Garceta
grupo editorial

Técnico Superior en Informática ASIR - DAM - DAW

Lenguajes de marcas y sistemas de gestión de información

Juan Manuel Castro Ramos
José Ramón Rodríguez Sánchez

Lenguajes de marcas y sistemas de gestión de información

Lenguajes de marcas y sistemas de gestión de información

Juan Manuel Castro Ramos
José Ramón Rodríguez Sánchez

ISBN: 978-84-1545-217-1
IBERGARCETA PUBLICACIONES, S.L., Madrid 2012

Edición: 1.ª
Impresión: 1.ª
N.º de páginas: 416
Formato: 20 x 26 cm

Reservados los derechos para todos los países de lengua española. De conformidad con lo dispuesto en el artículo 270 y siguientes del código penal vigente, podrán ser castigados con penas de multa y privación de libertad quienes reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica fijada en cualquier tipo de soporte sin la preceptiva autorización. Ninguna parte de esta publicación, incluido el diseño de la cubierta, puede ser reproducida, almacenada o trasmitida de ninguna forma, ni por ningún medio, sea éste electrónico, químico, mecánico, electro-óptico, grabación, fotocopia o cualquier otro, sin la previa autorización escrita por parte de la editorial.

Diríjase a CEDRO (Centro Español de Derechos Reprográficos), www.cedro.org, si necesita fotocopiar o escanear algún fragmento de esta obra.

COPYRIGHT © 2012 IBERGARCETA PUBLICACIONES, S.L.
info@garceta.es

Lenguajes de marcas y sistemas de gestión de información

© Juan Manuel Castro Ramos, José Ramón Rodríguez Sánchez

1.ª edición, 1.ª impresión
OI: 270-2013
ISBN: 978-84-1545-217-1
Deposito Legal: M-26243-2012
Imagen de cubierta: © Cybrain_fotolia.com

Impresión: Print House, marca registrada de Coplar, S.A.

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Nota sobre enlaces a páginas web ajenas: Este libro puede incluir referencias a sitios web gestionados por terceros y ajenos a IBERGARCETA PUBLICACIONES, S.L., que se incluyen sólo con finalidad informativa. IBERGARCETA PUBLICACIONES, S.L., no asume ningún tipo de responsabilidad por los daños y perjuicios derivados del uso de los datos personales que pueda hacer un tercero encargado del mantenimiento de las páginas web ajenas a IBERGARCETA PUBLICACIONES, S.L., y del funcionamiento, accesibilidad y mantenimiento de los sitios web no gestionados por IBERGARCETA PUBLICACIONES, S.L., directamente. Las referencias se proporcionan en el estado en que se encuentran en el momento de publicación sin garantías, expresas o implícitas, sobre la información que se proporcione en ellas.

Juan Manuel Castro Ramos
José Ramón Rodríguez Sánchez

Técnico Superior en Informática **ASIR - DAM - DAW**

Lenguajes de marcas y sistemas de gestión de información

Garceta
grupo editorial

2.5.3. Enlaces 52
2.5.4. Agrupación del contenido 56
2.5.5. Imágenes 59
2.5.6. Tablas 62
2.5.7. Marcos 66
2.5.8. Objetos multimedia 68
2.5.9. Formularios 73
2.6. XHTML 82
2.6.1. Estructura y elementos del documento 83
2.6.2. Normas en XHTML 84
2.7. HTML 5 86
2.7.1. Elementos de sección 87
2.7.2. Formularios 90
2.7.3. Objetos multimedia 93
2.8. Validación 96
2.8.1. Validación en línea 98
2.8.2. Validación local 101
Ejercicios propuestos 105

