planeada en el análisis inicial del sistema obligue a la adaptación de todos los programas informáticos que utilizan esos ficheros.
- ✓ Evitar que la mala programación en el acceso/modificación/borrado de los datos en un fichero genere problemas de seguridad. Tener una capa de aplicación y una capa de datos distinta facilita la depuración y la trazabilidad de los programas.

La abstracción, independencia, consistencia, seguridad y accesibilidad son los objetivos que se buscan en estos sistemas. Además su funcionamiento suele ser muy simple. Se puede resumir en un comentario muy acertado de D. Ángel García Moreno, catedrático de la Universidad Politécnica de Madrid y profesor de la asignatura de Bases de Datos en la Ingeniería Informática:

*“Mientras que con los métodos clásicos se accedía a la información a través de ficheros y los datos paseaban por los programas; en los SGBD son los programas los que se dan una vuelta por las Bases de Datos para conseguir lo que necesitan”.*

Estos beneficios hacen que la práctica totalidad de los sistemas que quieran almacenar información recurran a un sistema de bases de datos acorde a sus necesidades, garantizando:

- ✓ Gestionar la redundancia para evitarla o al menos limitarla.
- ✓ Mantener los datos separados de las aplicaciones que lo usan.
- ✓ Localizar los datos en una ubicación desde la cual, las aplicaciones puedan acceder de manera distribuida, concurrente y asegurando la integridad de los mismos.
- ✓ Permite realizar una correcta política de copias de seguridad, ya que ya no es necesario ir copiando los ficheros en distintas ubicaciones.

En un SGBD no todo son ventajas. Existen una serie de inconvenientes que es necesario resaltar para tomar en el futuro una decisión adecuada. Como inconvenientes se pueden enumerar los siguientes:

- ✓ Un SGBD no es fácilmente administrable. Requiere de personal especializado en administración de sistemas y lenguajes de consultas contra las bases de datos. Una buena administración de los sistemas y de las bases de datos puede salvar la continuidad del negocio.
- ✓ Si los datos son simples y no requiere de acceso concurrente de usuarios o aplicaciones, puede ser sustituido por un simple fichero, como por ejemplo un archivo binario o un fichero XML.
- ✓ No es sencillo formar al personal que trabaja con los datos almacenados en un SGBD si no tiene un perfil orientado hacia la informática. El personal de nóminas de una empresa, por ejemplo, no tiene porqué aprender un lenguaje de programación de consultas. Requerirá que el sistema le proporcione una interfaz de comunicación tipo formulario de consultas que oculte la complejidad del sistema.
- ✓ Poner en funcionamiento un SGBD requiere de un coste inicial en hardware y software que puede no ser necesario para las necesidades normales de un entorno pequeño.

Siempre habrá que realizar un análisis de los beneficios y los inconvenientes que aportan la integración de estos SGBD en un entorno de trabajo concreto.

Un requisito que hasta el momento no se ha nombrado y que por exigencias del mercado se ha añadido en los SGBD modernos es el soporte transaccional. Una **transacción** es un conjunto de órdenes en secuencia que, en su conjunto, determina un trabajo completo. El trabajo es completo si y solo si se cumplen todas y cada una de sus operaciones en el orden dado. Tras la ejecución de una transacción solo existen dos resultados posibles:

- ✓ Finalizada con éxito.
- ✓ No completada (y asegura que no se ha cambiado nada).

Veamos un ejemplo de cómo de importante es una transacción. Supongamos que un usuario quiere sacar dinero de un cajero automático. La transacción es “*sacar dinero*”. Para ello se siguen una serie de pasos secuenciales que permiten realizar la transacción con éxito:

1. Validación del usuario (inserción del PIN).
2. Selección del importe a sacar.
3. Anotación en la cuenta bancaria del reintegro.
4. Expedir tique y el dinero solicitado al usuario.

Si está en el paso número 3 y después hubiera un corte en el suministro eléctrico, el usuario tendría anotada en su cuenta bancaria un reintegro que nunca ha sido entregado al usuario (paso 4). Obviamente el usuario se enfadaría y reclamaría.

Una transacción, si es ejecutada de manera exitosa, asegura que los pasos del 1 al 4 se han seguido en ese orden y que todo ha salido bien. Si la transacción no ha sido exitosa, asegura que ni el dinero ha salido por el cajero, ni que tampoco se ha anotado el reintegro en la cuenta bancaria del usuario (incluso aunque haya habido un corte en el flujo eléctrico y estuviese la transacción en el paso 3). Es decir, al no ser exitosa la transacción es como si nunca hubiese pasado.

Esto que, explicado con un ejemplo es tan simple, su nombre técnico en un SGBD se denomina características **ACID**, acrónimo de:

- **Atomicidad (Atomicity)**: Es la propiedad que asegura que una transacción no se ha quedado a medias. O se ha ejecutado completamente o ninguna de las acciones intermedias ha sido llevada a cabo en la base de datos.
- **Consistencia (Consistency)**: Una vez se ha determinado que la transacción ha sido exitosa, debe quedar reflejados sus resultados en la base de datos dejando totalmente consistente e integra la base de datos.
- **Aislamiento (Isolation)**: Una transacción no puede afectar a otra en el transcurso de su ejecución. Esto implica que dos transacciones que trabajen sobre el mismo conjunto de información no puede generar información inconsistente o error alguno en el sistema.
- **Durabilidad (Durability)**: Una vez realizada la transacción, sus resultados permanecerán inalterables, independientemente de si surgen problemas en el sistema. Esto no quiere decir que una transacción posterior no pueda modificar los datos anteriormente modificados (sí podría pero no de manera concurrente con otra transacción).

Hasta el momento no se ha hablado de cómo está definido un SGBD, por lo que se podría pensar que es un único bloque funcional, hecho *ad hoc*, para resolver un problema de almacenamiento de datos concreto. En realidad un SGBD puede verse como un conjunto de actores y componentes que se intercomunican entre sí para hacer más fácil el acceso a los datos. Estos componentes son:

- **El Hardware**: Dispositivos físicos donde residen todos los demás componentes del SGBD. Normalmente son dispositivos con capacidades de redundancia y con alta disponibilidad para soportar cualquier contingencia que surja durante su funcionamiento.
- **El Software**: Aplicación que permite abstraerse de las características físicas del hardware y que permite hacer al SGBD ser independiente de la plataforma hardware sobre la que se ejecuta.
- **Los Datos**: Información almacenada dentro del recinto físico (hardware) y administrada por el software.
- **Los Usuarios**: Actores a los que se les permite interactuar con el SGBD, haciéndoles transparente el acceso a los datos, independientemente del Hardware o Software utilizado.

En relación a los usuarios, dependerá del software SGBD que se vaya a utilizar pero genéricamente podemos identificar los siguientes roles:

- Administrador del Sistema informático.
- Administrador de la Base de datos y Consultas.
- Usuario Final.

Los anteriores componentes se unen entre sí haciendo que el sistema sea modular y fácil de usar/administrar.

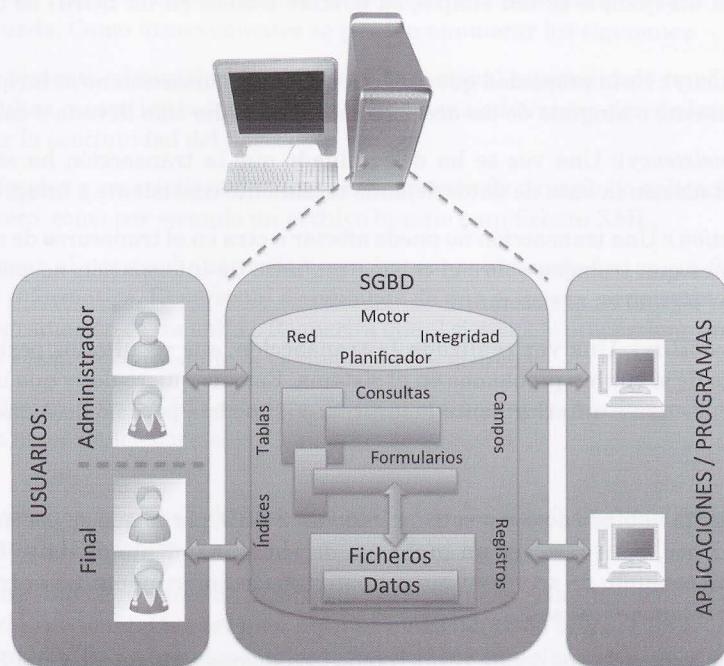


Figura 6.2. Diagrama de un SGBD

Únicamente queda por hablar del sistema hardware que se debe utilizar para llevar a cabo la instalación de un SGBD. Para pequeñas y medianas empresas cualquier servidor relativamente moderno puede servir. Todo dependerá del grado de concurrencia de accesos y la cantidad de datos a almacenar los que determinarán las características físicas del hardware. Pero lo que debe ser fundamental es el dimensionamiento del almacén de datos para que no se quede pequeño. Desde la máquina analítica de Charles Babbage hasta los netbook de hoy en día, el almacenamiento de la información ha pasado por muchas etapas:

- El cableado físico.
- La tarjeta perforada.
- La cinta magnética.
- El disco duro.
- Los disquetes.
- El CD-ROM / DVD.
- Las tarjetas de memoria.
- Los pendrives.
- Los discos duros en estado sólido.

Es claro que los discos duros actuales están alcanzando tamaños impensables, lo que el almacenamiento de datos puede no ser un problema. Simplemente sustituyendo un disco por otro de mayores dimensiones el problema se resuelve y además es seguro que será más rápido que el antiguo (lo que aceleran las transferencias de información).



## ¿SABÍAS QUE...?

Aunque la información actualmente se almacena de manera digital y se puede acceder a ella de manera muy simple, los sistemas de bases de datos tradicionales no permiten hacer un análisis profundo de las entidades complejas. Es por ello que surge el concepto de las bases del conocimiento (Knowledge Base) que permiten tener conciencia de lo que la base de datos sabe de sí misma a través de relaciones semánticas, ¿será el principio de la inteligencia artificial en las BD?

## ACTIVIDADES 6.2



- ▶ Estimar, por el método de Fermi, qué tamaño tendrían los datos de la base de datos de un videoclub. Hay que tener en cuenta el espacio requerido para los clientes y para el almacenamiento de los datos de las películas disponibles.
- ▶ Buscar en Internet qué es SQL. ¿Para qué sirve?
- ▶ Buscar en Internet qué es un **índice** en una BD. ¿Para qué sirve?
- ▶ Buscar en Internet y comparar en un cuadro de texto los siguientes tipos de Sistemas Gestores de Bases de Datos (al menos tres de ellos entre sí):
  - a. Oracle.
  - b. Microsoft SQL Server.
  - c. MySQL.
  - d. PostgreSQL.
  - e. Interbase.
  - f. SapDB.
  - g. SQLite.



Para obtener más información, vea la página [http://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n\\_de\\_sistemas\\_administradores\\_de\\_bases\\_de\\_datos\\_relacionales](http://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n_de_sistemas_administradores_de_bases_de_datos_relacionales).

## 6.2 UTILIZACIÓN DE XML PARA EL ALMACENAMIENTO DE LA INFORMACIÓN

Gran parte de las BD que hay hoy en día están basadas en un modelo de datos (también llamado modelo de entidad-relación). Es un modelo que ha funcionado bien durante mucho tiempo y que todavía seguirá funcionando. Si bien es cierto que las BD orientadas a objetos permiten simplificar el trabajo de integración con los lenguajes de programación orientados a objetos, cambiar un modelo de trabajo o un sistema que funciona con el modelo anterior es arriesgado. Algunos SGBD tienen modelos híbridos que permiten añadir extensiones al modelo relacional y avanzar a un modelo orientado a objetos de manera poco traumática y de manera transparente.

Tras la llegada de la era Internet, la compartición de la información empezó a resultar crucial para no quedarse desfasado y mejorar las relaciones comerciales. Por ejemplo, una empresa *X* necesita enviarle a la empresa *Y* información en relación a sus transacciones comerciales. El envío de esta información beneficia a *X* y también a *Y* (reducción de costes económicos, costes burocráticos, tiempo, etc.). Es lo que se denomina **B2B** (*Business To Business*). El problema es que *X* utiliza unos modelos de datos distintos a los de *Y*. Tampoco es fácil ponerse deacuerdo con el SGBD. Debe haber algo que permita que ambos sistemas puedan comunicarse entre sí, siendo distintos. La solución es XML.



Para obtener más información, vea la página [http://mymadcat.com/~madcat/talk\\_tracker.pdf](http://mymadcat.com/~madcat/talk_tracker.pdf).

XML permite definir de manera rápida e intuitiva una representación de la información que ambas empresas desean compartir. La empresa *X* usará su SGBD para exportar sus datos a XML y se los remitirá a *Y*. Ambas empresas conocen esa representación de la información por lo que la información fluirá sin problema (independientemente de los campos que tengan sus respectivas BDs o de los SGBD que estén utilizando).

Los SGBD actuales (sobre todo los que usan modelos relacionales), proporcionan en algunos casos extensiones que permitan trabajar con los modelos y representaciones definidas en documentos XML. Aún así, si el objetivo es utilizar XML desde el principio, se debería analizar las siguientes BD XML nativas:



Para obtener más información, vea la página <http://www.info-ab.uclm.es/asignaturas/300219/pdf/BBDDXML.pdf>.

- eXcelon XIS Lite<sup>12</sup>.
- TEXTML<sup>13</sup>.
- dbXML<sup>14</sup>.
- eXist<sup>15</sup>.

12 <http://xml.coverpages.org/ExcelonXIS-Lite.html>

13 <http://www.ixiasoft.com>

14 <http://sourceforge.net/projects/dbxml-core/>

15 <http://exist.sourceforge.net/>

Las dos primeras son comerciales y las dos últimas son del mundo OpenSource. Cuando se habla de BD XML nativas, se ha de dejar claro que existen dos maneras almacenar información dentro de ellas:

- **Usando un modelo centrado en el almacenamiento de los datos:** exactamente igual que las BD Relacionales (se guardan tuplas).
- **Usando un modelo centrado en el documento:** no hay campos, ni datos, tal y como se conoce en las BD Relacionales. Se guardan documentos XML.

La primera permite seguir utilizando los modelos relacionales dentro de BD XML. La segunda permite almacenar documentación de diferentes modelos dentro de la BD. Dependiendo de los objetivos de almacenamiento que se planteen quizás se ajuste más un modelo que otro. Ese análisis debe ser meditado concienzudamente.

Dado que el modelo relacional es el más utilizado hoy en día, se indicarán una serie de pasos para usando XML se pueda utilizar el mismo modelo de datos.



Para obtener más información, vea la página <http://www.di.uniovi.es/~labra/cursos/ver06/pres/XMLBD.pdf>.

### 6.2.1 BASES DE DATOS RELACIONALES

Cuando se utiliza un modelo centrado en el almacenamiento de los datos, la referencia son las BD relacionales. Todo se basa en un conjunto de tablas bidimensionales que permiten almacenar los datos del mundo real.

Para facilitar la comprensión, imaginemos que se quiere representar un conjunto de libros de una biblioteca. Para conseguir almacenar toda la información relativa a cada libro necesitaremos una tabla que almacene atributos como “Título”, “Autor”, “Editorial”, “Edición”, “ISBN” y “NumPáginas”. Esta tabla se llamaría “Libros”. Si se quisiera añadir un libro, se generaría una nueva entrada en la tabla (fila o tupla), en la que se completarán los atributos de ese libro. Si se quisiera añadir un segundo libro a almacenar, se añadiría otra tupla a la tabla. Y así sucesivamente.

TABLA LIBROS

Cód_Libro	Título	Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	Miguel de Cervantes Saavedra	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	Fernando de Rojas	Maxtor	1	9788471664938	320
3	Leyendas	Gustavo Adolfo Bécquer	Cátedra	21	9788437620244	416

Figura 6.3. Tabla “Libros”

Antes de continuar añadiendo libros, podríamos darnos cuenta que existen una serie de atributos que podrían ser compartidos entre muchos libros. Sería el caso de “Autor” (un autor puede escribir muchos libros), o “Editorial” (una editorial podría publicar muchos libros de muchos autores distintos), etc. Si por cada libro tuviéramos que llenar esa misma información, la tabla “Libros” tendría muchísima información redundante. Anteriormente se comentó que el modelo de datos de las BD relacionales hacían una gestión de la redundancia muy eficaz. ¿Cómo resuelve este caso?

Sin duda, si hay información redundante, toda esa información debe salir de la tabla “Libros” y ubicarse en una nueva tabla. Para el caso del atributo “Autor”, se podría crear una nueva tabla llamada “Autores” en la que almacenaríamos la información relativa a ellos más un código que le represente de manera única (“CódigoAutor”, “Nombre”, “Apellidos”, “FechaNacimiento”, etc.). Con ese código de autor, cada vez que se añade un nuevo libro de un autor ya añadido, únicamente haremos referencia en el campo “Autor” del nuevo libro al código que referencia a dicho autor (en la tabla “Autores”). Esta manera de indireccionar información a otras tablas se denomina **relación**. De la misma manera se haría con el campo “Editorial”.

TABLA LIBROS

Cód_Libro	Titulo	Cód_Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	1	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	2	Maxtor	1	9788471664938	320
3	Leyendas	3	Cátedra	21	9788437620244	416

Figura 6.4. Tabla “Libros” con relación con “Autores”

TABLA AUTORES

Cód_Autor	Nombre	Apellidos	Fecha Nacimiento
1	Miguel	de Cervantes Saavedra	29/09/1547
2	Fernando	de Rojas	01/01/1470
3	Gustavo	Adolfo Bécquer	17/02/1836

Figura 6.5. Tabla “Autores”

### 6.2.2 TRANSFORMACIÓN A XML

En principio la adaptación del modelo de datos de una BD relacional a XML es relativamente sencillo de realizar. Una vez obtenido el modelo relacional, como el explicado en el punto anterior, se podría realizar una transformación de tablas a un documento XML simplemente creando una DTD y creando un documento XML bien formado. A continuación se detallarán los pasos a realizar para una tabla de libros y de autores como la anterior:

**1** Cada tabla del modelo relacional será un elemento dentro del DTD:

```
<!DOCTYPE Libros[
 <!ELEMENT Libros (libro)*>
]>
```

En este caso se tiene una tabla llamada “Libros” que almacena tuplas de tipo “libro” (cero o más libros).

**2** Cada tupla de la tabla se llama “libro”. Es necesario indicar qué campos componen a la tupla:

```
<!ELEMENT libro (Cod_Libro, Titulo, Autores, Editorial, Edicion, ISBN, NumPaginas)>
```

**3** Cada columna de la tabla deberá establecerse como un tipo de dato almacenable (*char*, *integer*, etc.):

```
<!ELEMENT Cod_Libro(#PCDATA)>
<!ELEMENT Titulo (#PCDATA)>
<!ELEMENT Editorial (#PCDATA)>
<!ELEMENT Edicion (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT NumPaginas (#PCDATA)>
```

**4** Si existe una columna compleja (como puede ser la de “Autores”), se debe crear un nuevo elemento similar al del paso 2. En este caso un libro puede tener uno o más autores (por eso el ‘+’). También se definen los tipos de datos a almacenar:

```
<!ELEMENT Autores (autor)+>
<!ELEMENT autor (Cod_Autor, Nombre, Apellidos, FechaNacimiento)>
<!ELEMENT Cod_Autor (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Apellidos (#PCDATA)>
<!ELEMENT FechaNacimiento (#PCDATA)>
```

**5** Si dos tablas están relacionadas entre sí a través de una tercera tabla de relación, se establece un nuevo elemento como en el paso 2 y se continua con los pasos siguientes.

La DTD que resulta de todo esto será similar a ésta:

```
<!DOCTYPE Libros[
 <!ELEMENT Libros (libro)*>
 <!ELEMENT libro (Cod_Libro, Titulo, Autores, Editorial,
 Edicion, ISBN, NumPaginas)>
 <!ELEMENT Cod_Libro(#PCDATA)>
 <!ELEMENT Titulo (#PCDATA)>
 <!ELEMENT Editorial (#PCDATA)>
 <!ELEMENT Edicion (#PCDATA)>
 <!ELEMENT ISBN (#PCDATA)>
 <!ELEMENT NumPaginas (#PCDATA)>

 <!ELEMENT Autores (autor)+>
 <!ELEMENT autor (Cod_Autor, Nombre, Apellidos,
 FechaNacimiento)>
 <!ELEMENT Cod_Autor (#PCDATA)>
 <!ELEMENT Nombre (#PCDATA)>
 <!ELEMENT Apellidos (#PCDATA)>
 <!ELEMENT FechaNacimiento (#PCDATA)>
]>
```

Y un ejemplo de documento XML ajustado a la anterior DTD puede ser el siguiente:

```
<?xml version="1.0">
<Libros>
 <libro>
 <Cod_Libro>1</Cod_Libro>
 <Titulo>Don Quijote de la Mancha</Titulo>
 <Editorial>Juan de la Cuesta</Editorial>
```

```
<Edicion>3</Edicion>
<ISBN>9788466745840</ISBN>
<NumPaginas>176</NumPaginas>
<Autores>
 <autor>
 <Cod_Autor>1</Cod_Autor>
 <Nombre>Miguel</Nombre>
 <Apellidos>de Cervantes Saavedra</Apellidos>
 <FechaNacimiento>29/09/1547</FechaNacimiento>
 </autor>
</Autores>
</libro>
<libro>
 <Cod_Libro>2</Cod_Libro>
 <Titulo>La Celestina</Titulo>
 <Editorial>Maxtor</Editorial>
 <Edicion>1</Edicion>
 <ISBN>9788471664938</ISBN>
 <NumPaginas>320</NumPaginas>
 <Autores>
 <autor>
 <Cod_Autor>2</Cod_Autor>
 <Nombre>Fernando</Nombre>
 <Apellidos>de Rojas</Apellidos>
 <FechaNacimiento>01/01/1470</FechaNacimiento>
 </autor>
 </Autores>
</libro>
</Libros>
```

## 6.3 LENGUAJES DE CONSULTA Y MANIPULACIÓN

Tras instalar un SGBD, los usuarios se comunicarán con él mediante algún tipo de lenguaje que permita manipular los datos almacenados. El lenguaje de consulta estructurado más popular es SQL (*Structure Query Language*). Este lenguaje permite, de forma declarativa, acceder a las BD relacionales y operar con ellas. Operaciones típicas son:

- ✓ Creación y borrado de tablas.
- ✓ Inserción, modificación y borrado de tuplas.
- ✓ Ejecución de búsquedas mediante consultas.

SQL es el lenguaje que en la actualidad se considera estándar de facto pues la inmensa mayoría de los SGBD lo implementan. Existen numerosas revisiones del lenguaje (SQL-86, SQL-92, SQL-2003, SQL-2006, SQL-2009) pero la mayoría de los SGBD parten del estándar establecido en 1992. Cabe destacar lo siguiente (en lo relativo a XML se refiere):

- ✓ El estándar 2003 establece ciertas características que permiten un soporte inicial a documentos XML.
- ✓ El estándar 2006 establece una mayor integración con los documentos XML, permitiendo importar y exportar datos en XML. Se establece XQuery como lenguaje de consulta para colecciones de datos en formato XML (y también SQL), aprobado por el W3C (World Wide Web Consortium).

Si, en definitiva, se va a trabajar con documentos XML para almacenar en su interior información mediante un modelo relacional, se necesitará algún lenguaje que permita extraer y manipular información de igual manera que SQL con las BD relacionales. Este lenguaje se llama **XQuery**.

XQuery es, por tanto, un lenguaje de consulta similar a SQL que permite recorrer los documentos XML de manera que se pueda extraer y manipular la información contenida en el mismo. Es un lenguaje muy sencillo que no requiere de conocimientos de programación avanzados (no es necesario saber C/C++, Java, Python, Visual Basic Script, etc). Además XQuery es compatible con muchas de las tecnologías que estandariza W3C (como XML, Namespaces, los esquemas, XSLT y XPath). Es por ello, por su facilidad para obtener resultados sin casi programar y por la compatibilidad con muchos de los estándares actuales, que se ha elegido para realizar las consultas dentro de este libro.

Cuando se va a analizar un documento XML, se crea un árbol de nodos del mismo. Ese árbol tiene un elemento raíz y una serie de hijos. Los hijos del nodo raíz pueden tener más hijos. Si repetimos ese proceso llegará un momento en el que el último nodo no tiene ningún hijo, lo que se denomina nodo hoja.

Establecido ese árbol de nodos, se recorre esa representación del documento XML buscando la información que se quiere (ya sea algún nodo concreto, algún atributo de un nodo concreto, etc.). ¿Qué tipos de nodos se puede encontrar en ese recorrido? Básicamente los siguientes:

- **Nodo raíz o “/”**: Es el primer nodo del documento XML. En el ejemplo de la biblioteca de libros explicado anteriormente, sería el elemento “Libros”.
- **Nodo elemento**: Cualquier elemento de un documento XML es un nodo elemento en el árbol. El nodo raíz es un caso especial de Nodo elemento (no tiene padre). Cada nodo elemento posee un parent y puede o no poseer hijos. En el caso que no tenga hijos, sería un nodo hoja. En el ejemplo de la biblioteca de libros explicado anteriormente
- **Nodo texto**: Cualquier elemento del documento que no esté marcado con una etiqueta de la DTD del documento XML.
- **Nodo atributo**: Un nodo elemento puede tener etiquetas que complementen la información de ese elemento. Eso sería un nodo atributo.

La extracción de la información durante el recorrido del árbol será tan simple como la detección de los nodos a buscar y el procesamiento de la información que se quiere extraer de ese nodo concreto. Esto que parece tan complicado, se realiza fácilmente con una tecnología denominada **XPath** (XML Path).

XPath es la herramienta que utiliza XQuery para procesar el árbol de nodos de un documento XML. Funciona en base a una serie de expresiones que permiten identificar qué parte del documento XML se quiere acceder o recorrer. La gran ventaja es que al ser una herramienta bastante genérica, XPath no solo es el motor de acceso en documentos XML para XQuery sino que es la base para otras tecnologías como XPointer, XLink o XSLT (como hemos visto anteriormente). Por tanto XQuery usa a XPath para hacer su trabajo.

Pero lo realmente interesante es cómo hacer consultas con XQuery.



## ¿SABÍAS QUE...?

Los orígenes de SQL están muy relacionados con el desarrollo de las bases de datos relacionales iniciadas en 1970 por Raymond Boyce. Inicialmente se llamaba **SEQUEL** (Structured English Query Language) y fue diseñado para manipular y extraer datos en el sistema de base de datos diseñado por IBM.

## ACTIVIDADES 6.3



➤ Buscar por Internet que estándar SQL cumplen los siguientes SGBD:

- Oracle.
- Microsoft SQL Server.
- MySQL.
- PostgreSQL.
- Interbase.
- SapDB.
- SQLite.

➤ Buscar por Internet qué es PL/SQL. ¿Para qué sirve?



Para obtener más información, vea la página <http://en.wikipedia.org/wiki/PL/SQL>.

### 6.3.1 HERRAMIENTA QIZX STUDIO

Qizx Studio<sup>16</sup> es un motor de BD XML que permite el almacenamiento de los documentos XML y la realización de búsquedas y transformaciones a gran velocidad. Qizx es un motor bastante versátil que en la que destacan las siguientes especificaciones técnicas:



Para obtener más información sobre Qizx Studio, vea las páginas <http://kiwi.emse.fr/DN/qizx-manual.pdf> y [http://www.xmlmind.com/qizx/\\_distrib/docs/manual.pdf](http://www.xmlmind.com/qizx/_distrib/docs/manual.pdf).

<sup>16</sup> <http://www.xmlmind.com/qizx/product.html>



Para obtener más información sobre las especificaciones técnicas de Qizx Studio, vea la página <http://www.xmlmind.com/qizx/features.html>.

- ✓ Está orientado a la consulta.
- ✓ No necesita de DTD o esquemas predefinidos, siempre y cuando el documento XML esté bien formado.
- ✓ Realiza un indexado automático lo que permite acelerar las consultas. También puede personalizarse el indexado si se quiere optimizar aún el rendimiento.
- ✓ Permite tener una colección jerárquica de documentos XML (también llamado “XML Library”).
- ✓ El almacenamiento de la información se realiza directamente en XML nativo.
- ✓ Soporta XQuery/XPath 2, siendo totalmente compatible con el estándar W3C actual.
- ✓ Permite realizar una representación compacta de la información, comprimiendo tanto los documentos como los índices. De esta manera el almacenamiento puede ser más eficiente y exportarse a otros medios de almacenamiento.
- ✓ Puede funcionar como un anexo a una aplicación propia (embebida) o configurarse como un servidor independiente sobre la que se envían las peticiones de consulta (vía HTTP).
- ✓ Es multiplataforma al haber sido desarrollado por completo en Java. Oficialmente soporta las plataformas:
  - Windows XP, Vista, 7.
  - Linux 2.4+.
  - Mac OS X 10.5+.
- ✓ Necesita de una máquina virtual de Java tipo JRE5+.

Se ha elegido este motor porque tiene la versión Open Source que permitirá practicar con el lenguaje XQuery de manera muy sencilla y sin ningún coste asociado. Esta edición gratuita de Qizx (*Free Engine Edition*) permite tener un límite de tamaño en las BD de 1 GByte, más que suficiente para practicar con XQuery. En cualquier caso, siempre se podría comprar la versión Professional de Qizx si el diseño así lo requiere.

Aunque de ahora en adelante se hablará de Qizx, existen otros desarrollos Open Source, como Apache Lucene<sup>17</sup>, que podrían utilizarse para este cometido.

Actualmente está disponible la versión 4.1 de Qizx Free Engine (liberada el 6 de diciembre de 2010). Es posible descargarla en la página web de XMLmind<sup>18</sup>. En caso de utilizar una versión más moderna, los pasos para instalar y ejecutar la BD XML nativa son muy similares a los siguientes:

<sup>17</sup> <http://lucene.apache.org>

<sup>18</sup> <http://www.xmlmind.com/qizx/download.shtml>

- 1 Descomprimir el fichero .zip descargado (en caso de que tengamos un entorno Windows).
- 2 Si se ha descomprimido en “C:\Archivos de Programa\qizx-fe-4.1p1”, habrá una carpeta llamada “bin” y dentro un ejecutable llamado “qizxstudio.bat”. Hacemos doble clic sobre ese fichero .bat.
- 3 Aparecerá una ventana parecida a la de la Figura 6.6.

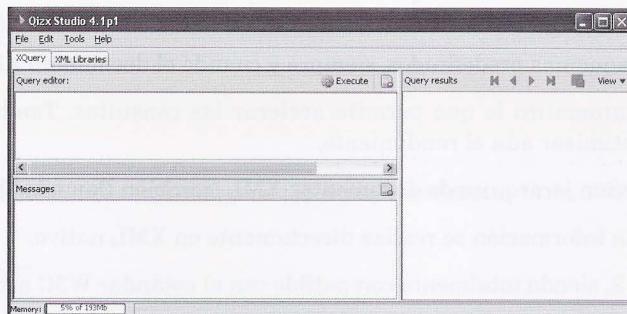


Figura 6.6. Motor Qizx Free Engine Edition

Podemos ver lo siguiente:

- Pestaña “XQuery”: servirá para insertar las consultas en formato XQuery.
- Pestaña “XML Libraries”: indica las librerías de documentos XML cargados en el sistema.
- En la barra de menú, entrada “File”: permite crear, abrir y salvar en fichero las consultas XQuery.
- En la barra de menú, entrada “Edit”: permite las acciones típicas de deshacer, rehacer, cortar, copiar y pegar.
- En la barra de menú, entrada “Tools”: permite abrir un grupo de librerías XML locales, conectarse y desconectarse de una BD XML remota y mostrar logs.
- En la barra de menú, entrada “Help”: permite acceder a la ayuda de usuario y a la versión concreta del producto.

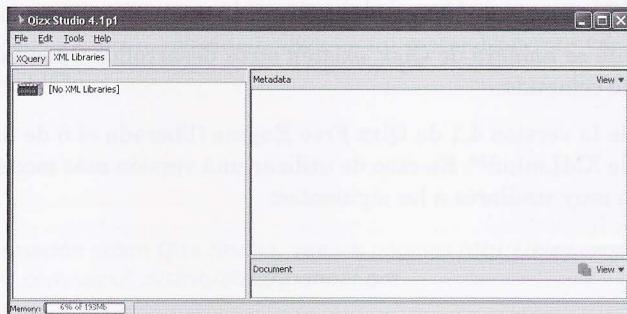


Figura 6.7. Librerías XML cargadas en Qizx

## ACTIVIDADES 6.4



- ▶ Buscar por Internet qué estándar XQuery soporta la BD “Oracle XML DB”.
- ▶ Buscar por Internet que estándar XQuery soporta la BD “BaseX”.

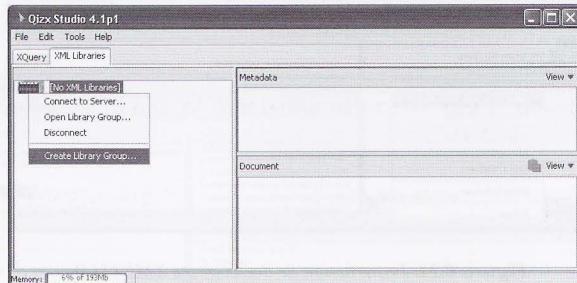


Para obtener más información, vea las páginas <http://www.saxonica.com/documentation/about/whatis.xml>, <http://saxon.sourceforge.net/> y [http://download.oracle.com/docs/cd/B28359\\_01/appdev.111/b28369/xdb\\_xquery.htm](http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28369/xdb_xquery.htm).

### 6.3.2 ADMINISTRACIÓN DE LIBRERÍAS XML

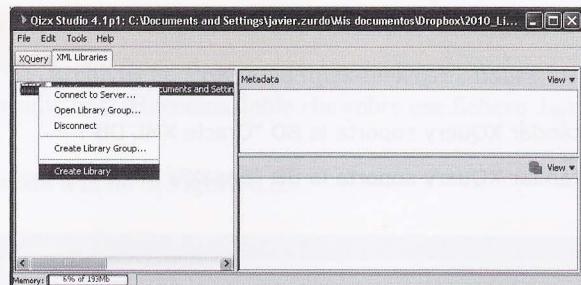
Partiendo del documento XML ejemplo que se vió de la biblioteca de libros en la sección “Transformación a XML”, vamos a intentar cargar ese fichero como Base de Datos de trabajo:

- 1** Seleccionar la pestaña “XML Libraries”.
- 2** Botón derecho “Create Library Group”. Ver Figura 6.8.

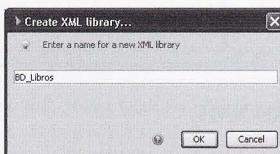


**Figura 6.8.** Creación de un grupo de librerías XML

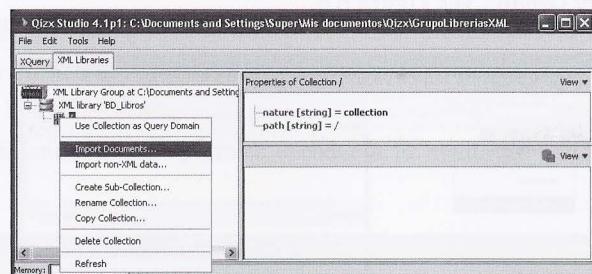
- 3** Elegimos un directorio nuevo llamado “GrupoLibreriasXML”, donde guardaremos todos nuestros documentos XML y se almacenarán los logs de las BD.
- 4** Botón derecho “Create Library”. Ver Figura 6.9.
- 5** Elegimos un nombre para la BD. En este caso daremos el nombre “BD\_Libros”. Ver Figura 6.10.

**Figura 6.9.** Creación de una librería XML (1/2)

- 6** Botón derecho “Import Documents”. Ver Figura 6.10.

**Figura 6.10.** Creación de una librería XML (2/2)

- 7** Aparecerá una ventana como en la Figura 6.11. Pulsar el botón “Add File/Folder ...”.

**Figura 6.11.** Importación de documentos XML (1/4)

- 8** Seleccionar el documento XML que se vió en la sección anterior. Llamarlo “BD\_Libros.xml” y pulsar el botón “Start Import”. Ver Figura 6.12. Si la importación ha sido exitosa, pulsar el botón “Close”.

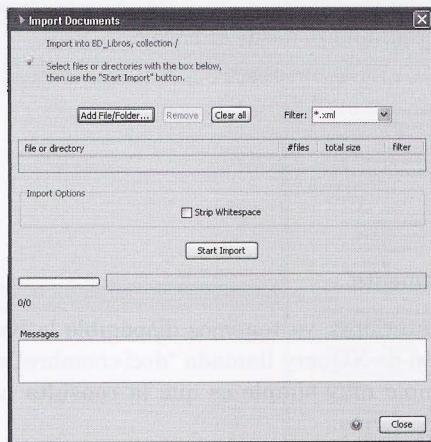


Figura 6.12. Importación de documentos XML (2/4)

9 Tras la importación, el documento quedará vinculado a la librería “BD\_Libros” y dentro del grupo de librerías “GrupoLibreriasXML”. Ver Figuras 6.13 y 6.14.

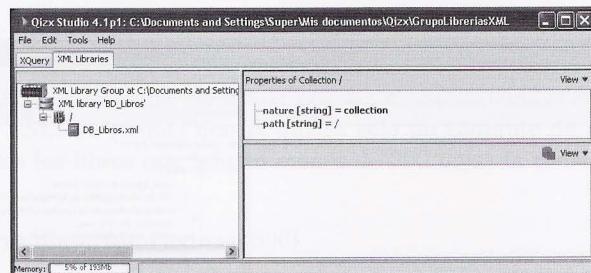


Figura 6.13. Importación de documentos XML (3/4)

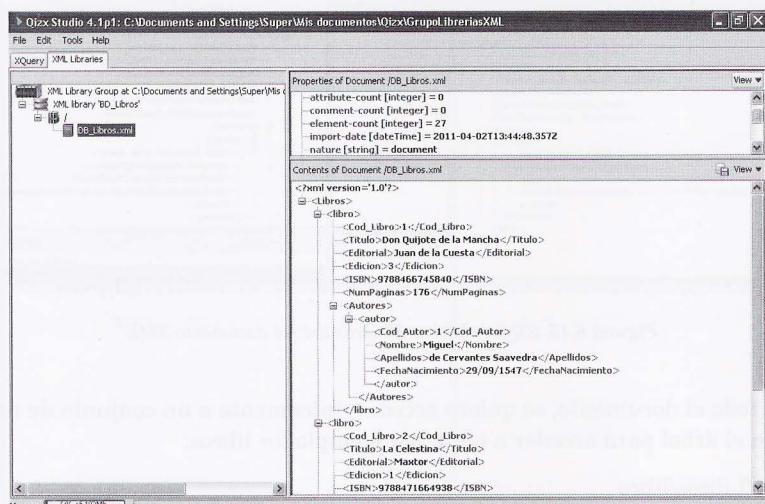


Figura 6.14. Importación de documentos XML (4/4)

Ya estaría disponible en la base de datos el documento XML. Si quisieramos importar otros documentos en la misma librería, se debería repetir el procedimiento desde el paso 6.

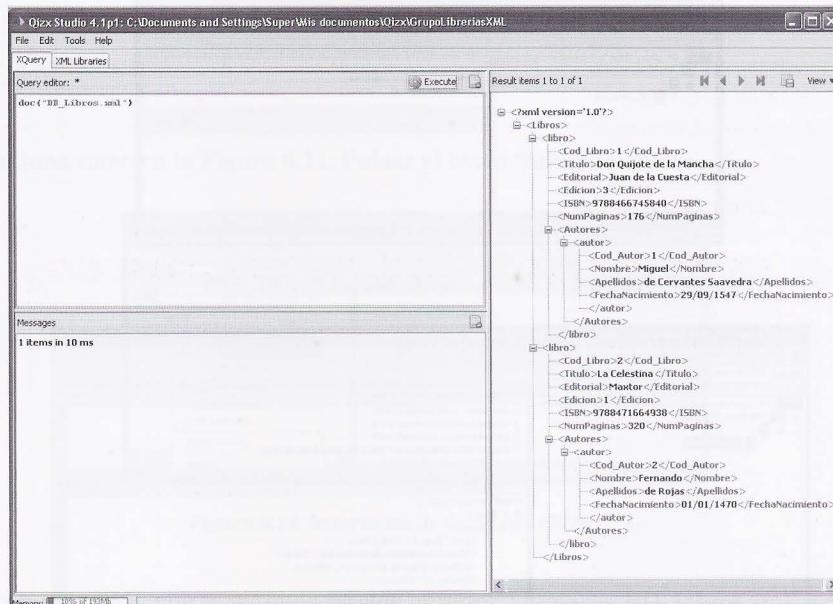
## 6.4 XQUERY

Se ha hablado de XQuery muy ligeramente.

Si se han seguido todos los pasos anteriores, ya tenemos disponible la base de datos para hacer consultas. En primer lugar se va a utilizar una función de XQuery llamada “`doc(<nombredocumento.xml>)`”, que permite extraer datos del documento indicado. Un ejemplo muy simple es que la consulta nos devuelva todo el documento XML almacenado. Esto se haría de la siguiente forma:

```
doc("BD_Libros.xml")
```

Habiendo escrito esta función en la pestaña “XQuery”, pulsamos el botón “Execute” y veremos en la parte inferior (“Messages”), si se ha ejecutado correctamente y el tiempo empleado para ello. Pero lo realmente interesante es el resultado de la consulta, que ha sido devuelta en la parte derecha de la aplicación.



**Figura 6.15.** XQuery básica: Mostrar todo el documento XML

Si en vez de devolver todo el documento, se quiere acceder únicamente a un conjunto de nodos bien identificados, añadiríamos el camino en el árbol para acceder a ellos, por ejemplo los libros:

```
doc("BD_Libros.xml")//Libros/libro
```

```
File Edit Tools Help
XQuery XML Libraries
Query editor: *
Execute View
Result items 1 to 2 of 2
doc(''BD_Libros.xml'')/Libros/libro
<libro>
 <Cod.Libro>1</Cod.Libro>
 <Titulo>Don Quijote de la Mancha</Titulo>
 <Editorial>Juan de la Cuesta</Editorial>
 <Edicion>3</Edicion>
 <ISBN>9788466745840</ISBN>
 <NumPaginas>3176</NumPaginas>
 <Autores>
 <autor>
 <Cod.Autor>1</Cod.Autor>
 <Nombre>Miguel</Nombre>
 <Apellidos>de Cervantes Saavedra</Apellidos>
 <FechaNacimiento>29/09/1547</FechaNacimiento>
 </autor>
 </Autores>
</libro>
<libro>
 <Cod.Libro>2</Cod.Libro>
 <Titulo>La Celestina</Titulo>
 <Editorial>Moxter</Editorial>
 <Edicion>1</Edicion>
 <ISBN>978847166930</ISBN>
 <NumPaginas>320</NumPaginas>
 <Autores>
 <autor>
 <Cod.Autor>2</Cod.Autor>
 <Nombre>Fernando</Nombre>
 <Apellidos>de Rojas</Apellidos>
 <FechaNacimiento>01/01/1470</FechaNacimiento>
 </autor>
 </Autores>
</libro>
```

Messages

1 items in 10 ms  
2 items in 20 ms

**Figura 6.16.** XQuery básica: Mostrar todos los libros

Como se puede observar, aparecen todos los libros que había almacenados en el documento XML inicial. Al igual que con las expresiones XSLT es posible que se quiera solo un conjunto de nodos dependiendo de un patrón de búsqueda. Si queremos todos los libros que tengan menos de 300 hojas la consulta podríamos completarla de la siguiente manera:

```
doc("BD_Libros.xml")//Libros/libro[NumPaginas<300]
```

```
File Edit Tools Help
XQuery XML Libraries
Query editor: XQuery_01.doc.qx*
Execute View
Result items 1 to 1 of 1
doc(''BD_Libros.xml'')/Libros/libro[NumPaginas<300]
<libro>
 <Cod.Libro>2</Cod.Libro>
 <Titulo>La Celestina</Titulo>
 <Editorial>Moxter</Editorial>
 <Edicion>1</Edicion>
 <ISBN>978847166930</ISBN>
 <NumPaginas>320</NumPaginas>
 <Autores>
 <autor>
 <Cod.Autor>2</Cod.Autor>
 <Nombre>Fernando</Nombre>
 <Apellidos>de Rojas</Apellidos>
 <FechaNacimiento>01/01/1470</FechaNacimiento>
 </autor>
 </Autores>
</libro>
```

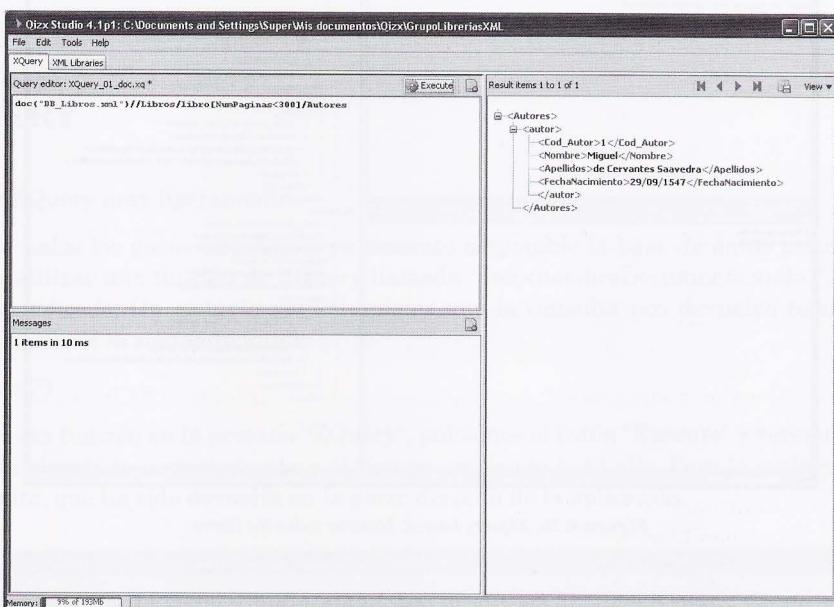
Messages

1 items in 10 ms  
2 items in 20 ms  
1 items in 60 ms

**Figura 6.17.** XQuery básica: Mostrar libros con menos de 300 páginas

¿Y si solo queremos los autores de los libros que tienen menos de 300 páginas? La consulta sería ésta:

```
doc("BD_Libros.xml")//Libros/libro[NumPaginas<300]/Autores
```



**Figura 6.18.** XQuery básica: Mostrar autores con libros de menos de 300 páginas

Los ejemplos anteriores son la manera sencilla de realizar búsquedas y selecciones de nodos concretos en un documento XML. Pero existe otra manera, mucho más potente, para realizar este trabajo. Es lo que se denomina **expresiones FLWOR**. FLWOR es la contracción del acrónimo For, Let, Where, Order by, Return. Una expresión FLWOR equivalente a la última consulta realizada, podría ser la siguiente:

```
for $libro in doc("BD_Libros.xml")//Libros/libro
where $libro/NumPaginas<300
return $libro/Autores
```

A continuación se explicará para qué sirven cada una de las cláusulas FLWOR:



Para obtener más información, vea la página [http://www.stylusstudio.com/xquery\\_flwor.html](http://www.stylusstudio.com/xquery_flwor.html).