Capítulo 3. CSS. Hojas de estilo ***111***

3.1. Introducción 112
3.2. Sintaxis 114
3.2.1. Cómo incluir CSS en el documento 114
3.2.2. Cómo construir las reglas CSS 116
3.3. Selectores 117
3.3.1. Selector universal 117
3.3.2. Selector de tipo 117
3.3.3. Selector descendiente 117
3.3.4. Selector hijo 118
3.3.5. Selector adyacente 119
3.3.6. Selector de atributos 119
3.3.7. Selector de clase 119
3.3.8. Selector ID 120
3.3.9. Pseudo-clases 121
3.3.10. Pseudo-elementos 122
3.4. Modelo de cajas 124
3.4.1. Fundamentos 124
3.4.2. Ancho, alto 126
3.4.3. Margin, padding 126
3.4.4. Bordes 127
3.4.5. Colores y fondos 128

3.4.6. Posicionamiento 131
3.4.7. Visualización 139
3.5. Texto 145
3.6. Listas 146
3.7. Tablas 149
3.8. Formularios 150
3.9. Layout 152
3.10. Prioridad 157
3.11. Miscelánea 159
3.11.1. Estructuración del código CSS 159
3.11.2. Aplicaciones Web 159
3.11.3. Sitios Web orientados al diseño 159
Ejercicios propuestos 160

Capítulo 4. XML. Almacenamiento de datos ***165***

4.1. Introducción 166
4.2. Documentos XML 166
4.3. Estructura jerárquica de un documento XML 167
4.4. Modelo de datos de un documento XML. Nodos 170
4.5. Corrección sintáctica: documento XML bien formado 181
4.6. Documentos XML válidos 184
4.7. Validación de documentos XML con DTD 184
4.7.1. Estructura de un DTD. Elementos 185
4.7.2. Poniendo todo junto 199
4.8. Validación de documentos XML con esquemas XML 200
4.8.1. Estructura de un esquema XML. Componentes 202
4.8.2. Componentes básicos de un esquema 204
4.8.3. Tipos de datos 208
4.8.4. Tipos de datos simples vs. complejos 214
4.8.5. Definición de tipos de datos complejos 224
4.8.6. Diferentes declaraciones de elementos 226
4.8.7. Modelos de diseño de esquemas XML 235
4.8.8. Poniendo todo junto 238
4.8.9. Construcción avanzada de esquemas 239
4.9 Otros mecanismos para validar XML 250
4.10 Otros lenguajes basados en XML 250
4.10.1. SVG 251
4.10.2. WML 252
4.10.3. RSS 253
4.10.4. Atom 253
4.10.5. DocBook 253

4.10.6. XBRL 254
4.11. Otras formas de almacenar información 254
4.11.1 JSON 254
4.11.2. YAML 256
Ejercicios propuestos 258

Capítulo 5. XML. Tratamiento y recuperación de datos *263*

5.1. Introducción 264
5.2. Bases de datos XML nativas 264
5.2.1. BaseX 264
5.3. XPath 267
5.4. XQuery 277
5.5. Otras tecnologías complementarias: XLink y XPointer 285
5.5.1. XLink 285
5.5.2. XPointer 287
5.6. Bases de datos relacionales con XML 288
5.7. Manejo de XML desde Java 292
5.7.1. SAX 292
5.7.2. DOM 293
5.7.3 JAXP 295
5.7.4 JAXB 298
Ejercicios propuestos 300

Capítulo 6. Transformación de documentos. XSLT *305*

6.1. Introducción 306
6.2. XSLT 306
6.2.1. Enlace de documentos XML con hojas de estilo 312
6.2.2. Elementos básicos de una hoja de transformaciones 314
6.2.3. Instrucciones de control 325
6.2.4. Generación de nuevos elementos y atributos 331
6.2.5. Elementos avanzados de una hoja de transformaciones 338
6.2.6. Instrucciones de XSLT 2.0 348
6.3. XSL-FO 348
6.3.1. Objetos de formateo 350
6.3.2. Objetos de formateo para la estructura de documentos 353
6.3.3. Objetos de formateo para el contenido de páginas 356
6.3.4. Objetos de formateo para generar listas 357
6.3.5. Objetos de formateo para generar tablas 358
6.3.6. Objetos de formateo para generar enlaces, imágenes 361
Ejercicios propuestos 364

Capítulo 7. Sindicación de contenidos. RSS *369*

7.1. Introducción a RSS 370
7.2. Estructura de un documento RSS 371
7.3. Elementos principales de un RSS 372
7.3.1. <channel> 372
7.3.2. <item> 374
7.4. Generación de RSS 375
7.5. Validación del archivo RSS 377
7.6. Publicación del archivo RSS 378
Ejercicios propuestos 381

Capítulo 8. Sistemas de gestión de información. ERP *383*

8.1. Introducción 384
8.2. Inteligencia del negocio 384
8.3. ERP 385
8.3.1. SAP 386
8.3.2. OpenBravo 387
8.4. CRM 392
8.4.1. SugarCRM 393
Ejercicios propuestos 398

CAPÍTULO 1

Introducción a los Lenguajes de Marcas

CONTENIDO

Concepto y ventajas
SGML. El origen
Características de los lenguajes de marcas
Clasificación de los lenguajes de marcas
Organizaciones y estándares
Introducción a los principales lenguajes de marcas

1.1. Concepto y ventajas

Las personas utilizamos en nuestro lenguaje hablado ciertos gestos, diferente entonación, pausas, etc. que acompañan al contenido que queremos transmitir. De modo parecido en el lenguaje escrito utilizamos tamaños de letra, negrita, numeración, viñetas, tablas, colores, etc. que también acompañan a la información para que sea más fácil de entender. En definitiva estamos aplicando un determinado formato a la información que transmitimos.

En el caso de los documentos que intercambiamos a través de Internet, como las páginas web, son los lenguajes de marcas quienes nos permiten aplicar dicho formato.

Un documento que contenga exclusivamente texto es perfectamente legible por nosotros, aunque evidentemente, tedioso e inapropiado para publicar páginas web. Si le aplicamos formato mediante un lenguaje de marcas como por ejemplo HTML, obtenemos un archivo también legible pero más difícil de interpretar. De esta tarea se encargará el navegador o mejor dicho el agente de usuario, quien *interpreta* las marcas de formato y las aplica convenientemente al texto para dar lugar a una página web, que será mucho más agradable de leer que el texto original.

Una "marca" es una señal colocada dentro de un texto, con el fin de delimitar una parte del mismo y en muchos casos, aplicarle un determinado formato (aunque existen marcas con otros propósitos).

Las marcas más comunes están formadas por una palabra que describe su función encerrada entre los símbolos menor que (<) y mayor que (>) como **<html>**.

Es muy habitual que aparezcan por parejas, una de comienzo y otra de fin.

Por ejemplo:

<h1> Hola, este texto aparece más grande **</h1>**
<h3> Este texto aparece más pequeño, adiós **</h3>**

Actividad 1.1:

Guarda el texto anterior en el archivo saludo.txt, ábrelo con el navegador y comprueba que el texto permanece igual. Renombra el archivo como saludo.html y comprueba cómo interpreta las marcas el navegador.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación aunque se llamen también lenguajes. De hecho, no debemos utilizar la palabra "programar" cuando nos referimos a lenguajes de marcas, puesto que no disponen de los elementos típicos como variables, arrays, sentencias de control, funciones, etc.

Sin embargo, los lenguajes de marcas se pueden combinar dentro del mismo documento, con otros lenguajes como JavaScript o PHP, que sí son lenguajes de programación, con el objetivo de aportar funcionalidad y dinamismo a la página web.

Otro aspecto importante a tener en cuenta cuando hablamos de lenguajes de marcas es el destinatario de la información. Quizás lo más habitual, es un usuario final utilizando un navegador web en el PC de su casa, pero tenemos que considerar el resto de opciones que van

en aumento, otros destinatarios podrían ser: usuarios en dispositivos móviles, usuarios con deficiencias visuales o motrices, usuarios de avanzada edad[1], un periférico como la impresora, los robots de los buscadores, etc. Por esta razón, es más correcto utilizar el término general, agente de usuario (user-agent) en lugar de navegador.