- **for:** Esta sentencia permite seleccionar los nodos que se quieren consultar, guardándose en la variable (el identificador que le precede el símbolo \$).
- **let:** Esta cláusula es opcional. Esta sentencia establece una nueva variable sobre el mismo u otro documento XML. Permite simplificar las expresiones posteriores y tener un código mucho más legible.

- **where:** Clausula que permite establecer una condición sobre la variable indicada en “for” y “let”.
- **order by:** Clausula que define el orden de presentación de resultados.
- **return:** Permite devolver un valor concreto de los resultados obtenidos de las anteriores cláusulas (uno por nodo).

Cabe destacar que la utilización de expresiones FLWOR resulta tremadamente similar a las consultas realizadas en SQL en las bases de datos relacionales. Usaremos estas expresiones para extraer la información de nuestras bases de datos.

## 6.5 CONSULTAS

Antes de continuar, se va a crear un nuevo documento XML y se añadirá en la base de datos de Qizx Studio. La razón fundamental es que necesitamos documento más denso en datos para poder realizar ejemplos complejos.

Imaginemos que tenemos un conocido que se gana la vida con una academia de bailes de salón. Nos pide ayuda para almacenar información que considera fundamental en su negocio. Estos datos son:

- Nombre del baile.
- Precio de la clase (indicando la periodicidad de la cuota y la moneda de pago).
- Número de plazas disponibles.
- Fecha de comienzo de las clases.
- Fecha de finalización de las clases.
- Nombre del profesor que la imparte.
- Sala en la que se desarrollará la clase.

Tras un período recogiendo los datos y plasmándolos en XML, el documento queda así:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Bailes>
 <baile id="1">
 <nOMBRE>Tango</nOMBRE>
 <precio cuota="mensual" moneda="euro">27</precio>
 <plazas>20</plazas>
 <comienzo>1/1/2011</comienzo>
 <fin>1/12/2011</fin>
 <profesor>Roberto Garcia</profesor>
 <sala>1</sala>
 </baile>

 <baile id="2">
 <nOMBRE>Cha-cha-cha</nOMBRE>
 <precio cuota="trimestral" moneda="euro">80</precio>
 <plazas>18</plazas>
 <comienzo>1/2/2011</comienzo>
 <fin>31/7/2011</fin>
 <profesor>Miriam Gutierrez</profesor>
 <sala>1</sala>
 </baile>
```

```
<baile id="3">
 <nOMBRE>Rock</nOMBRE>
 <precio cuota="mensual" moneda="euro">30</precio>
 <plazas>15</plazas>
 <comienzo>1/1/2011</comienzo>
 <fin>1/12/2011</fin>
 <profesor>Laura Mendiola</profesor>
 <sala>1</sala>
</baile>

<baile id="4">
 <nOMBRE>Merengue</nOMBRE>
 <precio cuota="trimestral" moneda="dolares">75</precio>
 <plazas>12</plazas>
 <comienzo>1/1/2011</comienzo>
 <fin>1/12/2011</fin>
 <profesor>Jesus Lozano</profesor>
 <sala>2</sala>
</baile>

<baile id="5">
 <nOMBRE>Salsa</nOMBRE>
 <precio cuota="mensual" moneda="euro">32</precio>
 <plazas>10</plazas>
 <comienzo>1/1/2011</comienzo>
 <fin>1/12/2011</fin>
 <profesor>Jesus Lozano</profesor>
 <sala>2</sala>
</baile>

<baile id="6">
 <nOMBRE>Pasodoble</nOMBRE>
 <precio cuota="anual" moneda="euro">320</precio>
 <plazas>8</plazas>
 <comienzo>1/1/2011</comienzo>
 <fin>31/12/2011</fin>
 <profesor>Miriam Gutierrez</profesor>
 <sala>2</sala>
</baile>
</Bailes>
```

La inclusión de este documento en la base de datos Qizz es tan sencilla como ir a la pestaña “XML Libraries”, seleccionar como “XML Library Group” la “GrupoLibreriasXML”. Después seleccionar “BD\_Libros” como “XML library”. Y en el raíz de BD\_Libros, botón derecho “Import Documents ...”. El procedimiento es igual que el indicado en el apartado 6.3.2.

Una vez cargado en la base de datos, podemos volver a la pestaña “XQuery” y realizar las consultas. Establezcamos las siguientes consultas para practicar:



## EJEMPLO 6.1

Se necesita saber qué bailes se realizan en la sala número 1:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
where $baile/sala = 1
return $baile/nombre
```

The screenshot shows the Oizx Studio 4.1 interface with the following details:

- Query editor:** XQuery\_04\_FLWOR\_Baile\_sala1.xq
 

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
where $baile/sala = 1
return $baile/nombre
```
- Messages:** 3 items in 10 ms
- Result:** Result items 1 to 3 of 3
 

```
<nombre>Tango</nombre>
<nombre>Cha-cha-cha</nombre>
<nombre>Rock</nombre>
```

Figura 6.19. FLWOR: Nodos de bailes impartidos en la sala 1

Como puede observarse en la Figura 6.19, los resultados salen con las etiquetas de XML. Si quisieramos solo tener un listado, obviando las etiquetas “<nombre>” y “</nombre>”, simplemente tendríamos que indicar en la cláusula “return” que extraiga los datos de ese elemento XML:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $n:=$baile/nombre
where $baile/sala = 1
return data($n)
```

The screenshot shows the Oizx Studio 4.1 interface with the following details:

- Query editor:** XQuery\_04\_FLWOR\_Baile\_sala1\_bis.xq
 

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $n:=$baile/nombre
where $baile/sala = 1
return data($n)
```
- Messages:** Current category: BD\_LIDOS
  - 3 items in 130 ms
  - 3 items in 20 ms
- Result:** Result items 1 to 3 of 3
  - xsuntypedAtomic = Tango
  - xsuntypedAtomic = Cha-cha-cha
  - xsuntypedAtomic = Rock

Figura 6.20. FLWOR: Datos de los nodos de bailes impartido en la sala 1

Tal y como se ve en la Figura 6.20, los resultados ya no están con las etiquetas del elemento “nombre”. Todo es debido a que existe una función llamada “data(<variable>)”, que extrae del nodo la información almacenada.



## EJEMPLO 6.2

Se necesita extraer los nodos de aquellos bailes que se imparten en la sala número 2 y cuyo precio sea menor que 35 euros:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $n:=$baile/nombre
where $baile/sala = 2 and $baile/precio < 35
 and $baile/precio[@moneda="euro"]
return $n
```

El resultado será:

```
<nombre>Salsa</nombre>
```



## EJEMPLO 6.3

Se necesita saber el nombre de los profesores que dan clases con cuotas mensuales:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $profesor:=$baile/profesor
where $baile/precio[@cuota="mensual"]
return $profesor
```

El resultado será:

```
<profesor>Roberto Garcia</profesor>
<profesor>Laura Mendiola</profesor>
<profesor>Jesus Lozano</profesor>
```

Estos tres ejemplos han permitido descubrir las grandes similitudes que tiene XQuery con SQL. Una de las ventajas de XQuery es que devuelve los nodos del árbol XML, por lo que cualquier programa o aplicación puede trabajar con los resultados que la consulta devuelva. Si lo que se quiere es extraer solo la información y no los nodos, hemos descubierto que la función "data(<variable>)" nos permite conseguirlo. Pero, ¿y si lo que se quiere es devolver un fichero HTML con los datos de la consulta? Quizás el lector recuerde que con XSL se podía tratar este caso. A continuación veremos como generar HTML con una consulta a una base de datos XML nativa, mediante expresiones FLWOR.

### 6.5.1 DE FLWOR A HTML

Queremos crear una consulta XQuery que tras ejecutarla nos devuelva los resultados en formato HTML. Podemos unir dentro de XQuery etiquetas HTML y expresiones o cláusulas FLWOR. La única limitación es que cuando se fusionan en una consulta, indiquemos al motor de consultas qué parte es la que tiene que procesar como consulta. Para ello indicaremos entre llaves ("[", "]"), que parte es FLWOR. La mejor manera de verlo es a través de un par de ejemplo (se marcará en negrita la parte de la consulta FLWOR):



## EJEMPLO 6.4

Queremos una consulta XQuery cuyo resultado sea una tabla HTML que nos muestre el nombre del baile, el profesor que lo imparte y el número de plazas ofertadas:

```
<html>
<body>
 <h1> Bailes ofertados </h1>
 <table border="1">
 <tr>
 <th>Nombre baile</th>
 <th>Nombre profesor</th>
 <th>Plazas ofertadas</th>
 </tr>
 {
 for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
 let $nombre:=$baile/nombre
 let $profesor:=$baile/profesor
 let $plazas:=$baile/plazas
 return
 <tr>
 <td>{data($nombre)}</td>
 <td>{data($profesor)}</td>
 <td>{data($plazas)}</td>
 </tr>
 }
 </table>
</body>
</html>
```

El resultado es:

```
<html>
<body>
 <h1> Bailes ofertados </h1>
 <table border="1">
 <tr>
 <th>Nombre baile</th>
 <th>Nombre profesor</th>
 <th>Plazas ofertadas</th>
 </tr>
 <tr>
 <td>Tango</td>
 <td>Roberto Garcia</td>
 <td>20</td>
 </tr>
 </table>
</body>
</html>
```

```
<tr>
<td>Cha-cha-cha</td>
<td>Miriam Gutierrez</td>
<td>18</td>
</tr>
<tr>
<td>Rock</td>
<td>Laura Mendiola</td>
<td>15</td>
</tr>
<tr>
<td>Merengue</td>
<td>Jesus Lozano</td>
<td>12</td>
</tr>
<tr>
<td>Salsa</td>
<td>Jesus Lozano</td>
<td>10</td>
</tr>
<tr>
<td>Pasodoble</td>
<td>Miriam Gutierrez</td>
<td>8</td>
</tr>
</table>
</body>
</html>
```

**Bailes ofertados**

Nombre baile	Nombre profesor	Plazas ofertadas
Tango	Roberto Garcia	20
Cha-cha-cha	Miriam Gutierrez	18
Rock	Laura Mendiola	15
Merengue	Jesus Lozano	12
Salsa	Jesus Lozano	10
Pasodoble	Miriam Gutierrez	8

*Figura 6.21. FLWOR: Tabla HTML con baile, profesor y plazas*

Como puede observarse en la Figura 6.21, el resultado es totalmente compatible con HTML y visible en cualquier navegador actual.



## EJEMPLO 6.5

Queremos realizar la misma consulta anterior pero estableciendo la condición de ser bailes con cuota trimestral. Además se ordenará, de menor a mayor, los bailes según el número de plazas disponibles. La consulta XQuery es exactamente igual que antes pero cambiando únicamente la parte FLWOR. Remarcamos en negrita la diferencia con respecto a lo anterior:

```
{
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $nombre:=$baile/nombre
let $profesor:=$baile/profesor
let $plazas:=$baile/plazas
where $baile/precio[@cuota = "trimestral"]
order by $baile/plazas
return
 <tr>
 <td>{data($nombre)}</td>
 <td>{data($profesor)}</td>
 <td>{data($plazas)}</td>
 </tr>
}
```

El resultado es:

```
<html>
<body>
 <h1> Bailes ofertados </h1>
 <table border="1">
 <tr>
 <th>Nombre baile</th>
 <th>Nombre profesor</th>
 <th>Plazas ofertadas</th>
 </tr>
 <tr>
 <td>Merengue</td>
 <td>Jesus Lozano</td>
 <td>12</td>
 </tr>
 <tr>
 <td>Cha-cha-cha</td>
 <td>Miriam Gutierrez</td>
 <td>18</td>
 </tr>
 </table>
</body>
</html>
```

Bailes ofertados		
Nombre baile	Nombre profesor	Plazas ofertadas
Merengue	Jesus Lozano	12
Cha-cha-cha	Miriam Gutierrez	18

Figura 6.22. FLWOR: Tabla HTML con bailes con cuota trimestral

Puede verse el resultado en el navegador web en la Figura 6.22.

## 6.6 ACTUALIZACIÓN

Hasta el momento solo se ha podido realizar consultas sobre los contenidos ya almacenados en la base de datos XML. Quizás nuestro amigo quiera insertar, reemplazar, renombrar, cambiar y/o borrar entradas dentro de la base de datos. Para ello indicaremos qué hacer en cada caso.



Para obtener más información, vea las páginas <http://www.xmlplease.com/xquery-update>, [http://exist.sourceforge.net/update\\_ext.html](http://exist.sourceforge.net/update_ext.html) y <http://stackoverflow.com/questions/5335046/xquery-update-insert-or-replace-depending-if-node-exists-not-possible>.

### 6.6.1 INSERCIÓN



#### EJEMPLO 6.6

Se quiera añadir un nuevo baile en la base de datos. Los datos son:

- Nombre: Foxtrot.
- Precio: 22 dólares.
- Pago: mensual.
- Plazas: 12.
- Comienzo: 01/01/2012.
- Fin: 31/07/2012.
- Profesor: Freddy Astaire.
- Sala: 3.

La inserción se realizaría de la siguiente manera:

```
insert node
<baile id="7">
 <nOMBRE>Foxtrot</nOMBRE>
 <precio cuota="mensual" moneda="dolares">22</precio>
 <plazas>12</plazas>
 <comienzo>01/01/2012</comienzo>
 <fin>31/07/2012</fin>
 <profesor>Freddy Astaire</profesor>
 <sala>3</sala>
</baile>
before doc("DB_BailesDeSalon.xml")//Bailes/baile[1]
```

Este código inserta el nodo indicado en la base de datos "DB\_BailesDeSalon.xml". El nodo se insertará antes del primer nodo de la base de datos (por la cláusula "before"). Si se quiere insertar después, solo cambiaría "before" por "after". Ambas utilizan como referencia al primer nodo ("[1]").

Una sentencia equivalente para insertar al principio de la base de datos, sin tener que referenciar a nodo alguno de la base de datos, sería sustituyendo la última línea por esta otra:

```
as first into doc("DB_BailesDeSalon.xml")//Bailes
```

Si por el contrario, lo que se desea es insertar al final de la base de datos, sin referenciar a ningún nodo, se realizaría de la siguiente manera:

```
as last into doc("DB_BailesDeSalon.xml")//Bailes
```

## 6.6.2 REEMPLAZO



### EJEMPLO 6.7

En la inserción anterior se cometieron dos errores:

1. El nombre correcto era "Angel Correllada".
2. El número de plazas realmente eran 14.

Si en la inserción anterior se realizó antes del primer nodo de la base de datos, entonces el elemento insertado será ahora el primero. Se cambiarán esos datos de dos maneras distintas:

- Mediante la modificación del valor del nodo.
- Mediante el reemplazo del nodo completo.

La modificación se realizaría de la siguiente manera:

```
replace value of node
doc("DB_BailesDeSalon.xml")//Bailes/baile[1]/profesor
with "Angel Correllada"
,
```

```
replace node
doc("DB_BailesDeSalon.xml")//Bailes/baile[1]/plazas
with <plazas>14</plazas>
```

Como puede verse, las dos modificaciones se han realizado a la vez pero separándolas por una coma ",". En el caso de no saber en qué posición de la base de datos se ha realizado la inserción (en principio debe dar igual la posición mientras esté insertados los datos), se podría tener un problema de actualizar erróneamente la tupla incorrecta. Para evitar este problema, lo más común es realizar la modificación mediante la utilización del identificador del baile (en este caso id=7).

```
replace value of node
doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]/profesor
with "Angel Correllada"
,
```

```
replace node
doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]/plazas
with <plazas>14</plazas>
```

**6.6.3 BORRADO****EJEMPLO 6.8**

Después de la inserción y la modificación del nuevo baile en la base de datos, finalmente parece que no se va a desarrollar ese curso. Se pide que finalmente se borre esa tupla en la base de datos (id=7):