La presentación de la misma página web para cada uno de estos usuarios debe ser lógicamente muy distinta, así por ejemplo, un texto en negrita puede representarse respectivamente, por caracteres con mayor grosor, por un volumen más alto en el sintetizador de voz, por más puntos en el papel, etc.

La cuestión es que el lenguaje de marcas debe ser independiente del destinatario final, es el intérprete del lenguaje quien se encarga de representar las marcas de la forma adecuada. HTML por ejemplo, no especifica en sus etiquetas cómo serán representadas más tarde por el navegador. Esta es una de las razones por la que podemos encontrar ciertas diferencias en la visualización de una *misma* página, por parte de diferentes navegadores.

Por ejemplo, la siguiente tabla se visualiza diferente porque Firefox considera la barra / como salto de línea cuando no dispone de espacio para el texto y Explorer sin embargo, expande la columna. También podemos observar que Explorer asigna un grosor de letra algo mayor que Firefox.

Navegador	Texto	Salto de línea
M.Firefox	/MozillaFirefox /realizaunsaltodelíneaprevio	Espacio, Guión y Barra
I.Explorer	/InternetExplorer /norealizasaltodelíneaconlabarra	Espacio y Guión

Figura 1.1: Vista con Firefox

Navegador	Texto	Salto de línea
M.Firefox	/MozillaFirefox/realizaunsaltodelíneaprevio	Espacio, Guión y Barra
I.Explorer	/InternetExplorer/norealizasaltodelíneaconlabarra	Espacio y Guión

Figura 1.2: Vista con Explorer

Por otro lado, para independizar aún más la representación de la página web de su contenido, se creó CSS, que no es un lenguaje de marcas sino de estilos. Mediante CSS podemos especificar con mayor precisión y eficacia la representación de la información, para cada intérprete y para diferentes soportes, como monitores, dispositivos móviles, papel, voz, etc.

Dado el auge de los dispositivos móviles, muchas páginas presentan diferentes versiones adaptadas al dispositivo que utilice el usuario, en este caso, se trata de documentos html diferentes o bien del mismo documento html, pero aplicándole una hoja de estilos distinta.

Por ejemplo, el sitio del Museo del Prado, cuando detecta como agente de usuario un dispositivo móvil, nos presenta una página diferente cuya dirección es http://m.museodelprado.es

Figura 1.3: Vista con dispositivo móvil

Actividad 1.2:

Visita la web http://www.w3c.es, utiliza las vistas de la parte superior de la página e indica las diferencias que observes entre escritorio, móvil e impresora.

Descarga el complemento "User Agent Switcher" o similar para Firefox, modifica el agente de usuario y comprueba cómo cambia el aspecto de un sitio que esté configurado para dispositivos móviles.

1.2. SGML. El origen

En los años 60 las empresas de publicación y manejo de documentos electrónicos tenían el problema de falta de compatibilidad entre aplicaciones. El problema existente era que cada aplicación utilizaba sus propias marcas para describir los diferentes elementos, esto impedía el intercambio de documentos entre plataformas. Otra carencia importante era la separación entre estructura y aspecto del documento.

IBM, empresa pionera en investigación en informática y electrónica (más de 5.000 patentes en 100 años de historia) intentó resolver estos problemas a través de un lenguaje de marcas denominado GML (Generalized Markup Language).

GML independiza el documento del dispositivo que lo va a utilizar, usando marcas genéricas. Por otro lado GML incorpora marcas descriptivas para la estructura del documento que permiten distinguir el texto, de las listas, las tablas, etc. El mismo documento puede, entonces, ser utilizado por varios dispositivos, simplemente especificando un perfil para cada uno.

En 1986 GML pasó a manos de ISO y se convirtió en SGML (ISO 8879), Standard Generalized Markup Language, software libre y de código abierto.

Es importante tener en cuenta que SGML no es estrictamente un lenguaje sino un metalenguaje, es decir, un conjunto de normas que permiten crear otros lenguajes de marcas. Esto se hace definiendo un vocabulario o conjunto de elementos a utilizar, y una gramática o conjunto de reglas que rigen el uso de los elementos y sus atributos.