```
delete node doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]
```

**6.7 EXPORTACIÓN DE LIBRERÍAS XML**

Después de realizar todos los cambios en la base de datos, es un buen momento para exportar el contenido a un fichero XML externo. De esta manera se podría tener una copia de seguridad en caso de contingencia. Qizz Studio nos permite realizarlo de manera muy sencilla:

- 1** Ir a la pestaña “XML\_Libraries”.
- 2** Seleccionar “DB\_BailesDeSalon.xml”, botón derecho y pulsar “Export to File ...”.
- 3** Aparece una nueva ventana en la que se puede seleccionar el fichero de salida, la codificación y el formato (XML, HTML, XHTML, o texto). Para hacer una copia de seguridad se selecciona XML y se procederá a volcar la información. Ver Figura 6.23.

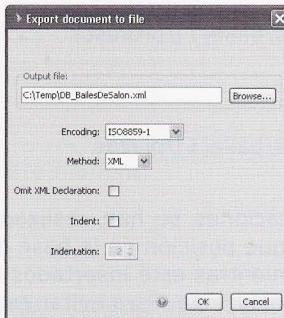


Figura 6.23. Exportación de librerías XML en Qizz

Para finalizar, se habló anteriormente de cómo utilizar expresiones FLWOR para que devuelvan código HTML. El procedimiento para exportar los resultados es muy sencillo. Una vez que se está en la pestaña “XQuery” y se ejecuta la consulta, en la parte derecha muestra los resultados. Hay que pulsar el botón de exportación de resultados “Save results” y aparecerá una ventana similar a la de la Figura 6.23. De esta manera tendremos en fichero los resultados de la consulta.

## 6.8 OTRAS FUNCIONES O LIBRERIAS

Se ha visto que XQuery<sup>19</sup> es una herramienta de alto nivel que permite realizar todas las tareas administrativas de un SGBD sin tener que programar ninguna línea en C/C++ o Java. Pero XQuery no es solamente lo que se ha visto aquí. Recordemos que XQuery usa por XPath y éste tiene muchas funcionalidades que no se han tratado en este capítulo. Existen funciones para acceder a nodos padre, funciones para tratamiento de valores numéricos, para tratar strings, resolver URIs, funciones para tratar fechas/horas, para acceder a nodos, para establecer secuencias... En el caso de necesitar más potencia a la hora de generar las consultas, se recomienda consultar las funciones de XPath<sup>20</sup>.

Otra manera de acceder a los nodos de los documentos XML es mediante la librería DOM<sup>21</sup> (*Document Object Model*). DOM es una interfaz de programación de aplicaciones (API), que permite representar documentos HTML y/o XML de manera que las aplicaciones que la utilicen puedan representar el documento en forma de árbol. Una vez cargado el árbol, se puede recorrer buscando información, modificando los propios nodos o la información que contienen, cambiando la estructura del árbol, etc. Esta API es un estándar de acceso aprobada por la W3C. El inconveniente es que es necesario conocer un lenguaje de programación para hacer uso de esta API. En la actualidad la práctica mayoría de los navegadores la tienen implementada.

Por último tenemos otra API llamada SAX<sup>22</sup> (*Simple API for XML*). Es una librería que inicialmente se utilizaba en Java pero que en la actualidad es posible utilizarla en muchos lenguajes de programación. La principal ventaja de usar un parser SAX es que el documento XML se lee una única vez, de manera secuencial y sin que se cargue todo el en memoria (sin generar ningún árbol). Esto redunda en que los analizadores SAX son más eficientes en la memoria utilizada que los DOM. Como desventaja es que una vez pasada la lectura, no se puede volver a atrás (en contraposición de DOM que sí se permite).

En resumen, existen muchas maneras de manipular documentos XML. En este libro se ha elegido XQuery pues permite utilizarse en los grandes sistemas SGBD y también en pequeñas aplicaciones con motores de BD XML nativas como Qizx. No será necesario tener conocimientos de programación en lenguajes como C/C++ o Java para manejar API's como SAX o DOM. En cualquier caso, la decisión final de utilizar una u otra solución dependerá de las necesidades que tenga el proyecto al que vaya a implantarse, haciendo especial hincapié en el almacenamiento y el tratamiento de la información.

## 6.9 CASO PRÁCTICO

Se propone analizar y diseñar un caso concreto de almacenamiento de información en bases de datos. En este caso se dará libertad al alumno (bajo supervisión del profesor) del sistema a modelar. Podrían ser casos como:

- Una mediateca.
- Almacenamiento de transacciones bancarias.
- Almacenamiento de mensajes de texto recibidos de publicidad en TV.
- Almacenamiento de ofertas de hoteles y viajes.
- Etc.

19 <http://www.w3.org/TR/xquery/>

20 <http://www.w3.org/TR/xpath/>

21 <http://www.w3.org/DOM/>

22 <http://www.saxproject.org/>

Se realizarán todos los pasos necesarios para pasar de un modelo de datos relacional a un modelo en base de datos XML nativa. Se generará un documento XML que almacene dicha información y se realizarán, al menos dos consultas para:

- Inserción.
- Modificación.
- Borrado.
- Consulta de información con presentación de resultados en HTML.



## RESUMEN DEL CAPÍTULO



En este capítulo se ha pretendido poner en conocimiento la técnica Fermi para la aproximar las necesidades de un entorno cuando este último no tiene datos suficientes para ello. Esta técnica no está explícitamente explicada pero se propone como actividad por sus grandes resultados en las estimaciones de cualquier disciplina (y no solo en la estimación del almacenamiento requerido en una base de datos).

Se ha puesto de manifiesto qué tipos de bases de datos existen en la actualidad y se han relacionado entre sí mostrando las fortalezas y las debilidades de cada una de ellas. Se ha introducido el concepto de Sistemas de Gestión de Bases de Datos como el elemento central en la administración de las bases de datos. Se incide en la importancia real de las transacciones y en las características que se deben implantar en los SGBD para impedir la pérdida de información (ACID).

Se ha relacionado SQL y XQuery como lenguajes que permiten realizar las búsquedas, inserciones, borrados o modificaciones de las tuplas, tablas o bases de datos en sistemas relacionales y sistemas nativos XML respectivamente. Además se ha proporcionado un método para realizar la conversión de un modelo de datos relacional a un sistema basado en documentos XML para su inclusión en una BD XML nativa.

Se ha proporcionado los elementos básicos para manejar una BD XML nativa, como Qizx Studio sin que el usuario tenga que realizar un desembolso económico elevado para practicar las consultas XQuery. Además se ha enseñado como con las expresiones FLWOR se permite generar como resultados documentos XML o HTML utilizables en otras aplicaciones externas.

Para finalizar se ha comparado someramente como otras APIs permiten realizar una funcionalidad similar a XQuery pero necesitando conocimientos de lenguajes de programación como C/C++ o Java para tener acceso a todos los nodos del documento XML.



## EJERCICIOS PROPUESTOS



■ 1. ¿A qué científico debemos la posibilidad de realizar estimaciones muy precisas a problemas con falta de datos suficientes? Pista, buscar en Internet por "cuestiones de Fermi".

■ 2. Buscar por Internet los siguientes conceptos:

- B2B.
- B2C.
- B2E.

Realizar una tabla en la que se muestren las ventajas de la aplicación de estas estrategias comerciales en una empresa.

■ 3. Añadir en la DTD de la biblioteca de libros, la nacionalidad del autor. Completar el nuevo campo en el documento XML que los almacena.

■ 4. Añadir en el documento XML 10 nuevos libros.

■ 5. Realizar una consulta XQuery con expresiones FLWOR que muestre todos los libros almacenados.

■ 6. Realizar una consulta XQuery con expresiones FLWOR que muestre todos los libros almacenados con más de 500 páginas.

■ 7. Realizar una consulta XQuery con expresiones FLWOR cuyo resultado sea una tabla HTML con todos los libros de la biblioteca.

■ 8. Realizar una consulta XQuery con expresiones FLWOR cuyo resultado sea una tabla HTML con todos los libros de la biblioteca siempre que tengan más de 150 páginas.

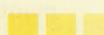
■ 9. Realizar una inserción de un nuevo libro con XQuery (con algún dato erróneo).

■ 10. Realizar una modificación del libro anterior corrigiendo el dato erróneo.

■ 11. Realizar el borrado del último libro insertado en la base de datos con XQuery.



## TEST DE CONOCIMIENTOS



**1** ¿Qué es un problema de Fermi?

- a) Es una manera de evaluar una solución a un problema cuando se tienen datos suficientes.
- b) Es una manera de evaluar una solución a un problema cuando se tienen datos insuficientes.
- c) Es una manera de estimar una solución a un problema cuando se tienen muy pocos datos disponibles para resolverlo.
- d) Es una manera de estimar una solución a un problema cuando se tiene datos suficientes para resolverlo.
- e) Ninguna de las anteriores.

**2** Si unos datos que se han conseguido nunca van a cambiar:

- a) Se recomienda utilizar una base de datos que los almacene de manera estática. De esta manera las consultas serán más rápidas y eficientes.
- b) Se recomienda utilizar una base de datos que los almacene de manera dinámica. De esta manera las consultas serán más rápidas y eficientes.
- c) Se recomienda utilizar una base de datos que los almacene de manera estática. De esta manera las consultas serán un poco más rápidas si los datos cambian en el futuro.
- d) Ninguna de las anteriores.

### 3 ¿Qué es una tupla?

- a) Es una columna de una tabla en la que se almacena los datos de manera compacta.
- b) Es una columna o fila (es indiferente), de una tabla en la que se almacena los datos de manera eficiente.
- c) Es una fila de una tabla en la que se almacena los datos de manera compacta.
- d) Ninguna de las anteriores.

### 4 Una base de datos jerárquica:

- a) Utiliza nodos con múltiples padres en los que se ingresan en tablas relacionales para no perder la redundancia de datos.
- b) Utiliza nodos con un único parente en la que se mantienen las redundancias de datos de manera sencilla gracias a su estructura de árbol.
- c) Utilizan árboles para distribuir la información entre sus nodos pero son poco eficientes al realizar las búsquedas.
- d) Utilizan árboles para distribuir la información entre sus nodos pero son poco eficientes para guardar la redundancia de datos.

### 5 Una base de datos de red:

- a) Utiliza nodos con múltiples padres en los que se ingresan en tablas relacionales para perder la redundancia de datos y ser eficientes.
- b) Utiliza nodos con un único parente en la que se mantienen las redundancias de datos de manera sencilla gracias a su estructura de árbol.
- c) Es tremadamente distinta a una jerárquica pues la de red utiliza árboles con un único parente. No distribuye bien la información entre sus nodos pero son muy eficientes para realizar búsquedas.
- d) Ninguna de las anteriores.

### 6 Una base de datos relacional:

- a) Utiliza nodos con múltiples padres en una estructura de árbol.
- b) Utiliza nodos con un único parente en una estructura de árbol.
- c) Utiliza tablas para almacenar la información. La ubicación de la tupla es muy importante en este tipo de base de datos, no como en las bases de datos jerárquicas que era indiferente.
- d) Utiliza tablas para almacenar la información. La ubicación de la tupla no es importante, no como en las bases de datos jerárquicas que era fundamental.

### 7 ¿Qué es ACID?

- a) Un conjunto de características que se deben cumplir para que las bases de datos jerárquicas almacenen la redundancia.
- b) Un conjunto de características que se deben cumplir para que las bases de datos relacionales almacenen la redundancia.
- c) Un conjunto de características que permiten asegurar que una transacción se ha realizado correctamente (o bien o nada).
- d) Ninguna de las anteriores.

### 8 Un Sistema de Gestión de Bases de Datos (SGBD) puede verse como:

- a) Un hardware en el que se le instala el SGBD (software) para almacenar los datos en las bases de datos y los usuarios accederán a él para realizar consultas.
- b) Un hardware donde se almacena los datos en bruto sin necesitar ningún software. Los usuarios accederán al hardware y podrán consultar los datos.
- c) Un software de SGBD y un hardware específico para él. En cualquier caso, el binomio hardware-software se compra junto e inseparable para que funcione y los usuarios acceden a él para realizar consultas.
- d) Ninguna de las anteriores.

### 9 ¿Qué roles se pueden asignar a un usuario en un Sistema de Gestión de Bases de Datos (SGBD)?

- a) Rol de administrador de entradas, rol de administrador de salidas y rol de usuario final.
- b) Rol de administrador de sistemas, rol de administrador de la base de datos, rol de usuario final y rol de gerente general.
- c) Rol de administrador de sistemas, rol de administrador de la base de datos, rol de usuario final.
- d) Ninguna de las anteriores.

### 10 En la implementación de un modelo de datos relacional:

- a) Cuando dos tablas están relacionadas por algún campo común, dichas relaciones se almacenan en una tercera tabla que almacena la relación.
- b) La tupla es una fila que contienen un conjunto de datos de interés.
- c) Las dos anteriores son correctas.
- d) Ninguna de las anteriores.

## 11 ¿Qué es XQuery?

- a) Es un lenguaje para poder modificar los documentos XSL.
- b) Es un lenguaje para poder consultar en una base de datos XML.
- c) XQuery es a XML lo que SQL a las bases de datos relacionales.
- d) B y C son correctas.
- e) Ninguna de las anteriores.

## 12 ¿Qué es XPath?

- a) Es una herramienta que permite procesar las consultas de SQL.
- b) Es una herramienta que permite recorrer árboles de nodos en documentos XML. XQuery es de más alto nivel que XPath por lo que no lo usa.
- c) Es una herramienta que permite recorrer árboles de nodos en documentos XML. XQuery lo usa para acceder a los nodos del árbol y realizar así una consulta.
- d) A y C son correctas.
- e) Ninguna de las anteriores.

## 13 Sabiendo que el documento "BD\_Autores.xml" tiene todos los autores de una biblioteca, ¿Qué devuelve la siguiente consulta XQuery?

- ```
doc("BD_Autores.xml")//Autores/Autor
```
- a) Devolvería todos los nodos del árbol que sean del tipo "Autor".
 - b) Devolvería todo el documento XML.
 - c) Devolvería solo el contenido de los nodos del árbol que sean del tipo "Autor".
 - d) Ninguna de las anteriores.

14 Sabiendo que el documento "BD_Autores.xml" tiene todos los autores de una biblioteca, ¿qué devuelve la siguiente consulta XQuery?

- ```
doc("BD_Autores.xml"):
```
- a) Devolvería todos los nodos del árbol que sean del tipo "Autor".
  - b) Devolvería todo el documento XML.
  - c) Devolvería solo el contenido de los nodos del árbol que sean del tipo "Autor".
  - d) Ninguna de las anteriores.

## 15 Sabiendo que el documento "BD\_Autores.xml" tiene todos los autores de una biblioteca, ¿qué devuelve la siguiente consulta XQuery?

- ```
doc("BD_Autores.xml")//Autores/Autor[Cod_Autor=5]
```
- a) Devuelve el nodo del árbol de tipo "Autor" cuyo identificador es igual a '5'.
 - b) Devuelve el quinto nodo del árbol de tipo "Autor".
 - c) Devuelve el quinto nodo del árbol de tipo "Libro".
 - d) Ninguna de las anteriores.

16 ¿Qué es FLWOR?

- a) Final, Línea, tipo Word, función OR, Retorno.
- a) Fecha, Devuelve el quinto nodo del árbol de tipo "Autor".
- b) For, Let, Where, Order by, Return.
- c) Ninguna de las anteriores.

17 ¿FLWOR permite devolver resultados en HTML?

- a) Sí, pero hay que formalizar una consulta especial en la que se genere las partes del documento HTML.
- b) Sí, sin ningún problema. Es más, XQuery siempre devuelve en HTML por defecto.
- c) No, en ningún caso se podría realizar.
- d) Ninguna de las anteriores.

18 Se quiere hacer una inserción de un nuevo autor en una base de datos XML mediante XQuery. Si se quiere insertar en primer lugar del árbol de nodos, ¿qué debería hacerse?

- a) insert node <autor><nombre>Barbic</autor> before doc("BD_Autores.xml")//Autores/autor[1]
- b) insert node <autor><nombre>Barbic</autor> as first into doc("BD_Autores.xml")//Autores
- c) insert node <autor><nombre>Barbic</autor> as last into doc("BD_Autores.xml")//Autores
- d) A y B son correctas.
- e) A y C son correctas.
- f) Ninguna de las anteriores.

7

Aplicación de los lenguajes de marcas a la sindicación de contenidos

OBJETIVOS DEL CAPÍTULO

- ✓ Aprender qué es la sindicación de contenidos y su uso.
- ✓ Conocer los estándares más usados para sindicación, entre ellos RSS 0.91, RSS 0.92, RSS 1.0 y RSS 2.0.
- ✓ Aprender qué son los agregadores de contenidos y directorios de canales.

7.1 INTRODUCCIÓN A LA SINDICACIÓN DE CONTENIDOS

Hoy en día existe un gran número de páginas, sitios y, en general, contenido disponible en la web. Tradicionalmente, los usuarios debían ir buscando y accediendo a dichos contenidos de forma manual e individual, lo que provocaba que no fueran capaces casi nunca de llegar a visitar todo el contenido que era realmente de su interés.

En este contexto es donde surge la sindicación de contenidos, que pretende hacer que parte del contenido de un sitio esté disponible para su uso a través de otros servicios o canales de información. Habitualmente, el contenido sindicado suele ser simplemente la información más relevante del propio contenido, como pueden ser titulares, enlaces al contenido, un resumen o, incluso, únicamente algún tipo de metadato (información acerca del contenido como, por ejemplo, palabras clave). El **contenido sindicado** se conoce también con el término inglés *feed*.

La presencia de canales de sindicación en los sitios web hace posible utilizar algún tipo de software o servicio que sea capaz de agregar todos los *feeds* que puedan contener información de interés para un usuario y mostrárselo de forma unificada y resumida. De esta manera, el propio usuario no tiene que ir recorriendo los distintos sitios para encontrar los contenidos sino que puede, de un vistazo rápido, encontrar lo que le interesa y descartar el resto.

7.2 ESTRUCTURA DE UN SISTEMA DE SINDICACIÓN

La estructura de un sistema de sindicación de contenidos parte del ciclo de vida de los propios contenidos. Habitualmente, este ciclo de vida comienza con el trabajo, tanto creativo como informativo, de uno o varios autores y termina cuando el contenido llega y es consumido, de alguna u otra manera, por el público objetivo.

Entre medias existen diferentes opciones para recorrer este camino, así como diferentes tecnologías y arquitecturas posibles. La más simple y tradicional es generar el contenido y publicarlo directamente a través de los canales de difusión de páginas web tradicionales usando, por ejemplo, HTML. Otra opción, la más usada actualmente, es servirse de un sistema específico de gestión de contenidos (**CMS**, del inglés *Content Management System*), que facilita tanto la gestión de los mecanismos de publicación como el desarrollo de contenidos de forma colaborativa. Entre los CMS más habituales de uso libre se pueden encontrar *Joomla* o *Drupal*, si bien existen otros muchos.

Cualquiera de estas dos opciones parte de un contenido, en el primer caso generado por los autores y en el segundo haciendo uso de alguna tecnología (texto plano, bases de datos, documentos XML), y producen una transformación del mismo para presentarlo de forma amigable y estética ante los usuarios. La salida de esta transformación suele estar compuesta de HTML o XHTML combinado con CSS y algunas tecnologías adicionales como Javascript o Macromedia Flash®.

Sin embargo, la salida de esta transformación puede ser también simplemente un conjunto de titulares o metadatos que formateados usando XML pueden ser sindicados. De esta forma, los usuarios tendrán otra interfaz para acceder a los contenidos que les resulten de interés. Por contra, los usuarios no van a encontrar amigable ni atractivo leer directamente la salida en XML generada, por lo que para ello se usarán herramientas específicas que transformen dicho contenido sindicado en algo más legible.

Además, los usuarios generalmente desean poder sindicarse a más de un canal de generación de contenidos. Para facilitar esta tarea han surgido en los últimos años herramientas que permiten mostrar de forma conjunta los

contenidos de distintos canales, permitiendo distintos tipos de organización tanto por temática como por ejemplo por fecha de publicación. Este tipo de herramientas se denominan **agregadores de contenidos** o bien **lectores de feeds**. Los agregadores de contenidos proporcionan al usuario un punto de entrada único para acceder a los contenidos que le interesan. Además, permite al usuario no tener que estar accediendo a los distintos canales de forma independiente sino que, por decirlo de alguna forma, los propios contenidos “buscan” al usuario. Cuanto mayor sea la cantidad de canales de contenidos que el usuario quiera acceder de forma habitual mayor será el tiempo que ahorra usando los agregadores de contenidos.

La Figura 7.1 recoge la arquitectura de un sistema de sindicación de contenidos así como la integración de los agregadores de contenidos en todo el proceso de sindicación.

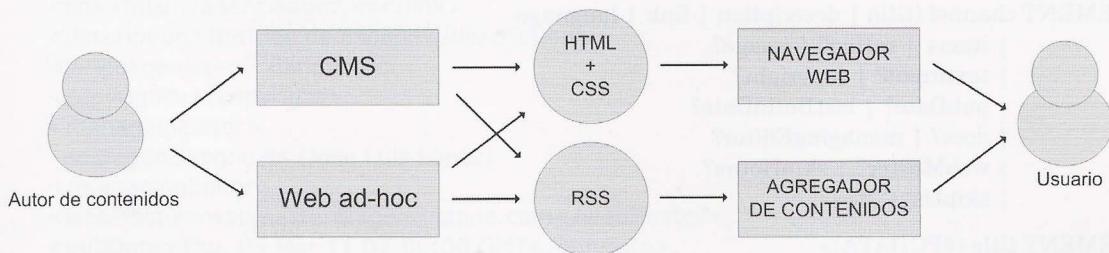


Figura 7.1. Estructura de un sistema de sindicación

7.3 ESTÁNDARES ACTUALES PARA SINDICACIÓN DE CONTENIDOS

Hoy en día el formato usado para sindicación de contenidos es el denominado **RSS** (del inglés *Really Simple Syndication*). RSS usa XML para representar el contenido que se va a sindicar y ha sufrido una evolución en los últimos diez años que lo ha llevado desde poder agregar unos pocos canales de contenidos a trabajar con una cantidad de información casi ilimitada.

RSS nació alrededor del año 1997 mediante un desarrollo inicial por parte de Dave Winer llamado *scriptingNews*, el cual fue adoptado posteriormente en 1999 por Netscape dando lugar a la primera versión pública de RSS: RSS 0.90. A partir de ahí, en poco tiempo, de nuevo entre Dave Winer de Userland (una empresa de desarrollo de software muy ligada a este tipo de tecnologías) y Netscape, desarrollaron una nueva versión denominada RSS 0.91. Poco después, Netscape dejó de desarrollar RSS y UserLand publicó la especificación oficial de RSS 0.91 y posteriormente una versión mejorada RSS 0.92.

En el año 2000, un consorcio liderado por Rael Dornfest desarrolló la versión RSS 1.0 que, aunque parece que podría ser una continuación de RSS 0.91, en realidad, supuso una versión completamente independiente y distinta en muchos aspectos. En paralelo a este consorcio, Dave Winer y UserLand desarrollaron RSS 0.92, siendo el propio Winer, posteriormente en 2003, el que publicaría RSS 2.0, que junto con RSS 0.91 son las versiones más usadas hoy en día.

Una de las diferencias principales entre RSS 1.0 y el resto es que el primero se basa en el estándar RDF (*Resource Description Framework*) de la W3C (*World Wide Web Consortium*). De hecho, las siglas de RSS fueron reinterpretadas como *RDF Site Summary*. Por otro lado RSS 0.91 y RSS 2.0 son más sencillos como se verá en apartados sucesivos.

En el resto del apartado se van a enumerar las distintas versiones y sus características.

7.3.1 RSS 0.91 Y RSS 0.92

Como se ha mencionado previamente, RSS 0.91 es el estándar RSS más antiguo pero sigue estando en uso en numerosos sitios web.

RSS 0.91 es un dialecto de XML y permite representar un único canal que puede contener hasta 15 elementos o ítems. Cada ítem, a su vez, puede contener otras características como el título, la descripción, el lenguaje, etc. El DTD que recoge la parte más importante de RSS 0.91 es el siguiente:

```
<!ELEMENT rss (channel)>
<!ATTLIST rss version CDATA #REQUIRED>
<!ELEMENT channel (title | description | link | language
    | item+ | rating? | image?
    | textinput? | copyright?
    | pubDate? | lastBuildDate?
    | docs? | managingEditor?
    | webMaster? | skipHours?
    | skipDays?)*>

<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT link (#PCDATA)>
<!ELEMENT image (title | url | link |
    width? | height? | description?)*>
<!ELEMENT url (#PCDATA)>
<!ELEMENT item (title | link | description)*>
<!ELEMENT textinput (title | description | name | link)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rating (#PCDATA)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT width (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT copyright (#PCDATA)>
<!ELEMENT pubDate (#PCDATA)>
<!ELEMENT lastBuildDate (#PCDATA)>
<!ELEMENT docs (#PCDATA)>
<!ELEMENT managingEditor (#PCDATA)>
<!ELEMENT webMaster (#PCDATA)>
<!ELEMENT hour (#PCDATA)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT skipHours (hour+)>
<!ELEMENT skipDays (day+)>
```

Se puede observar que en RSS 0.91 es obligatorio definir un canal. Por otro lado, existen cinco elementos obligatorios a la hora de definir un canal (*title*, *link*, *description*, *language* e *image*) mientras que también se pueden definir otros opcionales como *copyright* o *rating*.

Una vez definido el canal, se pueden definir elementos o ítems. En RSS 0.91 se pueden definir hasta 15 ítems. Los ítems son los que definen el contenido que se quiere sindicar mediante los subelementos obligatorios *title* (título del contenido), *link* (URL del contenido completo) y el subelemento opcional *description* (descripción o resumen del contenido).

Siguiendo este DTD, a continuación, se muestra un ejemplo de *feed RSS 0.91* de una agencia de noticias en español ficticia.



EJEMPLO 7.1

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
<channel>
    <title>Agencia de noticias AGNE </title>
    <link>http://agenciaagne.es</link>
    <description>Noticias de español</description>
    <language>es-es</language>
    <copyright></copyright>
    <managingEditor>
        ed@agenciaagne.es (Jose Luis Lopez)
    </managingEditor>
    <webMaster>webmaster@agenciaagne.com</webMaster>
    <pubDate>Thu, 03 Mar 11 07:00:00 GMT</pubDate>
    <image>
        <title>WriteTheWeb</title>
        <url>http://agenciaagne.es/imagen.gif</url>
        <link>http://agenciaagne.es</link>
        <width>100</width>
        <height>200</height>
        <description>Noticias en español </description>
    </image>
    <item>
        <title>Previsión de lluvia en todo el país</title>
        <link>http://agenciaagne.es/?noticiaId=512</link>
        <description>
            La agencia estatal de meteorología ...
        </description>
    </item>
    <item>
        <title>
            El número de accidentes de tráfico ha decrecido
            en el último año
        </title>
        <link>http://agenciaagne.es/?noticiaId=599</link>
        <description>
            La DGT ha publicado los datos de accidentes ...
        </description>
    </item>
</channel>
</rss>
```

ACTIVIDADES 7.1



- Añada dos nuevas noticias al *feed* del ejemplo anterior.

El estándar RSS 0.92 es muy similar que el RSS 0.91 y, por tanto, permite representar un único canal, pero, en este caso, puede manejar un número ilimitado de ítems y soporta unos metadatos más completos y a nivel de ítem. Además, elimina una restricción presente en RSS 0.91 que limita el número de caracteres que pueden contener los elementos. Por otro lado, algunos elementos se convierten en opcionales mientras que aparecen algunos nuevos para enriquecer el contenido:

- **source**: para relacionar ítems con otros ítems de los que derivan.
- **enclosure**: para definir un archivo asociado a un ítem.
- **category**: para definir jerarquías de categorías. La jerarquía se representa separando las categorías con una barra invertida (por ejemplo Categoría1/Categoría1.1/Categoría1.1.1).
- **cloud**: sirve para definir los procesos de publicación y suscripción (se verán en detalle más adelante).

ACTIVIDADES 7.2



- Busque el DTD correspondiente a RSS 0.92 y compruebe las diferencias entre RSS 0.91 y RSS 0.92.
- Reescriba el RSS 0.91 presentado en el ejemplo anterior, usando RSS 0.92, añadiendo posibles metadatos que se consideren oportunos.

7.3.2 RSS 1.0

El RSS 1.0 es relativamente diferente al RSS 0.91 y RSS 0.92 por las razones históricas expuestas en los apartados previos. La principal diferencia radica en el uso de tecnologías asociadas a XML más avanzadas. Si bien la estructura XML de RSS 0.91 y RSS 0.92 se encuentra especificada mediante DTD, los documentos XML de RSS 1.0 hacen uso de esquemas, espacios de nombres y RDF, lo que provoca que estos documentos sean mucho más complejos pero, a su vez, permiten una mayor expresividad para usar metadatos. Además, el uso de esquemas y espacios de nombres permite una mayor extensibilidad y versatilidad.

A continuación, se muestra el mismo ejemplo usado para RSS 0.91 pero esta vez usando RSS 1.0:



EJEMPLO 7.2

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="http://agenciaagne.es/noticias.rss">
    <title>Canal de noticias AGNE</title>
    <link>http://agenciaagne.es/</link>
    <description>Noticias en español</description>
    <items>
      <rdf:Seq>
        <rdf:li
          resource="http://agenciaagne.es/2010/12/01"/>
        <rdf:li
          resource="http://agenciaagne.es/2010/12/05"/>
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="http://agenciaange.es/2010/12/01">
    <title>Previsión de lluvia en todo el país </title>
    <link> http://agenciaange.es/2010/12/01/</link>
    <description>
      La agencia estatal de meteorología ...
    </description>
    <dc:date>2010-12-01</dc:date>
  </item>
  <item rdf:about="http://agenciaange.es/2010/12/05">
    <title>
      El número de accidentes de tráfico ha decrecido
      en el último año
    </title>
    <description>
      La DGT ha publicado los datos de accidentes ...
    </description>
    <link>http://agenciaange.es/2010/12/05/</link>
    <dc:date>2010-12-05</dc:date>
  </item>
</rdf:RDF>
```

En este ejemplo se puede observar que en la primera etiqueta se definen los espacios que hacen que el documento sea más complejo pero a la vez más flexible y versátil.

Un feed RSS 1.0 debe comenzar con la línea:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
```

Y contener, al menos, un canal, que se define usando:

```
<channel rdf:about="URI_IDENTIFICADOR">
  ELEMENTOS
</channel>
```

donde *URI_IDENTIFICADOR* es un URI que sirve de identificador único para dicho canal.

Otra diferencia con respecto a RSS 0.91 y RSS 0.92 es que algunos elementos como, por ejemplo, *image* ya no se encuentran dentro del elemento *channel* sino al mismo nivel.

ACTIVIDADES 7.3



- Compruebe que el documento XML del ejemplo es un documento válido y bien formado usando un validador (por ejemplo, <http://validator.w3.org/check>).
- Añada dos noticias más en el feed RSS 1.0.

Tal y como se ha mencionado previamente, RSS 1.0 soporta el uso de módulos para una mayor flexibilidad. A continuación, se muestra un ejemplo de módulo y cómo se usa. Por ejemplo, el modulo *mod_annotation* permite añadir un nuevo elemento donde se puede definir una URL en la que se puede discutir sobre el contenido del ítem al que acompaña. Para usarlo hay que añadir el espacio de nombres correspondiente:



EJEMPLO 7.3

```
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns="http://purl.org/rss/1.0/"  
    xmlns:annotate="http://purl.org/rss/1.0/modules/annotate/">
```

Y se podrá añadir el elemento siguiente:

```
<annotate:reference rdf:resource="URL_DISCUSION"/>
```

ACTIVIDADES 7.4



- Añada el módulo *mod_annotation* y el elemento *annotate* en el ejemplo usado anteriormente y compruebe que el documento XML producido sigue siendo un documento bien formado (por ejemplo, <http://validator.w3.org/check>) y válido (por ejemplo, <http://validator.w3.org/feed/check.cgi>).
- Busque información acerca de otros módulos y su uso, e incorpore alguno al ejemplo propuesto.

7.3.3 RSS 2.0

RSS 2.0 sigue la línea marcada por RSS 0.91 con la particularidad de que su estructura está pensada para proporcionar una mayor modularidad, lo que se consigue mediante la introducción de módulos. A su vez, estas características hacen que este estándar genere documentos más complejos. Al igual que ocurre con RSS 1.0, RSS 2.0, no sigue la línea empezada por RSS 1.0 (el uso de RDF) sino que más bien es una continuación de RSS 0.92 llamándose a veces RSS 0.94 (RSS 0.93 fue un desarrollo que se abandonó al poco tiempo de su gestación).

Además del nuevo uso de módulos, RSS 2.0 incorpora un conjunto de nuevos elementos, como pueden ser los denominados *comments*, *author* y *guid*, entre otros. Este último es recomendable usarlo ya que permite definir una URL única para el ítem al que acompaña mejorando la interoperabilidad entre sistemas. Además, en los elementos que ya existían en RSS 0.91 y RSS 0.92 se han introducido algunos pequeños cambios. Por ejemplo, el elemento *image* ahora es opcional.

La Figura 7.2 muestra el árbol con los posibles metadatos que permite almacenar RSS 2.0.

Como se ha mencionado previamente, uno de los cambios más importante de RSS 2.0 frente a RSS 0.92 es la inclusión de módulos. En RSS 1.0 el uso de módulos estaba basado en incluir la definición del espacio de nombres correspondiente en la declaración del elemento raíz. Sin embargo, en RSS 2.0 no se usa RDF y, por tanto, es algo distinta, pero al igual que RSS 1.0, se basa en definir espacios de nombre en el elemento raíz.

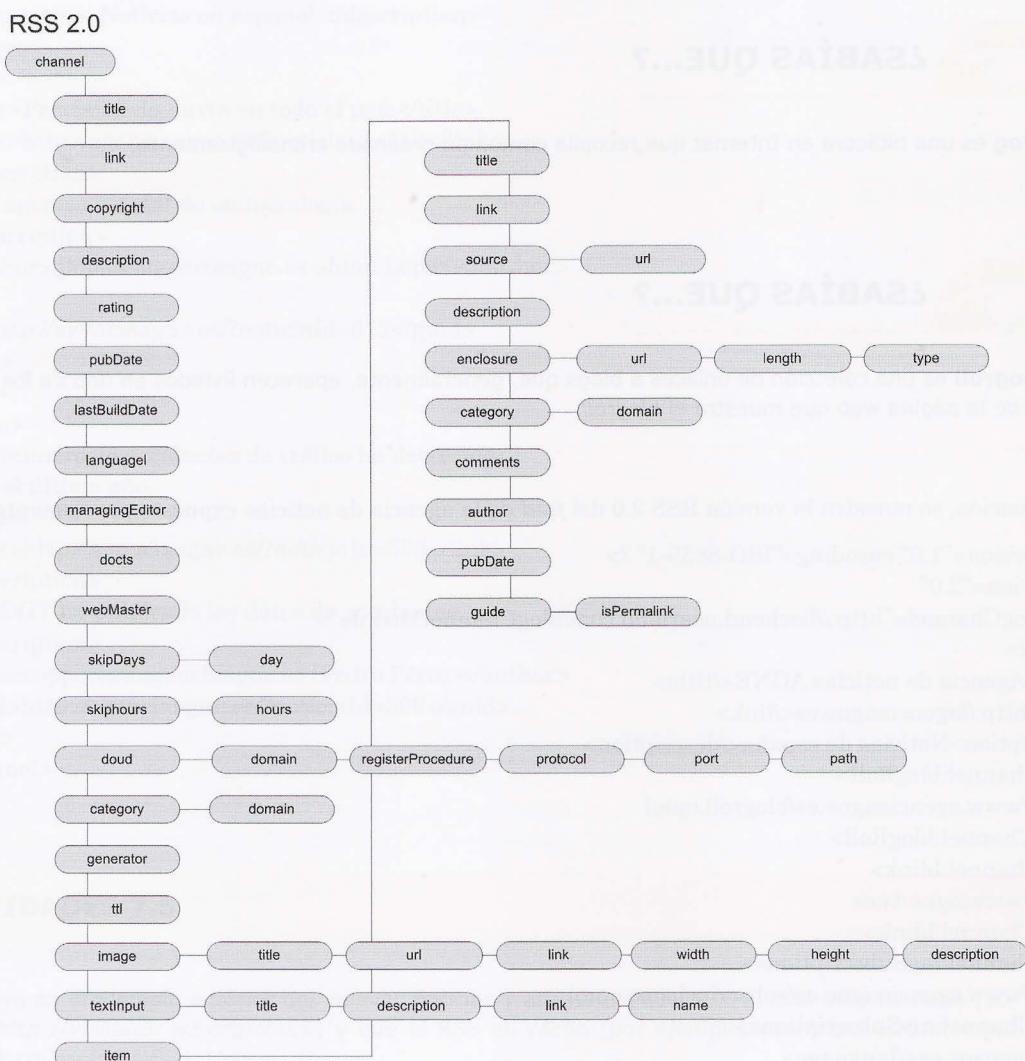


Figura 7.2. Árbol de metadatos en RSS 2.0

El primer módulo que Dave Winer (la principal persona responsable del estándar RSS 2.0) desarrolló fue *blogChannel* para permitir incluir información en el *feed* acerca de blogs. Para poder usarlo, hay que incluir su espacio de nombres en el elemento raíz de la siguiente forma:

```
<rss version="2.0" xmlns:blogChannel="http://backend.userland.com/blogChannelModule">
```

Una vez incluido el espacio de nombres, se podrán usar tres nuevos elementos:

- blogChannel:blogRoll: contiene una URL que apunta al *blogroll* del sitio.
- blogChannel:blink: contiene una URL a un sitio web que el autor considera interesante.
- blogChannel:mySubscriptions: contiene una URL al fichero que contiene los feeds RSS a los que el autor del sitio web está suscrito.



¿SABÍAS QUE...?

Un **blog** es una bitácora en Internet que recopila contenido ordenado cronológicamente.



¿SABÍAS QUE...?

Un **blogroll** es una colección de enlaces a blogs que, generalmente, aparecen listados en uno de los lados de la página web que muestra el *blogroll*.

A continuación, se muestra la versión RSS 2.0 del *feed* de la agencia de noticias expuesto previamente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0"
      xmlns:blogChannel="http://backend.userland.com/blogChannelModule">
<channel>
    <title>Agencia de noticias AGNE</title>
    <link>http://agenciaagne.es</link>
    <description>Noticias de España</description>
    <blogChannel:blogRoll>
        http://www.agenciaagne.es/blogroll.opml
    </blogChannel:blogRoll>
    <blogChannel:blink>
        http://www.agne-tv.es
    </blogChannel:blink>
    <blogChannel:mySubscriptions>
        http://www.agenciaagne.es/subscripciones.opml
    </blogChannel:mySubscriptions>
    <language>es-es</language>
    <copyright></copyright>
    <managingEditor>
```

```
ed@agenciaagne.es (Juan Lopez)
</managingEditor>
<webMaster>
  webmaster@agenciaagne.com (Webmaster)
</webMaster>
<pubDate>Thu, 03 Mar 2011 07:00:00 GMT</pubDate>
<image>
  <title>Agencia de noticias AGNE</title>
  <url>http://agenciaagne.es/imagen.gif</url>
  <link>http://agenciaagne.es</link>
  <width>100</width>
  <height>200</height>
  <description>Noticias en español </description>
</image>
<item>
  <title>Previsión de lluvia en todo el país</title>
  <link>http://agenciaagne.es/?noticiaId=512</link>
  <description>
    La agencia estatal de meteorología ...
  </description>
  <author>jlopez@agenciaagne.es (Juan López)</author>

<guid>http://agenciaagne.es/?noticiaId=512</guid>
</item>
<item>
  <title>
    El número de accidentes de tráfico ha decrecido
    en el último año
  </title>
  <link>http://agenciaagne.es/?noticiaId=599</link>
  <description>
    La DGT ha publicado los datos de accidentes ...
  </description>
  <author>pperez@agenciaagne.es (Pedro Pérez)</author>
  <guid>http://agenciaagne.es/?noticiaId=599</guid>
</item>
</channel>
</rss>
```

ACTIVIDADES 7.5

- Incluya en el ejemplo anterior dos nuevas noticias y asegúrese que el XML está bien formado (por ejemplo, con <http://validator.w3.org/check>) y que el RSS es válido (por ejemplo, con <http://validator.w3.org/feed/check.cgi>).
- Busque información acerca de otros módulos existentes.

7.4 SISTEMAS DE AGREGACIÓN Y DIRECTORIOS DE CANALES

Actualmente, se puede encontrar en Internet un número muy elevado de *feeds* RSS, por ejemplo, *feeds* de noticias, blogs, etc. Si un usuario desea poder leer todos estos *feeds* se hace necesario contar con una herramienta que le facilite la tarea. Aquí es donde entran en juego los sistemas de agregación que permiten mostrar de forma conjunta los contenidos publicados en distintos *feeds*.

Por otro lado, dado el volumen de información que se encuentra disponible, resulta muy interesante el uso de algún tipo de catálogo que permita clasificar y realizar búsquedas sobre los contenidos de distintos *feeds*. A este tipo de herramientas se les denomina **directorios de canales**.

Estas dos herramientas pueden ser tanto online como herramientas para escritorio.

ACTIVIDADES 7.6



- Encuentre agregadores y directorios de canales en Internet, cree una cuenta, añada *feeds* y analice el comportamiento de los mismos.



RESUMEN DEL CAPÍTULO



En este capítulo se ha presentado el concepto de sindicación de contenidos, así como el ciclo de vida de los contenidos dentro de los sistemas de gestión de contenidos, denominados CMS.

A continuación, se han expuesto los estándares RSS más usados así como sus características principales y ejemplos de los mismos, detallando RSS 0.91, RSS 0.92, RSS 1.0 y RSS 2.0.

Por último, se han definido los agregadores de contenidos y los directorios de canales.



EJERCICIOS PROPUESTOS



Dada la siguiente web de recetas de cocina:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Recetas de cocina</title>
</head>
<body>
<h1>Mis recetas de cocina favoritas</h1>
<h2>
<a href="/rec/polloallimon">Pollo al limón</a>
</h2>
<div>3 de febrero de 2010</div>
<p>

Para preparar esta receta, lo ingredientes son ...
</p>
<h2>
<a href="/rec/albondigas">Albondigas con tomate
</a></h2>
<div>25 de enero de 2010</div>
<p>

Los ingredientes necesarios para esta receta son ...
</p>
<h2>
<a href="/rec/paella">Paella valenciana</a></h2>
<div>13 de enero de 2010</div>
<p>

Esta receta empieza ...
</p>
</body>
</html>
```

- 1. Generar el RSS 0.91 (RSS 0.92) asociado a dicha contenido y validarla con un validador de RSS como el de W3C. Si se considera necesaria una URL completa para el sitio usar una ficticia.
- 2. Generar el RSS 1.0 a partir de la web propuesta o partir del RSS 0.91 o RSS 0.92 generado en el apartado anterior. Realizar la validación correspondiente.
- 3. Generar el RSS 2.0 a partir de la web propuesta o partir del RSS 0.91 o RSS 0.92 generado en el primer apartado. Realizar la validación correspondiente.



TEST DE CONOCIMIENTOS



1 La sindicación de contenidos está pensada para:

- a) Blogs.
- b) Cualquier sitio web con contenido.
- c) Sitios de noticias.
- d) La sindicación no es aplicable para sitios web.

3 De los estándares RSS, ¿cuál hace uso de RDF?

- a) RSS 1.0 y RSS 2.0.
- b) RSS 0.91 y RSS 0.92.
- c) RSS 2.0.
- d) RSS 1.0.

2 Las siglas CMS se refieren a:

- a) Un sistema de gestión de contenidos.
- b) Un sistema de sindicación de contenidos.
- c) Un sistema de agregación de canales.
- d) Un directorio de canales.

4 Un agregador de contenidos:

- a) Proporciona un punto de entrada único a un conjunto de feeds.
- b) Es un gestor de contenidos.
- c) Sirve para poder sindicar los contenidos de un sitio.
- d) Ninguna de las anteriores es correcta.

8

Sistemas de gestión empresarial

OBJETIVOS DEL CAPÍTULO

- ✓ Saber qué es un ERP.
- ✓ Identificar qué características tiene un ERP.
- ✓ Conocer que sistemas ERP comerciales y OpenSource hay disponibles.
- ✓ Identificar las ventajas e inconvenientes de la implantación de un ERP.
- ✓ Riesgos de seguridad que deben tenerse en cuenta antes, durante y después de la implantación de un sistema informático.
- ✓ Cómo importar y exportar en un sistema ERP genérico.

8.1 INTRODUCCIÓN A LOS ERP

Para que una empresa pueda ser eficiente en sus procesos productivos, necesita ser consciente de todos sus recursos empresariales y gestionarlos de manera eficaz. Sin embargo no todas las empresas son capaces de alcanzar esa eficiencia.

Imaginemos a una persona que posee una pequeña empresa dedicada a la venta de calzado deportivo y zapatos. No tiene contratado a ningún empleado puesto que no lo considera necesario. Por tanto todo el trabajo recae sobre el dueño. Los objetivos empresariales, el modelo de negocio y el mercado potencial se centraría en la venta de todo su stock de calzado a los vecinos y, en menor medida, a los visitantes que pasan por el barrio de manera esporádica.

En este ejemplo, el dueño debe vender y facturar el calzado a los clientes en su horario de venta al público. Al finalizar dicho horario, debe dedicarse a actualizar el inventario de existencias en el almacén, llamar a los proveedores y realizar nuevos pedidos. Además, una vez al mes debe poner en orden todas las cuentas para realizar el cálculo de los impuestos a pagar al Estado, pago de proveedores, luz, agua, teléfono, etc.

Parece claro que existen muchas tareas que se deben realizar para mantener en funcionamiento la empresa. Sin embargo, al ser una empresa pequeña, el propietario puede asumirlas de manera manual. ¿Qué pasaría si el negocio le va bien y quiere expandir su radio de acción? Simplemente el dueño no podrá desdoblarse en el espacio-tiempo y requerirá contratar personal para atender las nuevas ubicaciones. Quizás pueda asumir seguir llevando las cuentas pero los inventarios se duplicarían y empezaría a perder mucho tiempo en la gestión de la propia empresa.

En esos casos en los que la envergadura de una empresa es tal que se requieren tener controlados todos los recursos empresariales, es necesario recurrir a algún sistema automatizado de gestión y administración empresarial. Actualmente estos sistemas se llaman **ERP** (en inglés *Enterprise Resource Planning*), y se especializan en manejar todo el conjunto de datos que son relevantes para la continuidad de la empresa.



¿SABÍAS QUE...?

Un **CRM** (Gestor de Relaciones con los Clientes) es un ERP especializado en la relación con los clientes que tiene una empresa.

ACTIVIDADES 8.1



- Buscar por Internet qué relación tiene un sistema ERP y un CRM (*Customer Relationship Management*).
- ¿Cómo un CRM puede complementar a un ERP? ¿Por qué?

8.2 COMPOSICIÓN DE UN ERP

Un ERP consta de multitud de módulos que se pueden interconectar entre sí. De esta manera una única herramienta ERP puede dar servicio a empresas muy distintas cambiando el conjunto de módulos activos y las relaciones entre ellos.

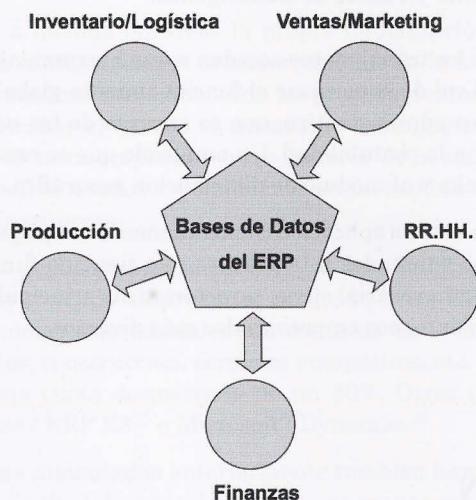


Figura 8.1. Diagrama de módulos básicos en un ERP

Los módulos, por tanto, son la parte central de una herramienta ERP. Estos módulos permitirán almacenar, buscar, mostrar y representar cada proceso interno de la empresa. Una herramienta ERP se puede componer por el siguiente conjunto básico de módulos activos:

- **Finanzas:** Base del ERP. Almacenamiento de cada transacción y su impacto administrativo. Facilita las auditorías.
- **Producción:** Núcleo que se encarga de los movimientos físicos de los artículos, planificación de los materiales, etc.
- **Inventario y Logística:** Stock, almacenes, flujos de entrada y salida, etc.
- **Ventas y Marketing:** Interfaz de interacción con los clientes, trazabilidad de los pedidos, estrategias de ventas y rentabilidad, precios y promociones, etc.
- **Recursos Humanos:** Gestión de personal, nóminas, aplicación de normativas legales, repartición de beneficios, productividad, bonus e incentivos.