SGML, por tanto, es un metalenguaje que permite definir lenguajes de marcado. HTML por ejemplo, es uno de los lenguajes creados a partir de SGML.

Ventajas de SGML: Reutilización de los datos, integridad y control sobre los datos, portabilidad, adaptabilidad.

Inconvenientes de SGML: Alta complejidad

Un documento SGML consta de 2 partes:

- El prólogo: contiene la estructura.
  - La declaración: indica que el documento es SGML y algunos parámetros.
  - La definición de tipo de documento (DTD): indica la sintaxis particular del lenguaje creado.

- La instancia de documento: contiene los datos.

En los siguientes ejemplos se ha omitido la declaración SGML, la DTD sería el vocabulario y las reglas de uso, y la instancia de documento serían los datos.

Ejemplo 1: Los módulos del Ciclo ASIR

Vocabulario: asir, módulo, título, contenido, unidad.

Reglas: asir contiene varios módulos, un módulo tiene un elemento simple título y un elemento contenido, contenido tiene varias unidades y toda unidad debe estar en un contenido, las unidades son texto simple, detrás de cada unidad solo puede ir otra unidad o fin de contenido, detrás de un módulo solo puede ir otro módulo o fin de asir.

```
<asir>
  <módulo><título>Lenguajes de Marcas</título></módulo>
    <contenido>
      <unidad>Introducción</unidad>
      <unidad>HTML</unidad>
      <unidad>CSS</unidad>
      <unidad>XML</unidad>
      <unidad>XLST</unidad>
      <unidad>Sindicación</unidad>
      <unidad>ERP</unidad>
    </contenido>
  </módulo>
  ...
</asir>
```

Ejemplo 2: Mini HTML

Vocabulario: html, head, title, body, p.

Reglas: html contiene un elemento head y un elemento body, head contiene un elemento simple title, body contiene varios elementos p y todo elemento p debe estar en body, p es texto simple, detrás de p solo puede ir otro p o fin de body, detrás de body solo puede ir fin de html.

```
<html>
   <head><title>Mi página</title></head>
   <body>
      <p>Hola mundo</p>
      <p>Esta es mi página</p>
  </body>
</html>
```

Actividad 1.3:

Crea tu propio documento SGML indicando el vocabulario y las reglas usadas.

1.3. Características de los lenguajes de marcas

TEXTO PLANO

Los archivos de texto plano son aquellos que están compuestos únicamente por caracteres de texto, a diferencia de los archivos binarios que pueden contener imágenes, sonido, archivos comprimidos, programas compilados, etc.

Estos caracteres se pueden codificar con distintos códigos dependiendo del idioma o alfabeto que se necesite, por ejemplo: ASCII, ISO-8859-15, UTF-8.

Una de las principales ventajas de los archivos de texto plano es que pueden ser interpretados directamente por un simple editor de texto, a diferencia de los binarios que necesitan software específico (visores multimedia, descompresores, compiladores, etc.)

Esta característica hace que los documentos sean independientes del sistema operativo o programa con el que fueron creados, esto facilita la interoperabilidad, que constituye una importante ventaja para el intercambio de información en Internet.

COMPACIDAD

Las instrucciones de marcado se mezclan con el propio contenido, por ejemplo, `<h2>Contenido</h2>`.

El código entre corchetes como `<h2>`, son instrucciones de marcado, también llamadas etiquetas. Esta etiqueta en concreto es una etiqueta de presentación, indica que el texto comprendido debe tener el formato asignado a la cabecera nº 2.

El texto entre las marcas es el propio contenido del documento.

INDEPENDENCIA DEL DISPOSITIVO FINAL

El mismo documento puede ser interpretado de diferentes formas dependiendo del dispositivo final, así tendremos diferentes resultados si se usa un dispositivo móvil, un ordenador de sobremesa o una impresora.

ESPECIALIZACIÓN

Inicialmente los lenguajes de marcas se idearon para visualizar documentos de texto, pero progresivamente se han empezado a utilizar en muchas otras áreas como gráficos vectoriales, sindicación de contenidos, notación científica, interfaces de usuario, síntesis de voz, etc.