Es claro que no todas las empresas se pueden definir completamente con los anteriores módulos básicos. Por ello, los sistemas ERP se complementan con otros módulos como los siguientes:

- Previsión de ventas.
- Mantenimiento.
- Gestión de cambios del producto (PDM o *Product Data Management*).
- Configuración de productos a medida.
- Gestión de relaciones con los clientes (CRM o *Customer Relationship Management*).
- Controles de planta y de almacenes.
- Seguridad, Planes de Continuidad y Planes de Contingencia.

Es importante indicar que no todos los trabajadores acceden a una herramienta ERP de igual manera. Es probable que el gerente de la empresa tenga el rol de supervisar el funcionamiento global de la empresa, pudiendo acceder a todos los módulos del sistema. Pero un administrativo, que se encarga de las nóminas, solo debería tener acceso al módulo que da acceso a las nóminas y a la contabilidad. Un empleado que se encarga de los envíos por la ciudad solo necesitará acceso a los almacenes, stocks y al módulo de distribución geográfica.

Como se puede intuir, un ERP puede ser una aplicación tremadamente compleja, con muchos módulos personalizados y con un conjunto de roles y permisos adaptados a la manera que tiene de funcionar cada empresa. Por tanto, un ERP será una herramienta de gestión empresarial cuyas características principales serán la MODULARIDAD de sus sistemas y su ADAPTABILIDAD a los entornos empresariales más diversos.

8.3 IMPLANTACIÓN

Instalar y configurar una solución ERP requiere de muchísimo conocimiento interno de la propia empresa. Pero ese coste temporal y económico se verá ampliamente recompensado porque permite obtener un seguimiento íntegro de la empresa una vez implantado. Principalmente se consiguen las siguientes ventajas en una implementación de un ERP:

- Detección de puntos débiles en la gestión empresarial.
- Optimización de los procesos que se desarrollen en la empresa.
- Información centralizada, actualizada y coherente con los procesos internos que se ejecuten en cada momento.
- Acceso a toda la información de la empresa de manera modular y basada en roles.
- Compartición de la información relevante para cada uno de los procesos que lo requieran.
- Reducción de los costes asociados y reducción de tiempos.
- Análisis del estado de la empresa desde una visión global.

Aunque las ventajas son muy atractivas, también se observan algunas desventajas que se deben analizar antes de integrar una solución ERP en la empresa:

- Es necesario tener integrado en los costes de implantación, los recursos necesarios para formar a cada trabajador y su reciclaje continuo.
- Se deben identificar perfectamente los roles de cada uno de los trabajadores que operen con el ERP. La asignación correcta de permisos evitará la propagación de errores y las fugas de datos.
- La instalación y puesta en marcha de un sistema ERP es costosa económicamente y en tiempo.

- El acceso a recursos centralizados puede verse como un cuello de botella. El ERP debe dimensionarse para evitar que las bases de datos sean un punto de fallo crítico.
- Hay que identificar procesos que utilicen datos comunes de manera concurrente. No hay que dejar que se "corrompan" con valores antiguos o inexactos con el estado actual de la empresa.
- Posiblemente los programas antiguos de gestión sean incompatibles con los modelos de datos del ERP.
- Si la empresa es grande, el hecho de compartir información entre departamentos puede resultar contraproducente.

A veces el coste de implantarlo a medida encarece la propia implantación por lo que se recomienda intentar adaptar los procesos de la empresa a configuraciones predeterminadas de la herramienta ERP. Aunque las ventajas son puramente económicas, hay que analizar que cambios en los procedimientos de trabajo que tienen automatizados los propios empleados pueden determinar el éxito o el fracaso de la implantación. Una política de transparencia, información y recepción de temores de estos empleados a los gestores de la empresa puede resultar tremadamente positiva, implicando de manera proactiva a todos los empleados en un cambio tan radical en la propia concepción de la información y su tratamiento en la empresa.

A la hora de elegir un sistema de gestión empresarial ERP, se debe analizar las diferentes soluciones existentes en el mercado. En los últimos años el tejido empresarial dedicado a los ERP ha crecido de manera sustancial, pudiendo adaptar los módulos a modelos empresariales tan distintos como la administración pública, sistemas de telecomunicaciones, centros sanitarios, construcción, servicios energéticos, etc. El mayor proveedor de herramientas ERP, a día de hoy, es SAP²³ con una cuota de mercado de un 30%. Otros proveedores comerciales son Oracle²⁴, E-Business Suite²⁴, Sage²⁵ Logic Class / ERP X3²⁵ o Microsoft²⁶ Dynamics²⁶.

Otras corporaciones distintas a las comentadas anteriormente también han visto en los ERP el modelo ideal para incrementar su productividad. La principal desventaja de estas pequeñas empresas es que no pueden asumir una implantación a medida de toda su organización por motivos meramente presupuestarios. Aunque es posible acceder a ERP comerciales genéricos de coste económico más reducido, estas pequeñas empresas comenzaron a analizar las alternativas abiertas u OpenSource. Efectivamente el mundo del software libre también posee herramientas ERP que no tienen nada que envidiar a las más completas de las comerciales. Como ejemplos de ERP OpenSource podemos nombrar a los siguientes:



Para obtener más información, vea la página [http://www.informatica-hoy.com.ar/software-erp/
Sistemas-ERP-de-software-libre.php](http://www.informatica-hoy.com.ar/software-erp/Sistemas-ERP-de-software-libre.php).

- **Fisterra**²⁷: ERP dedicado a las reparaciones de garajes y talleres mecánicos. Tiene módulos de clientes, reparaciones, facturación, almacén, caja, contabilidad y gestión documental.

23 <http://www.sap.com>

24 <http://www.oracle.com/es/products/applications/ebusiness>

25 <http://www.sagelogiccontrol.com/websage/empresa.aspx>

26 <http://www.microsoft.com/dynamics>

27 <http://www.fisterra.org>

- **OpenBravo**²⁸: es un ERP basado en software libre que permite acceder a sus funcionalidades a través de un navegador vía Internet, con una parametrización basada en necesidades (casi sin programación), y cuyo coste es muy reducido integrando una implantación “en la nube” o en propios servidores de la empresa (Windows o GNU/Linux).
- **AbanQ**²⁹: basado en el ERP anteriormente denominado FacturaLUX. OpenSource español iniciado desde 2001 y que ha llegado a considerarse uno de los mejores ERP. La empresa que lo liberó y que lidera su desarrollo, InfoSIAL S.L. lo pone disponible tanto para Windows como para GNU/Linux. El modelo de negocio de la empresa es la adaptación, creación de extensiones y la formación sobre el software a las empresas que lo requieren.
- **OpenERP**³⁰: anteriormente llamado TinyERP. Es un sistema muy modular que permite comenzar con una versión muy simple e ir agregando módulos a medida que el usuario los vaya necesitando. Su flexibilidad es referenciada debido a que su configuración se puede realizar mediante una interfaz gráfica intuitiva y sencilla.

■ **PenXpertya.**



Figura 8.2. Portal web de un entorno OpenERP

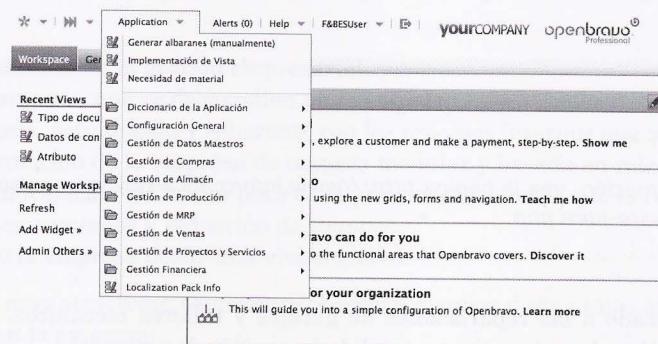


Figura 8.3. Portal web de un entorno OpenBravo

28 <http://www.openbravo.com>

29 <http://www.infosial.com>

30 <http://www.openerp.com>

La elección de un ERP comercial u OperSource dependerá de muchos factores. Si bien es cierto que los sistemas comerciales ofrecen un soporte continuado de la aplicación, se han de analizar fríamente todos los costes de adaptación, garantías, renovación de mantenimientos, etc. No hay mayor error que elegir un ERP y darse cuenta que realiza multitud de funciones salvo la que realmente se quiere. Por ello normalmente se recomienda a las pequeñas empresas que elijan aquel que se adapte a la mayoría de sus necesidades sin que les penalice con cargas burocráticas elevadas por los procedimientos implantados.



Para obtener más información, vea la página <http://www.openerpsite.com/erp-openerp-modulos/metodologia-y-tecnologia>.

ACTIVIDADES 8.2



- Elegir dos sistemas ERP de los nombrados anteriormente (ya sea comercial u OpenSource) y documentarse indicando en una tabla al menos lo siguiente:
- Qué módulos tienen cada uno.
 - Qué requisitos tienen cada uno.
 - Coste, si es comparable.

8.4 SEGURIDAD

En todo entorno empresarial que haya informatizado sus procesos, debe existir un sistema que permita conservar y mantener los datos de manera integra durante todo su período útil. Se debe, por tanto, evitar que se produzcan una serie de riesgos que comprometan la estabilidad del sistema y la propia continuidad del negocio empresarial.

Para aumentar la seguridad de los sistemas se han de analizar los riesgos a los que puede estar sometido nuestro entorno informatizado (ya sea ERP o no). Por ello, se podrían clasificar los riesgos de la siguiente manera:

- **Riesgos físicos:** son aquellos que pueden suceder cuando fallan algunos componentes electrónicos de nuestro sistema informático. Fallos como la rotura de los discos duros, memorias, pueden ser los más habituales. Las causas por las que pueden fallar son muy variadas como su fin de su vida útil y las agresiones externas (altas temperaturas de la sala, incendios, inundaciones, explosiones o incluso robo de componentes).
- **Riesgos lógicos:** son aquellos que pueden suceder cuando hay una política inadecuada de autenticación en los sistemas informáticos, accesos no autorizados, bugs y errores en el sistema operativo o en el software utilizado, intrusiones externas a través de la red o la instalación de software incompatibles o que tengan una funcionalidad oculta (troyano) que limite el correcto funcionamiento del sistema informático.

Una correcta gestión de los riesgos a los que puede estar sometido un sistema informático permite mantener una continuidad de la línea de negocio ante cualquier contingencia que suceda. Por ello es necesario realizar la identificación de los riesgos durante todo el período útil del sistema informático (antes, durante y después de la instalación de un sistema informático).

Ante los riesgos físicos, se deben ubicar los sistemas informáticos en habitaciones acondicionadas para su función. Deben ser climatizadas para evitar el sobrecalentamiento de los sistemas. No deben estar en las partes bajas de un edificio o si lo están, que una inundación no pueda limitar su uso. Debe costar de sistemas de detección contra incendios, con medidas preventivas y correctivas. El control de acceso a la sala debe estar regulado y protegido para evitar robos y hurtos de material. Y sobre todo, una política de copias de seguridad que permita salvaguardar los datos importantes ante una catástrofe no contemplada.

En el ámbito de los ERP, los datos que almacenan las bases de datos deberían ser la principal fuente de preocupación. Una política de copias de seguridad continuada que permita almacenarse en otras ubicaciones distintas a las del edificio de la empresa, permitirá restaurar el sistema en cualquier otro sitio con los datos salvados hasta el momento.



¿SABÍAS QUE...?

En accidentes como el incendio del 2005 en el edificio Windsor de Madrid, o el ataque terrorista a las torres gemelas el 11 de septiembre de 2001 en Nueva York, aún siendo catástrofes poco previsibles, aquellas empresas que almacenaban sus backups en el propio edificio se vieron abocadas al cierre.



Figura 8.4. Incendio del Windsor en Madrid



Figura 8.5. 11-S en Nueva York

ACTIVIDADES 8.3



- Realizar una tabla donde se indiquen al menos diez riesgos físicos y otros diez lógicos.
- Completar la tabla con las acciones que se pueden realizar para evitarlos, o si no se puede, minimizarlos.
- ¿Qué es un plan de contingencia? Buscar información sobre ello.



Para obtener más información, vea la página <http://mariblauditoria.blogspot.com/2010/02/plan-de-contingencia.html>.

-
- Búsqueda de información sobre el incendio del edificio Windsor y las razones por las que se produjo el incendio.



Para obtener más información, vea la página <http://www.rankia.com/foros/seguros/temas/113238-incendio-windsor-200-millones-juego>.

8.5 IMPORTACIÓN Y EXPORTACIÓN DE INFORMACIÓN

El almacenamiento de la información empresarial en un ERP permite tener de manera centralizada toda la lógica de negocio. Una vez establecidas la instalación y las políticas de seguridad, los usuarios deberán utilizar este sistema y no el sistema anterior para mantener la coherencia de los datos y la continuidad del negocio. En algunos momentos se necesitará realizar informes sobre nóminas o facturación por ejemplo, por lo que el propio ERP deberá proporcionar dichos datos de manera fácil y sencilla a través de formularios predefinidos.

Pero puede darse el caso de que no exista un formulario concreto que muestre lo que el usuario necesita puntualmente. Esto puede suceder cuando se informatiza todo un sistema con un ERP y se viene de un sistema anterior que funcionaba de manera manual. Los gerentes en estos casos ven que el sistema puede proporcionar cualquier información para optimizar los recursos empresariales y solicitan a los usuarios ERP dicha información (si ellos mismos no la pueden obtener por sus permisos). Si dicho formulario de extracción de información no existe, podrá desencantar a los gerentes con el sistema implantado. Quizás el conjunto de módulos contratados o activos en ese momento no permitan esas consultas.

Para solucionar este problema siempre se puede generar formularios nuevos que consulten a las bases de datos del sistema ERP lo necesario para generar el informe solicitado. La creación, por tanto, de nuevos formularios se antoja fundamental para flexibilizar el acceso y extracción de información empresarial.

No todos los sistemas ERP tienen la misma forma de generar formularios y resultaría tremadamente extenso indicar en este libro cómo realizarlo para cada ERP concreto. Por ello vamos a indicar una serie de técnicas de extracción de información que pueden resultar independientes del sistema ERP implantado. Las maneras de extracción de información podrían ser:

- Conectores (importación/exportación).
- Exportación a documentos ofimáticos.
- Importación/exportación en ficheros XML.
- Importación/exportación en ficheros CSV.

Los conectores son pequeños módulos software que permiten acceder a las bases de datos de los sistemas ERP. Dependiendo del ERP que se utilice, la programación de estos conectores se realizará en un lenguaje de programación como puede ser C/C++, Python, Shell, Java, etc. La creación de un conector requiere de experiencia en dicho lenguaje de programación y, sobre todo, conocimiento de la arquitectura y estructura del propio ERP. Como parece obvio, un conector que se desarrolle para un ERP no tiene porqué funcionar en otro ERP distinto pero la programación *ad-hoc* del conector que se necesita permitirá ser más agiles a la hora de generar documentación no contemplada en la implantación inicial. Además permite adaptar el sistema ERP en cualquier momento a nuevas necesidades.

Cabe destacar que si un pequeño conector programado por un usuario del ERP se publica como OpenSource, cualquier usuario de otras empresas que usen el mismo ERP podrían utilizarlo. Esto genera una corriente muy positiva donde las necesidades de unos pueden ser cubiertas con las necesidades de otros. El coste podría ser cero. De esta manera unos y otros se pueden beneficiar de un movimiento como puede ser el Open Source.

En relación a la exportación a documentos ofimáticos, todo dependerá del ERP concreto implantado y si posee esta característica de exportación de información. Muchos de ellos permiten, de un formulario del ERP, generar un documento tipo procesador de textos que contiene el informe solicitado. Si el informe que se quiere obtener no es completo, siempre se puede partir de uno parcial y manualmente completarlo con los datos que no exporta inicialmente. También se pueden solicitar datos en bruto, tal y como se haría con una base de datos normal, generando un documento tipo hoja de cálculo donde trabajar posteriormente para extraer la información buscada.

En los últimos tiempos los sistemas ERP suelen utilizar importadores de otros sistemas ERP para hacer una transición más sencilla. Estos importadores suelen hacer una conversión previa a XML donde reconocer entidades y atributos que le son conocidos. Al fin y al cabo la estructura de la información a almacenar en una nómina o en una factura suele ser tremadamente similar de un sistema ERP a otro. Si el sistema ERP puede importar y exportar a XML nada impide utilizar esos datos exportados para generar una hoja XSL, como se vió en el capítulo 5, y aplicar las transformaciones necesarias para obtener la información requerida.

En lo relativo a la exportación CSV (o *Comma-Separated Value*), es ampliamente utilizado en años anteriores. Cualquier fichero CSV podría ser leído simplemente con una hoja de cálculo. A partir de ahí, el simple conocimiento de creación de formulas o scripts podría facilitar la extracción de la información que inicialmente no nos permite extraer un formulario predeterminado del ERP.

A modo de ejemplo, OpenERP permite la exportación de datos e informes mediante hojas de cálculo y CSV. Además permite los siguientes métodos:



Para obtener más información, vea las páginas <http://www.openerpsite.com/openerp-preguntas-frecuentes/1118.html> y http://www.openerpsite.com/tag/openerp-jasper_reports.

- Modificación de los ficheros “RML”. Estos ficheros contienen un informe que puede extraer información del sistema ERP. Si conocemos el fichero “RML” que más que se aproxima a lo que se quiere obtener, el simple cambio podría generar un nuevo informe acorde a nuestras necesidades. Para ello se requiere altos conocimientos de programación y de la estructura del fichero pero es el que ofrece mejores resultados y permite modelar el ERP a lo que se quiera en cada momento.
- Modificación de los ficheros “SXW” que se utilizan para generar los informes. Si los cambios son mínimos, como los estilos del documento, formatos, colores, negritas, etc., esta plantilla OpenOffice se puede modificar mostrando resultados de manera transparente para el usuario. Se requiere tener conocimientos de la herramienta ofimática OpenOffice y del módulo ERP que accede a este tipo de documentos.
- Utilización de herramientas de generación de informes tipo Jasper/ireports. Es una herramienta que permite definir informes nuevos a medida mediante un interfaz gráfico bastante intuitivo. Es una herramienta similar a la comercial Crystal Reports. Para poder utilizar este gestor de informes es necesario integrar un nuevo módulo en OperERP llamado “jasper_reports”.

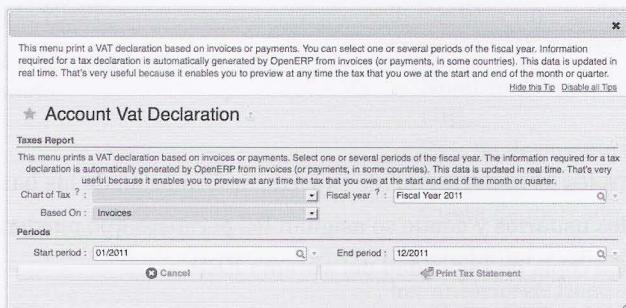


Figura 8.6. Formulario OpenERP para informe de impuestos

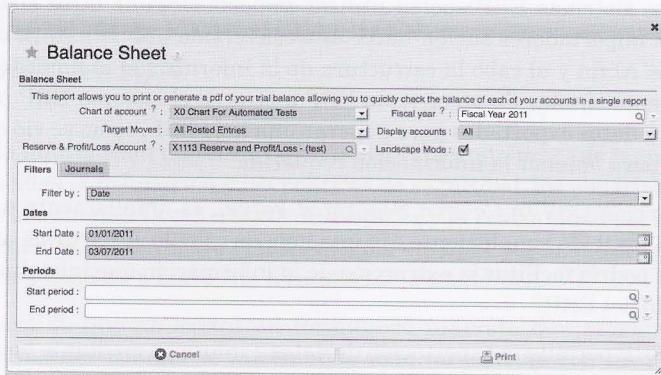


Figura 8.7. Formulario OpenERP para informe de balance de cuentas

Para finalizar, a veces un usuario requiere integrar parte del sistema ERP con algún módulo que permita utilizarlo conjuntamente con el correo electrónico. Una integración con el gestor de correo permitirá, por ejemplo, definir plantillas de envío para cada uno de los clientes para realizar tareas de *mailing* adjuntando documentación personalizada para cada envío. Para hacer esto posible se requeriría dos módulos: uno que se instale en el propio ERP como un conector; y otro módulo que se instalase en el gestor de correo electrónico que utiliza el usuario que demanda esta funcionalidad. Esto requeriría que existiera un módulo, por ejemplo, para Thunderbird, Outlook, FoxMail o incluso GMail o Hotmail.

8.6 CASO PRÁCTICO

Se propone elegir uno de los ERP OpenSource indicados en este capítulo que permiten la utilización de sus demos *online* a través de la webs (p.ej: OpenBravo³¹, OpenERP³²).

Una vez elegido el ERP, se pide identificar lo siguiente:

- Los recursos hardware que son necesarios para la instalación del sistema ERP elegido.
- Los módulos que tienen activos en su portal web y su ubicación dentro del interfaz gráfico. Al menos se deben identificar los siguientes:
 - Inventario/Logística.
 - Ventas/Marketing.
 - Producción.
 - Recursos Humanos.
 - Finanzas.
- Donde se puede acceder a los formularios donde exportar información de las bases de datos del ERP.
- Donde se dan de alta a los usuarios y donde se asignan los permisos que pueden utilizar en el ERP.
- Donde se puede hacer una copia de seguridad del sistema ERP.

31 <http://demo.openbravo.com>

32 <https://demo.my.openerp.com/openerp>



RESUMEN DEL CAPÍTULO



En este capítulo se ha iniciado en el concepto ERP como sistema automatizado de gestión y administración empresarial. Se han establecido una serie de actividades que permiten al alumno completar su formación buscando las relaciones entre un ERP (Plan de Recursos Empresariales) y un CRM (Gestor de Relaciones con los Clientes).

Para la implantación de un ERP en una empresa es necesario saber qué recursos se quieren controlar a través de él y, por tanto, cuáles son los módulos que hay que activar (módulos básicos + módulos personalizados). Además, se ha hecho hincapié en qué ventajas se pueden obtener en su implantación, sin dejar de lado los principales inconvenientes que pueden surgir.

Se ha hablado de los problemas de seguridad que pueden sufrir de manera genérica un sistema informático genérico y se ha puntualizado en la gran importancia de mantener copias de seguridad del ERP en caso de contingencia (puesto que es el centro de operaciones de cualquier negocio).