FLEXIBILIDAD

Los lenguajes de marcas se pueden combinar en el mismo archivo con otros lenguajes, como HTML con PHP y JavaScript. Incluso hay etiquetas específicas para ello como es `<script>`.

XML ha permitido que se puedan combinar varios lenguajes de marcas diferentes en un mismo archivo, como en el caso de XHTML con MathML y SVG.

1.4. Clasificación de los lenguajes de marcas

Normalmente los lenguajes de marcas se suelen clasificar en tres tipos, atendiendo al tipo de marcas que utilizan:

• **De presentación**:

Indican el formato del texto o tipografía, sin especificar su estructura, por ejemplo aumentar el tamaño de la fuente, centrar o cambiar a negrita.

Esta categoría incluye los lenguajes de procedimiento que agrupan varias marcas de presentación en una macro. Por ejemplo, para formatear un título, debe haber una serie de directivas inmediatamente antes del texto indicando: tamaño de letra 16p, fuente Arial, negrita. Justo después del título debe haber etiquetas inversas que anulen el formato, para continuar con el texto normal.

El software que representa el documento debe interpretar el código en el mismo orden en que aparece.

Los procesadores de texto y en general las aplicaciones de edición profesional utilizan este tipo de marcado.

Ejemplos: - nroff, troff, RTF

- TeX, Docbook (derivados de SGML)

• **Descriptivo, estructural o semántico**:

Indican las diferentes partes en las que se estructura el documento, pero sin especificar cómo deben representarse ni en qué orden.

XML es un metalenguaje expresamente diseñado para generar marcado descriptivo y los lenguajes derivados de XML con este propósito son: EBML, RDF, XFML, OWL y XTM. Aunque XML almacena información de todo tipo, los demás tienen contenido específico.

Estos lenguajes crean documentos con estructura en árbol que almacenan información, por eso son bases de datos, sin embargo no usan tablas ni respetan las reglas de integridad propias de las BD Relacionales, por ello se les llama bases de datos semiestructuradas.

Ejemplos: - ASN.1, YAML.

- EBML, RDF, XFML, OWL, XTM (derivados de XML)

• **Híbrido**:

Lenguajes que contienen marcas de los dos tipos anteriores indistintamente.

Ejemplos: - HTML (derivado de SGML)

- XHTML, WML (derivados de XML)

Una clasificación de lenguajes de marcas atendiendo a su funcionalidad:

- **Para crear documentación electrónica:**

- RTF, TeX, troff, nroff
- ASN.1, EBML, YAML
- Wikitexto, DocBook, LinuxDoc

- **Tecnologías de Internet:**

- HTML, XHTML, WML (páginas web)
- GladeXML, XForms, XAML (formularios/interfaces de usuario)
- RSS, Atom (sindicación de contenidos)
- WSDL, SOAP, UDDI (servicios web)
- XMPP (mensajería instantánea)

- **De propósito específico:**

- MathML, CML (fórmulas matemáticas)
- SSML, SRGS, VoiceXML (síntesis de voz)
- MusicXML (partituras de música)
- SVG, VML, X3D (gráficos vectoriales)
- SMIL (archivos multimedia)
- XLL (enlaces):
    - XLINK (asociación de recursos)
    - XML Base (URI básico)
    - XPOINTER (localización de recursos)
- XSLT (transformación de documentos)
- XTM (mapas conceptuales)
- RDF, XFML, OWL, XMP (catalogación y clasificación de documentos, metadatos)
- GML (información geográfica)
- OFX (intercambio de información financiera)
- ebXML (comercio electrónico)
- XML Dsig, XML Enc, SAML, XACML, XKMS, XrML (seguridad)
- XInclude (inclusión de archivos)

1.5. Organizaciones y estándares

Podemos definir la normalización o estandarización como el proceso de especificación de normas, para garantizar el correcto funcionamiento de elementos construidos de forma independiente.

Aplicado al contexto de los lenguajes de marcas, sería por ejemplo el desarrollo de páginas web atendiendo a las especificaciones oficiales del lenguaje utilizado.

Para la definición de estas normas existen organismos internacionales, nacionales incluso organizaciones privadas.

Las organizaciones más importantes en materia de software son W3C, ISO y Open Source.

Según el propio W3C:

"El World Wide Web Consortium (W3C) es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo."

El W3C recibe ingresos de las cuotas de sus miembros, de becas de investigación, subvenciones y donaciones privadas. Por tanto, no se trata de una empresa con fines lucrativos sino de una comunidad heterogénea formada por diferentes organismos que son miembros, un grupo de documentación técnica y los grupos de trabajo formados por expertos, que son quienes fabrican principalmente los estándares.

Toda organización de estándares pretende desarrollar normas que sean de amplio seguimiento por parte de la comunidad, para lo cual es imprescindible el consenso con las empresas involucradas como los navegadores, buscadores, desarrolladores web y fabricantes de dispositivos móviles.

Entre sus miembros se encuentran las principales empresas del sector como Microsoft, Apple o Google entre otras.

Además de producir estándares, la Comunidad W3C ha creado Software de Código Abierto, siendo el más conocido el validador W3C, que nos será de utilidad con HTML, CSS, contenido mobileOK y otras tecnologías.

Actividad 1.4:

1. Visita la página del W3C, http://www.w3c.es/ y consulta la especificación HTML 4.01.
2. Averigua el estado de las publicaciones sobre HTML 5.
3. Consulta los objetivos del W3C.
4. Consulta el listado de miembros españoles.