Finalmente se han dado pinceladas de cómo se extraería la información de un ERP genérico, ya sea a través de formularios, programación de módulos *ad-hoc*, compatibilidad con herramientas ofimáticas o con exportaciones en documentos XML. Este último podría ser utilizado para un tratamiento informático como el que se ha visto en temas anteriores (SGBD XML nativas, XSL, etc.).



EJERCICIOS PROPUESTOS

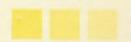


- 1. ¿Cuándo surgió el concepto ERP?
- 2. Realizar una tabla en la que se muestren 5 ERP de tipo OpenSource, mostrando:
 - Nombre del proyecto.
 - URL.
 - País de desarrollo (si procede).
 - Lenguaje de programación (si está disponible).
 - Cuota de mercado (si está disponible).
- 3. Realizar una tabla en la que se muestren 5 vendedores de ERP de tipo comercial, mostrando:
 - Nombre del proyecto.
 - URL.
 - País de desarrollo (si procede).
 - Lenguaje de programación (si está disponible).
 - Cuota de mercado (si está disponible).

En cualquier caso, se deberá analizar si alguno de los anteriores tiene una posición dominante en el mercado y si eso influye en los precios finales para los usuarios.



TEST DE CONOCIMIENTOS

**1**

¿Qué es un ERP?

- a)** Es un sistema de gestión automatizado para la gestión empresarial que permite asegurar la continuidad de la empresa.
- b)** Es un programa informático que solo sirve para las PyMES (Pequeñas y Medianas Empresas).
- c)** Es un programa informático que solo sirve para las grandes empresas.
- d)** Ninguna de las anteriores.

2

¿Qué es un CRM?

- a)** Es un simple ERP para PyMES.
- b)** Es un simple ERP para grandes empresas.
- c)** Es un ERP especializado para la relación entre cliente-empresa.
- d)** Ninguna de las anteriores.

3

Un ERP se compone de:

- a)** Un conjunto de plugins o add-ons que cubren la funcionalidad básica (Inventario, Ventas, Producción, RR.HH., Finanzas).
- b)** Un conjunto roles que cubren toda la funcionalidad básica (Inventario, Ventas, Producción, RR.HH., Finanzas).
- c)** Un conjunto de módulos que cubren la funcionalidad básica (Inventario, Ventas, Producción, RR.HH., Finanzas).
- d)** Ninguna de las anteriores.

4

Al implantar un ERP:

- a)** Se descentraliza la información.
- b)** Los usuarios usan siempre el mismo rol.
- c)** Se evita la compartición de información.
- d)** Ninguna de las anteriores.

5

Al implantar un ERP:

- a)** Los costes de implantación pueden ser elevados.
- b)** El acceso a los recursos pueden ser el cuello de botella.
- c)** A y B son correctos.
- d)** Ninguna de las anteriores.

6

Independientemente de la implantación de un ERP:

- a)** Hay que analizar la seguridad de los sistemas vitales de la empresa, evaluando los riesgos físicos/lógicos y estableciendo un plan de minimización de los mismos.
- b)** Las copias de seguridad ya no son necesarias en un entorno ERP al descentralizar la información y los procesos.
- c)** A y B son correctos.
- d)** Ninguna de las anteriores.

7

La importación/exportación de la información en un ERP genérico se puede realizar:

- a)** Únicamente mediante formularios habilitados para ello.
- b)** Únicamente mediante la exportación a documentos ofimáticos.
- c)** Existen muchas maneras pero destacan los conectores o plugins, módulos de importación/exportación en XML, ficheros CSV y los documentos ofimáticos vinculados al ERP.
- d)** Ninguna de las anteriores.

8

Un administrativo de nóminas tiene acceso al inventario del almacén, ¿sería correcto en un ERP?

- a)** No, porque no necesita los datos del almacén para su trabajo.
- b)** Sí, porque podría necesitar esos datos para su trabajo.
- c)** Depende. En teoría sería un mal uso de los roles si no necesitase esos datos del almacén.
- d)** Ninguna de las anteriores.

9

Herramientas

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer las diferentes herramientas que existen para diseño de páginas web, programación en XML.
- ✓ Poner a disposición del alumno las referencias y el acceso a cada una de las herramientas vistas en el libro.

9.1 SOFTWARE PARA DISEÑO WEB

9.1.1 ADOBE DREAMWEAVER

- **Nombre completo:** Adobe Dreamweaver.
- **Web:** <http://www.adobe.com/la/products/dreamweaver/>.
- **Licencia:** propietario.
- **Descripción:** Adobe Dreamweaver es una aplicación creada para la construcción y edición de sitios y aplicaciones Web. Como editor WYSIWYG que es, Dreamweaver permite ocultar el código HTML de cara al usuario, haciendo posible que alguien no entendido pueda crear páginas webs de manera intuitiva.

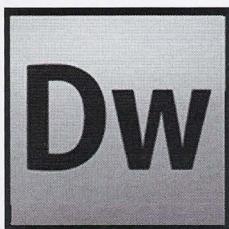
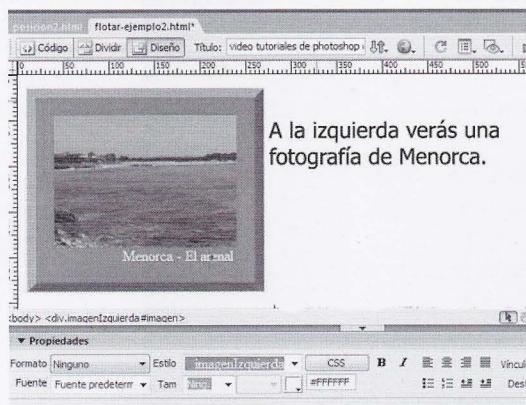


Figura 9.1. Icono de DreamWeaver

La interfaz de DreamWeaver es sencilla en todas sus versiones. Como se puede observar en la imagen, tres etiquetas indicarán el tipo de vista que se desea obtener. Si el usuario conoce HTML, puede activar la etiqueta *Código*, de manera que podrá programar las diferentes opciones de su aplicación web. Por otro lado, si el usuario desea hacerlo de una manera más gráfica, podrá pulsar la etiqueta *Diseño*, de manera que podrá incluir imágenes, botones y mover cada elemento de manera visual, sin tener que codificar nada.



9.1.2 KOMPOZER

- **Nombre completo:** KompoZer.
- **Web:** <http://kompozer.net/>.
- **Licencia:** MPL/GPL/LGPL.
- **Descripción:** Muy similar a DreamWeaver, este editor facilita el desarrollo de páginas web, gracias a las diferentes visualizaciones disponibles en su interfaz (código fuente, ventana WYSIWYG, visión con tags de HTML realizados), entre los cuales es posible cambiar mediante un sistema de pestañas. Está basado en Nvu.

9.1.3 EXPRESSION WEB



Figura 9.3. Icono de Expression Web

- **Nombre completo:** Microsoft Expression Web.
- **Web:** http://www.microsoft.com/expression/products/StudioWebPro_Overview.aspx.
- **Licencia:** propietario.
- **Descripción:** es un editor HTML desarrollado por Microsoft como una aplicación para sitios web, considerada una versión superior de FrontPage 2003 (se le considera su sucesor directo) por su semejanza en aspectos a este último. Utiliza el mecanismo WYSIWYG.

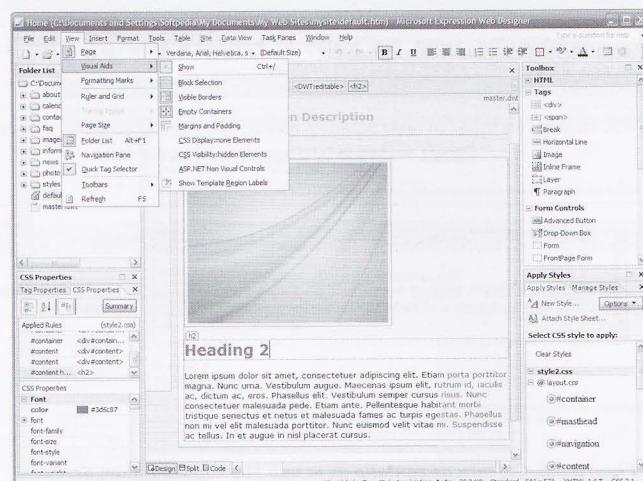


Figura 9.4. Captura de pantalla del software Microsoft Expression Web designer

Como se puede observar en la imagen, *Expression web* permite a aquellos usuarios que no conozcan el lenguaje HTML diseñar páginas web sencillas, a la vez que ver el código que se va generando simultáneamente. La pantalla se divide en diferentes áreas. En la del centro, el usuario irá creando la página HTML resultante.

9.2 HERRAMIENTAS PARA LA VALIDACIÓN DE DOCUMENTOS XML VÍA WEB

9.2.1 MARKUP VALIDATION SERVICE

- **Nombre completo:** Markup Validation Service.
- **Web:** <http://validator.w3.org/>.
- **Descripción:** consiste en un servicio libre vía web que ayuda al usuario a comprobar la validez de los documentos webs. Este software procesa documentos escritos en la mayoría de los lenguajes de marcas (soporta HTML 4.01, XHTML 1.1 y 1.0, MathML, SMIL y SVH (1.0 y 1.1).

Figura 9.5. Captura de pantalla de la web Markup Validation Service

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below that, there are links for "Jump To: Notes and Potential Issues" and "Congratulations · Icons". The main area displays the result of a validation: "This document was successfully checked as XML!". It shows the "Result" as "Passed, 1 warning(s)" and the "Source" code:

```
<telefono>745 15 56 11</telefono>
<fecha>22/9/2011</fecha>
<hora>15:05</hora>
<mensaje>Juego2: Arkanoid</mensaje>
</sms>
<sms>
<telefono>842 35 22 00</telefono>
<fecha>10/11/2011</fecha>
<hora>09:22</hora>
<mensaje>Juego3: Comecocos</mensaje>
</sms>
</BDSms>
```

Below the source code, there are fields for "Encoding: utf-8" and "Doctype: XML", both set to "(detect automatically)". The "Root Element: bdsms" is also listed. At the bottom left is a "I ❤ VALIDATOR" button, and at the bottom right are links for "The W3C validators rely on community support for hosting and development. [Donate](#) and help us build better tools for a better web." and a "Flattr" button.

Figura 9.6. Ejemplo de utilización de Markup Validation Service

9.2.2 XMLVALIDATION

- **Nombre completo:** XmlValidation.
- **Web:** <http://xmlvalidation.com>.
- **Descripción:** consiste en un servicio libre vía web que permite al usuario validar sus páginas con el objetivo de que cumplan todos los estándares correspondientes.

The screenshot shows the XML Validation service interface. The title bar says "<?xm 1?>". Below it are tabs for "Documentation", "An example", and "About". The main area is titled "Validate an XML file" and contains the following text: "Read here how to validate your XML files (including referenced DTDs) online with just a few mouse clicks." A text input field is provided for pasting XML code, with the placeholder "Please copy your XML document in here:". The XML code shown is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Bdsm>
<ELEMENT Bdsm (sms*)>
<ELEMENT sms (telefono, fecha, hora, mensaje)>
<ELEMENT telefono (#PCDATA)>
<ELEMENT fecha (#PCDATA)>
<ELEMENT hora (#PCDATA)>
<ELEMENT mensaje (#PCDATA)>
```

Below the input field, there's a section for "Or upload it:" with a "Seleciónar archivo" button and a note "No se ha s...n archivo". At the bottom, there's a note about validation against schemas and a "validate" button.

Figura 9.7. Ejemplo de utilización de XML Validation

9.2.3 VALIDOME

- **Nombre completo:** Validome.
- **Web:** <http://www.validome.org>.
- **Descripción:** permite comprobar la sintaxis de las páginas web de los usuarios con un servicio de validación de alta velocidad, acorde a los estándares oficiales. La validación de este software evitará al usuario futuros problemas con diferentes navegadores. Este software consiste en un servicio libre vía web que permite al usuario validar sus páginas con el objetivo de que cumplan todos los estándares correspondientes.

9.3 SOFTWARE PARA EDITAR Y VALIDAR DOCUMENTOS XML

9.3.1 SERNA FREE

- **Nombre completo:** Serna Free.
- **Web:** <http://www.syntext.com/products/serna-free/>.
- **Licencia:** Open Source.
- **Descripción:** es un editor de documentos XML que permite analizar la validez de las webs creadas, así como aplicar los diferentes estilos previamente definidos. Se le considera uno de los editores de XML más potentes que existen actualmente.

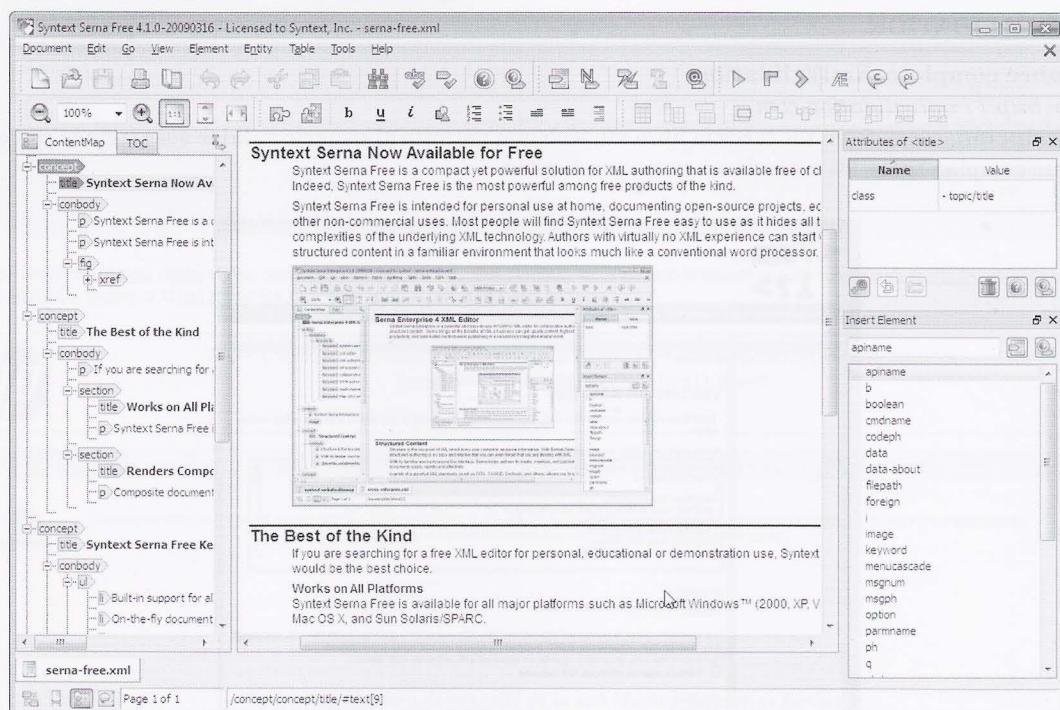


Figura 9.8. Captura de pantalla de la interfaz de Serna Free

También dispone de la versión avanzada, **Serna Enterprise XML editor**, que aunque es de pago, posee algunas características más avanzadas que la versión gratuita. El usuario, antes de comprarla podrá probar una versión de pruebas disponible en <http://cms.syntext.com/index/trial/input>.

9.3.2 <OXYGEN/>

- **Nombre completo:** oXygen XML Editor.
- **Web:** <http://www.oxygenxml.com/>.
- **Licencia:** Open Source.
- **Descripción:** es la única herramienta XML que soporta todos los lenguajes de esquema XML. La implementación de XSLT y XQuery se ha mejorado con depuradores de gran alcance y perfiles de rendimiento. Especialmente enfocado a los autores de contenido, <oXygen/> XML Author viene con un modo de edición configurable y extensible visual basado en hojas de estilo CSS del W3C, con listas para usar DITA, DocBook, TEI y el apoyo XHTML, por lo que <oXygen/> la creación de XML ideal solución. OXygen XML Editor apoya documentos de XML, de XSL, de TXT, de XSD y del DTD.

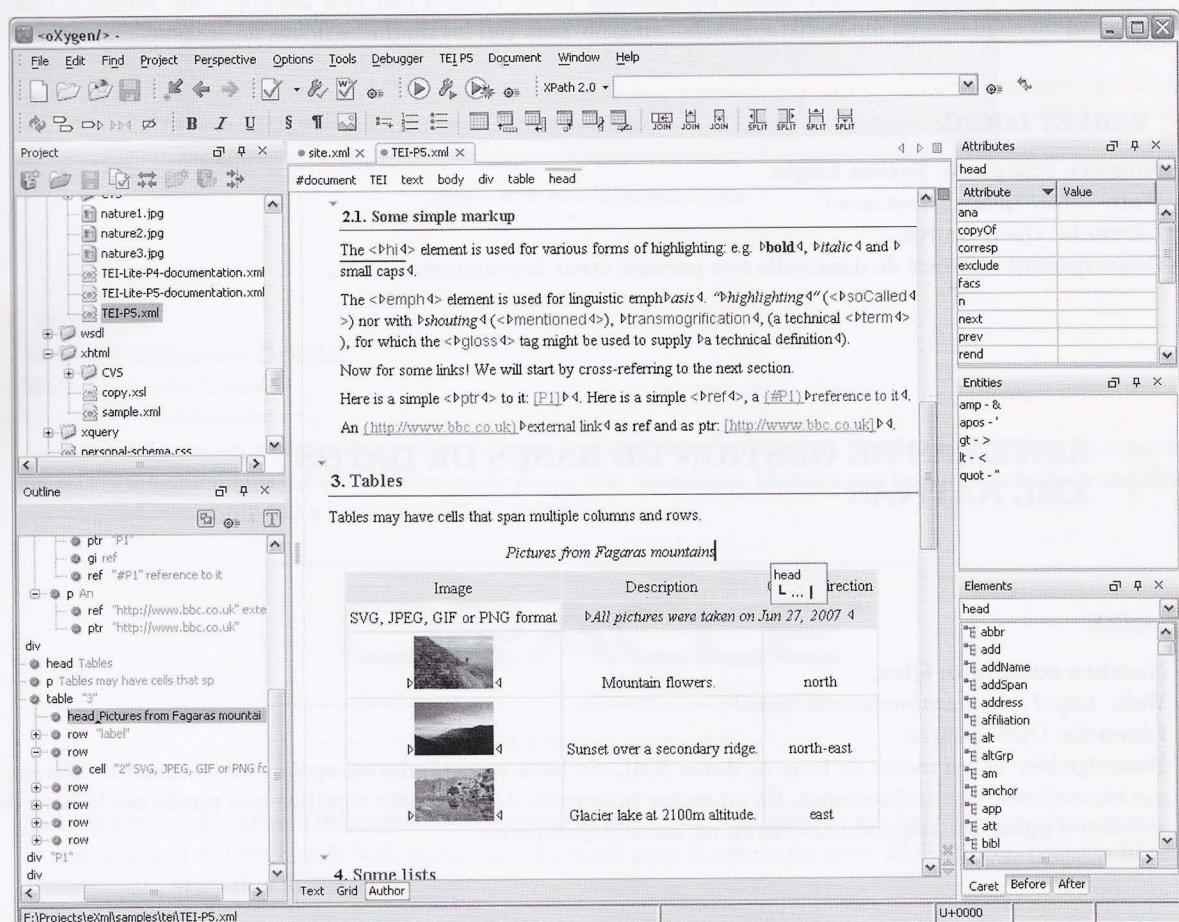


Figura 9.9. Captura de pantalla de la interfaz de oXygen

9.3.3 TOTALEDIT

- **Nombre completo:** TotalEdit.
- **Web:** <http://www.codertools.com/totaledit.aspx>.
- **Licencia:** Open Source.
- **Descripción:** es un editor para varios lenguajes (PHP, HTML, Javascript, JSP, XML...) que permite al usuario trabajar más rápido ya que posee características como identificación de palabras, completa automáticamente las palabras, visualización previa de páginas webs, etc.

9.3.4 XML NOTEPAD

- **Nombre completo:** Microsoft XmlNotepad.
- **Web:** <http://www.microsoft.com/downloads/en/details.aspx?familyid=72D6AA49-787D-4118-BA5F-4F30FE913628&displaylang=en>.
- **Licencia:** Open Source.
- **Descripción:** permite buscar y editar documentos XML. Cuenta con una interfaz muy limpia e intuitiva, presenta un esquema en modo vista de árbol sincronizado con los distintos nodos de edición.

9.3.5 VERVET LOGIC

- **Nombre completo:** Vervet Logic.
- **Web:** <http://www.vervet.com/>.
- **Licencia:** Open Source.
- **Descripción:** software de desarrollo que permite crear herramientas XML.

9.4 SISTEMAS DE GESTIÓN DE BASES DE DATOS XML NATIVAS

9.4.1 QIZX

- **Nombre completo:** Qizz.
- **Web:** <http://www.xmlmind.com/qizz/>.
- **Licencia:** Open Source.
- **Descripción:** es un motor de base de datos XML. Se basa en estándares: apoya completamente XQuery con sus extensiones y actualizaciones. Es un motor integrable de Java: esto significa que puede ser la base de un servidor o puede ser encajado directo en un uso independiente.

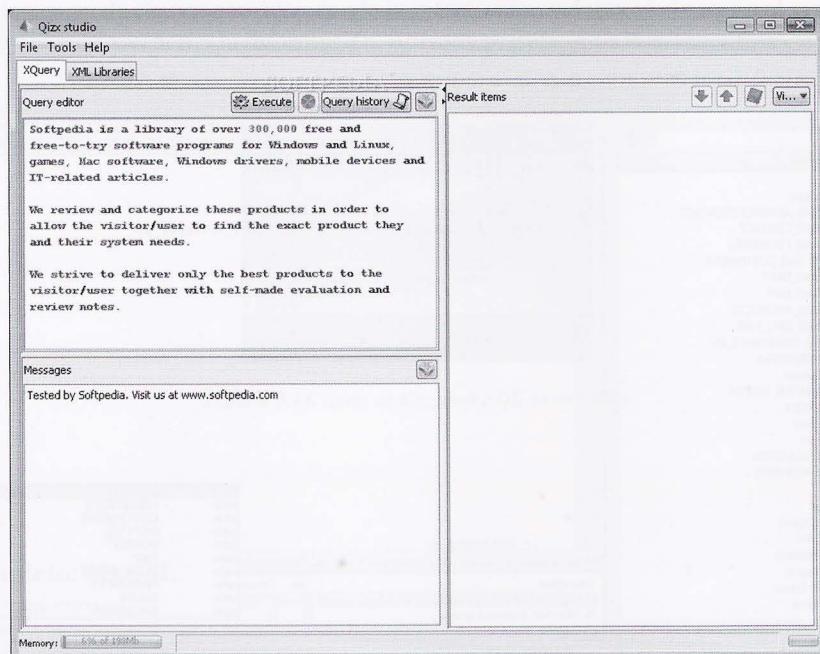


Figura 9.10. Interfaz de Quiz studio.

9.4.2 ORACLE

- **Nombre completo:** Oracle.
- **Web:** www.oracle.com.
- **Licencia:** propietario.
- **Descripción:** es un sistema de gestión de base de datos desarrollado por Oracle Corporation. Se considera a Oracle como uno de los sistemas de bases de datos más completos. Destaca por tener soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma.



Figura 9.11. Icono de ORACLE

Su dominio en el mercado de servidores empresariales ha sido casi total hasta hace poco, recientemente sufre la competencia del Microsoft SQL Server de Microsoft y de la oferta de otros RDBMS con licencia libre como PostgreSQL, MySQL o Firebird.

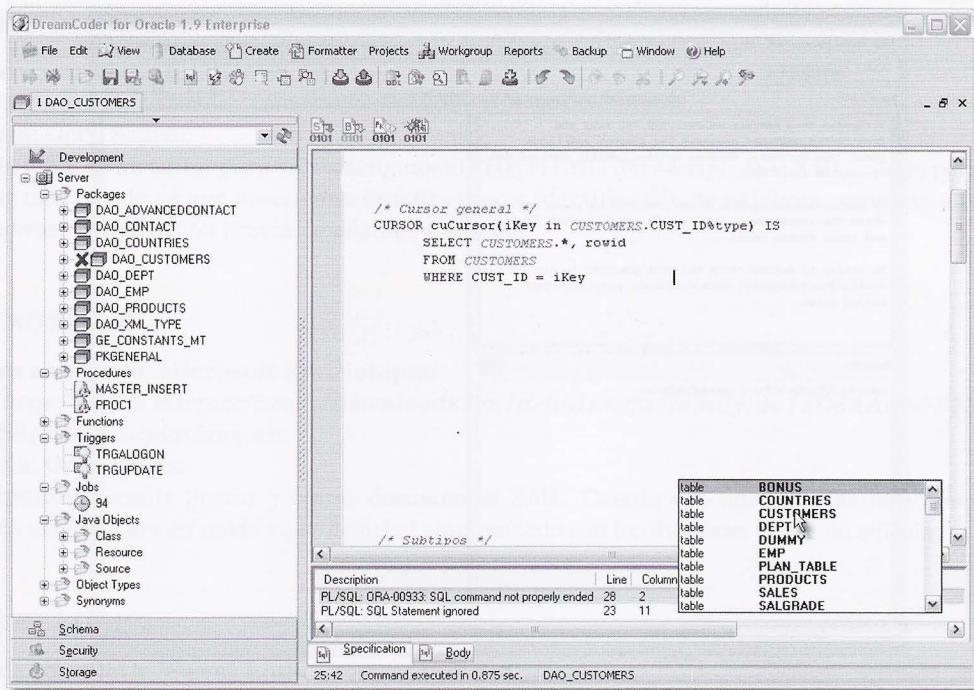


Figura 9.12. Interfaz de Oracle 1.9 Enterprise

9.4.3 MICROSOFT SQL SERVER

- **Nombre completo:** Microsoft SQL Server.
- **Web:** www.microsoft.com/sql/.
- **Licencia:** Microsoft EULA.
- **Descripción:** es un sistema para la gestión de bases de datos basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL. Sus características principales son el soporte de transacciones, escalabilidad, estabilidad y seguridad. Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.

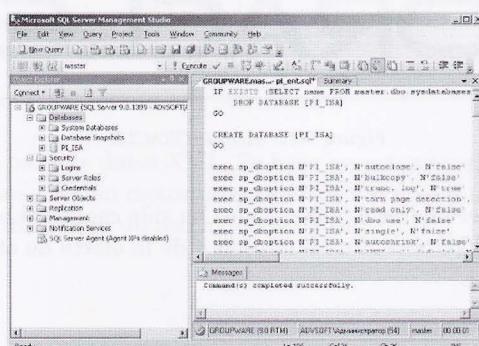


Figura 9.13. Interfaz de Microsoft SQL Server Management Studio

Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red solo acceden a la información. Además permite administrar información de otros servidores de datos.



Figura 9.14. Icono de Microsoft SQL Server 2008

9.4.4 MYSQL

- **Nombre completo:** MySQL.
- **Web:** www.mysql.com.
- **Licencia:** GPL o uso comercial.
- **Descripción:** es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos deben comprar a la empresa una licencia específica que les permita este uso.

Es muy utilizado en aplicaciones web, en plataformas (Linux /Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla.

9.4.5 eXIST

- **Nombre completo:** eXist.
- **Web:** <http://exist-db.org/>.
- **Licencia:** Open Source.
- **Descripción:** es un sistema para la gestión de bases de datos libre, que utiliza la tecnología XML, acorde a los modelos de datos XML y procesamiento basado en índices por XQuery. Una de las ventajas principales de este software es que tiene una comunidad de usuarios y desarrolladores muy activa, con múltiples foros y código disponible.



Figura 9.15. Icono de eXist

9.5 SOTWARE PARA GESTIÓN EMPRESARIAL

9.5.1 OPENBRAVO

- **Nombre completo:** Openbravo ERP Community Edition.
- **Web:** <http://www.openbravo.com/es/>.
- **Licencia:** Openbravo Public License, basada en la MPL.
- **Descripción:** es una aplicación de código abierto de gestión empresarial del tipo ERP. Es una aplicación con arquitectura cliente/servidor web escrita en Java. Se ejecuta sobre Apache y Tomcat y con soporte para bases de datos PostgreSQL y Oracle.

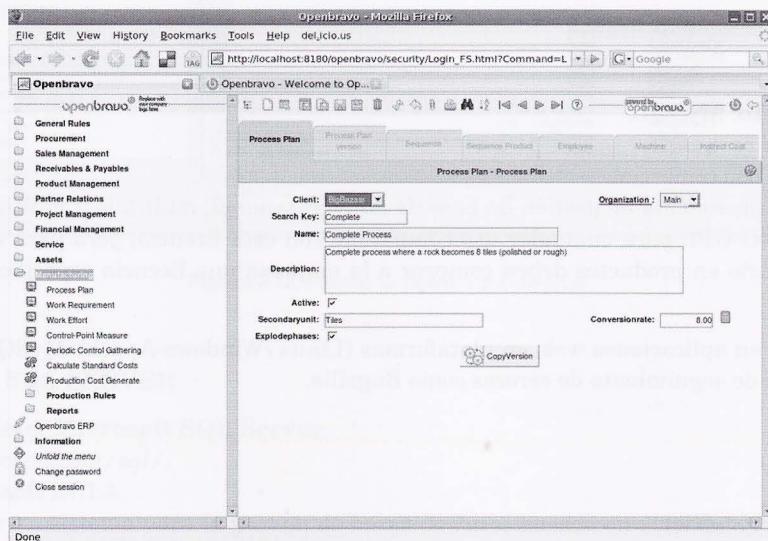


Figura 9.16. Interfaz de OpenBravo

9.5.2 OPENERP

- **Nombre completo:** Open ERP.
- **Web:** <http://www.openerp.com/>.
- **Licencia:** GPL.
- **Descripción:** es un sistema ERP y CRM. Tiene componentes separados en esquema cliente-servidor. Dispone de interfaces XML-RPC y SOAP. Entre sus características están la contabilidad analítica, contabilidad financiera, gestión de almacenes/inventario, gestión de ventas y compras, automatización de tareas, campañas de marketing, etc. Dentro de la construcción misma del software se hace uso intensivo de flujos de trabajo.

9.5.3 WEBERP

- **Nombre completo:** webERP.
- **Web:** <http://www.web-erp.org/>.
- **Licencia:** GPL.
- **Descripción:** es un sistema de código abierto para la gestión de pequeñas y medianas empresas.

9.5.4 JD EDWARDS ENTERPRISEONE

- **Nombre completo:** JD Edwards EnterpriseOne.
- **Web:** <http://www.oracle.com/lad/products/applications/jd-edwards-enterpriseone/index.html>.
- **Licencia:** propietario.
- **Descripción:** es una suite de software de planificación de recursos empresariales completo con aplicaciones integradas que combina valor de negocio, tecnología basada en estándares y profunda experiencia del sector en una solución empresarial. Se le considera uno de los mejores sistemas de gestión empresarial del mercado.

9.5.5 AP BUSINESS ALL IN ONE

- **Nombre completo:** AP Business All in One.
- **Web:** <http://www.sap.com/spain/sme/solutions/businessmanagement/businessallinone/index.epx>.
- **Licencia:** propietario.
- **Descripción:** dispone de recursos empresariales (ERP), gestión de las relaciones con los clientes (CRM) e inteligencia empresarial.



Figura 9.17. Icono de SAP

ACTIVIDADES 9.1



- Escribe un software de cada uno de los apartados del capítulo, instálalo y conoce su interfaz.
- Busca por Internet, al menos, dos herramientas más de cada uno de los apartados del capítulo.



RESUMEN DEL CAPÍTULO

En este capítulo se han presentado diferentes programas que se utilizan de manera profesional en todo el mundo.

Se ha proporcionado información de cada uno de ellos con el objetivo de que el alumno pueda ampliarlos cuando desee evaluar los diferentes programas antes de decidirse utilizar uno u otro en su vida profesional.

Por supuesto, no han sido añadidos todos los programas existentes, pero la selección de los explicados se basa en los más utilizados, más completos o más intuitivos.

Material adicional

El material adicional de este libro puede descargarlo en nuestro portal Web: <http://www.ra-ma.es>.

Debe dirigirse a la ficha correspondiente a esta obra, dentro de la ficha encontrará el enlace para poder realizar la descarga. Dicha descarga consiste en un fichero ZIP con una contraseña de este tipo: XXX-XX-XXXX-XXX-X la cual se corresponde con el ISBN de este libro.

Podrá localizar el número de ISBN en la página 2 (página de créditos). Para su correcta descompresión deberá introducir los dígitos y los guiones.

Cuando descomprima el fichero obtendrá los archivos que complementan al libro para que pueda continuar con su aprendizaje.

INFORMACIÓN ADICIONAL Y GARANTÍA

- RA-MA EDITORIAL garantiza que estos contenidos han sido sometidos a un riguroso control de calidad.
- Los archivos están libres de virus, para comprobarlo se han utilizado las últimas versiones de los antivirus líderes en el mercado.
- RA-MA EDITORIAL no se hace responsable de cualquier pérdida, daño o costes provocados por el uso incorrecto del contenido descargable.
- Este material es gratuito y se distribuye como contenido complementario al libro que ha adquirido, por lo que queda terminantemente prohibida su venta o distribución.

Índice Alfabético

A

ACID, 159
Adobe Dreamweaver, 222
Agregador, 204
Agregadores de contenidos, 195
AJAX, 102
ANY, 117
AP Business All in One, 233
API, 22, 187
Atributo, 13, 116
Atributo action, 62
Atributo align, 59, 60, 77, 78
Atributo alt, 59
Atributo border, 60, 77
Atributo cellpadding, 77
Atributo cellspacing, 77
Atributo class, 37, 95
Atributo clear, 60
Atributo cols, 81, 82
Atributo colspan, 76
Atributo content, 54
Atributo disabled, 71
Atributo enctype, 63
Atributo for, 69
Atributo height, 59
Atributo href, 29, 47, 48, 51, 55, 79, 97
Atributo hspace, 60
Atributo http-equiv, 54
Atributo id, 37, 47, 50, 69, 95
Atributo lang, 54
Atributo longdesc, 59
Atributo media, 98
Atributo name, 47, 50, 51, 54, 63, 81
Atributo noresize, 84
Atributo post, 62
Atributo profile, 53
Atributo readonly, 71

Atributo rel, 55, 58, 97, 98
Atributo rev, 58
Atributo rows, 81, 82
Atributo rowspan, 76
Atributos, 29, 124
Atributo scr, 59, 81
Atributo scrolling, 84
Atributo selected, 63
Atributo size, 63
atributo style, 92
Atributo style, 92, 94
Atributo target, 85, 86
Atributo title, 34
Atributo type, 44, 97
Atributo valign, 78
Atributo value, 64
Atributo vspace, 60
Atributo width, 59

B

B2B, 162, 187, 189
B2C, 187, 189
B2E, 187, 189
BD, 156
bgcolor, 75, 76, 85
Blog, 202
Blogroll, 202
Botón, 64
Botón de radio, 65
Botón push, 64
Botón reset, 64
Botón submit, 64

C

Capa, 87
Casillas de selección, 65
CDATA, 107, 117

CMS, 194
 Comentario, 30
 Contenido sindicado, 194
 Control de objeto, 67
 Control oculto, 67
 CRM, 208, 218, 219
 CSS, 90, 92.
 Cuadro de texto, 66

D

Declaración, 31, 92
 Declaración <!DOCTYPE>, 32
 Deprecated, 17, 32
 Directorio de canales, 204
 Documento fuente, 60
 Documento bien formado, 130
 Documento válido, 130
 DOM, 22, 187
 Drupal, 194
 DSDL, 18
 DTD, 16, 114
 DTD estricta, 32
 DTD para documentos con marcos, 33
 DTD transicional, 32

E

ECMA-262, 102
 Elemento, 13, 116
 Elemento A, 29, 47, 48, 50, 55, 60, 85
 Elemento BASE, 48, 58, 86
 Elemento bgcolor, 78
 Elemento BODY, 31, 36, 80
 Elemento BR, 27, 60
 Elemento BUTTON, 64, 70
 Elemento CAPTION, 77
 Elemento DIV, 38, 95
 Elemento FIELDSET, 71
 Elemento Form, 62
 Elemento FRAME, 80, 83, 84
 Elemento FRAMESET, 31, 36, 80, 82
 Elemento H1, 92, 94
 Elemento HEAD, 31, 34, 53, 55
 Elemento HTML, 31
 Elemento ID, 87

Elemento IMG, 35, 59, 60
 Elemento INPUT, 63, 64
 Elemento LABEL, 69
 Elemento LEGEND, 71
 Elemento LI, 27, 43
 Elemento LINK, 47, 54, 55, 58
 Elemento META, 54, 64, 92, 98
 Elemento NOFRAMES, 80, 83
 Elemento OBJECT, 37, 61, 67
 Elemento OL, 45
 Elemento OPTGROUP, 66, 71
 Elemento OPTION, 63, 71
 Elemento P, 27, 38, 41
 Elemento raíz, 121
 Elementos, 25
 Elementos complejos, 126
 Elemento SELECT, 63, 71
 Elemento SPAN, 37, 38, 88, 95
 Elemento simple, 122
 Elemento STYLE, 93, 94
 Elementos y etiquetas, 29
 Elemento TABLE, 76
 Elemento TD, 76
 Elemento TEXTAREA, 66
 Elemento TH, 76
 Elemento TITLE, 34
 Elemento TR, 76
 Elemento UL, 27, 43
 EMPTY, 117
 Entidad, 116
 Entidad-Relación, 162
 ERP, 208
 Espacio de nombres, 109
 Esquema, 120
 Etiqueta, 10, 13, 24
 eXist, 231
 eXtensible Hyper Text Markup Language, 88

F

Feed, 194, 204
 FLWOR, 176
 Formulario HTML, 62
 Frame, 79

G

GML, 12, 15

H

Herramientas de validación, 131
 Hipertexto, 37, 47
 Hoja de estilo, 40, 46, 58, 74, 80, 90
 Hoja de estilo alternativa, 97
 Hoja de estilo externa, 92, 95
 HTML, 11, 22, 24.
 HTML 4.0 Frameset DTD, 17
 HTML 4.0 Strict DTD, 17
 HTML 4.0 Transitional DTD, 17
 HTTP, 54.
 Hyper Text Markup Language, 24

I

Indicador, 127
 Indicador de grupo, 127
 Indicador de ocurrencia, 127
 Indicador de orden, 127
 Índice, 161, 162
 Internet Explorer, 42
 Intranet, 47
 ISO, 14.

J

Java, 58
 JavaScript, 102
 JD Edwards EnterpriseOne, 233
 Joomla, 194
 JSON, 102

K

Knowledge Base, 161
 KompoZer, 223

L

Lector de feed, 195
 Lenguaje de marcado, 10
 Lenguaje de marcas, 10
 Lenguaje descriptivo, 11
 Lenguaje orientado a presentación, 10
 Lenguaje procedurale, 11

LJS, 102

Loose, 17

M

Marcador, 47, 50
 Marcos, 79
 Markup Validation Service, 224
 Menú desplegable, 66
 Microsoft, 42, 81
 Microsoft Expression Web, 223
 Microsoft SQL Server, 230
 Microsoft XMLNotepad, 228
 Modelo de objetos del documento, 22
 MySQL, 231

N

Navegador, 27, 30

O

Open ERP, 232
 Oracle, 229
 Otros controles, 67
 oXygen XML Editor, 227

P

PCData, 117
 PDM, 210
 Página de marco, 79
 Página web, 16
 Página web dinámica, 16
 Página web estática, 16
 Programa navegador, 13
 PSVI, 17

Q

Qizz, 228

R

RDF, 195
 Referencia a carácter, 30
 Regla de estilo, 41, 46, 92
 Relación, 164
 Relax NG, 18
 RELAX NG, 128

Restricción, 124

RFC, 24.

RIA, 102

RSS, 195.

RSS 0.91, 195, 196

RSS 0.92, 195, 198

RSS 1.0, 195, 198

RSS 2.0, 195, 200, 202

S

Schematron, 129

Secuencia de elementos, 127

Selección de archivos, 66

Selector, 37

SEQUEL, 168

Serna Enterprise XML editor, 227

Serna Free, 226

SGBD, 154

SGML, 11, 12, 15, 32

SQL, 161, 162, 166

T

Tag, 13

TotalEdit, 228

Transacción, 158

Unix, 49

URI, 33, 108.

URL, 33.

URN, 33

U

V

Validación de documentos XML, 129

Validome, 226

Vervet Logic, 228

W

W3C, 12, 167.

webERP, 233

WYSIWYG, 11

X

XHTML, 22, 88

XML, 102.

XML Schema, 17, 120

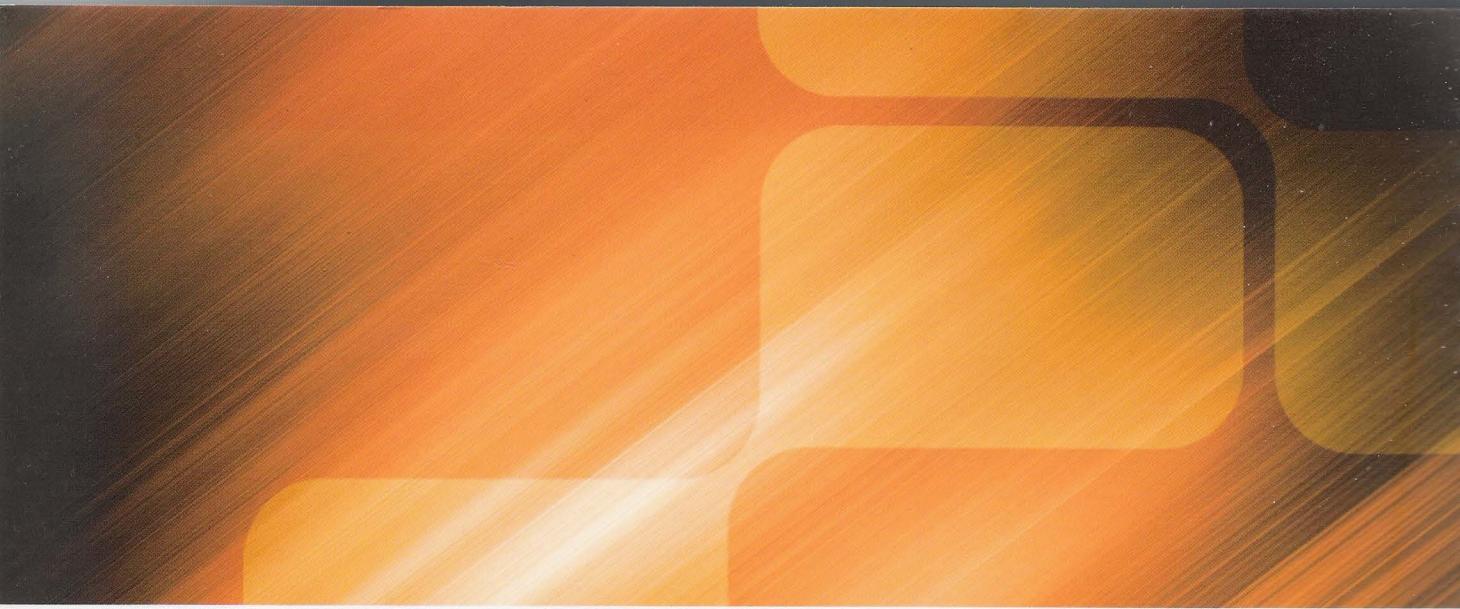
XML Schema Definition, 120

XmlValidation, 225

XPath, 167

XQuery, 167

XSD, 17, 120



La presente obra está dirigida a los estudiantes de los Ciclos Formativos de Grado Superior de **Administración de sistemas informáticos en red y Desarrollo de aplicaciones multiplataforma**, en concreto para el Módulo Profesional **Lenguajes de Marcas y Sistemas de Gestión de la Información**.

Los contenidos incluidos en este libro abarcan: el reconocimiento de las características de los lenguajes de marcas, los lenguajes para la visualización de la información (incluyendo HTML, XHTML, CSS), los lenguajes para el almacenamiento y transmisión de información (XML), la definición de esquemas y vocabularios en XML, la conversión y adaptación de documentos XML, el almacenamiento de información en XML, la aplicación de los lenguajes de marcas a la sindicación de contenidos (incluyendo RSS) y los sistemas de gestión empresarial (ERP). Así mismo, se incluye un capítulo donde se indican distintas herramientas que se pueden utilizar para los contenidos indicados anteriormente.

Los capítulos incluyen actividades, ejemplos y casos prácticos con el objetivo de facilitar la asimilación de los conocimientos tratados.

Así mismo, se incorporan test de conocimientos y ejercicios propuestos con la finalidad de comprobar que los objetivos de cada capítulo se han asimilado correctamente.

 En la página web de **Ra-Ma** (www.ra-ma.es) se encuentra disponible el material de apoyo y complementario.