1.6. Introducción a los principales lenguajes de marcas

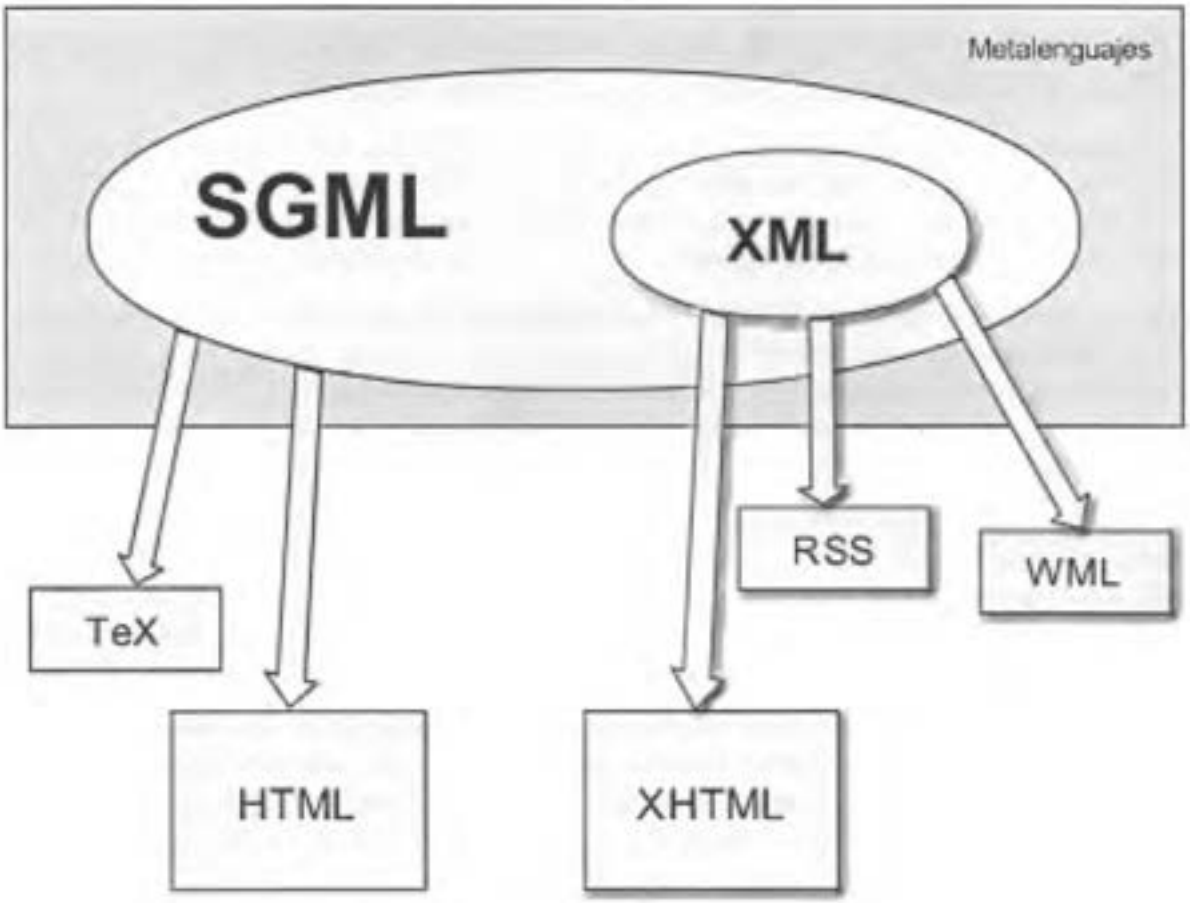


Figura 1.4: Principales lenguajes de marcas

El origen de los lenguajes de marcas como ya sabemos es SGML, del cual se derivan directamente algunos lenguajes como HTML. También se creó a partir de él por simplificación XML, otro metalenguaje más fácil de usar y entender. A partir de XML se han creado muchos lenguajes como XHTML, RSS y un largo etc.

1.6.1. HTML

HTML es un lenguaje destinado a la creación de páginas web, que nos permite mezclar en un mismo documento, texto y contenido multimedia. Además dispone de una potente herramienta que son los enlaces o hipervínculos, para conectar con otras páginas cuyo contenido está relacionado la página actual. Esto supone una ventaja importante frente a los documentos impresos (libros, revistas, etc.)

HTML fue creado por Tim Berners Lee[2] a principios de los 90 con objetivos divulgativos. No pensó que la web llegara a ser un área de ocio y un medio de comunicación tan potente, de modo que inicialmente no podía dar soporte a todas las funciones que más tarde tendría que realizar.

Estas carencias, se han ido resolviendo mediante la incorporación sucesiva de modificaciones y nuevos elementos, son las diferentes versiones de HTML.

Esta evolución poco planificada y a veces anárquica de HTML, ha supuesto una serie de inconvenientes y deficiencias, que han sido superados con la introducción de otras tecnologías capaces de mejorar el funcionamiento y la organización de los sitios Web. Ejemplos son CSS, JavaScript y los complementos de navegador.

Otro de los problemas que tiene HTML, especialmente cuando se combina con CSS, ha sido la diferente interpretación que hacen los navegadores de la misma página web, debido al diferente ritmo en que los navegadores incorporan las nuevas propiedades CSS (especialmente Internet Explorer). Esto ha provocado la aparición de los "hacks", que son pequeños trozos de código para conseguir un comportamiento homogéneo en los navegadores.

Además del navegador necesario para ver los resultados de nuestro trabajo, necesitamos otra herramienta capaz de crear la página. Un archivo HTML es texto plano, por lo que para escribir en HTML necesitamos un simple editor de textos. Existen aplicaciones específicas para la creación de páginas web, quizás la más conocida sea Dreamweaver, que nos ofrecen muchas funcionalidades para aumentar nuestra productividad.

No obstante, es aconsejable en un principio utilizar una herramienta lo más sencilla posible, para poder prestar la máxima atención a nuestro código en lugar de a la aplicación. Uno de los editores de texto recomendables para desarrollo web es Notepad++ http://notepad-plus-plus.org/

Se trata de software libre bajo licencia pública general de GNU para Windows. Como características útiles dispone de indentación y resalte de paréntesis, syntax highlight (coloreado de palabras reservadas), organización de archivos en pestañas, extensiones, macros, etc.

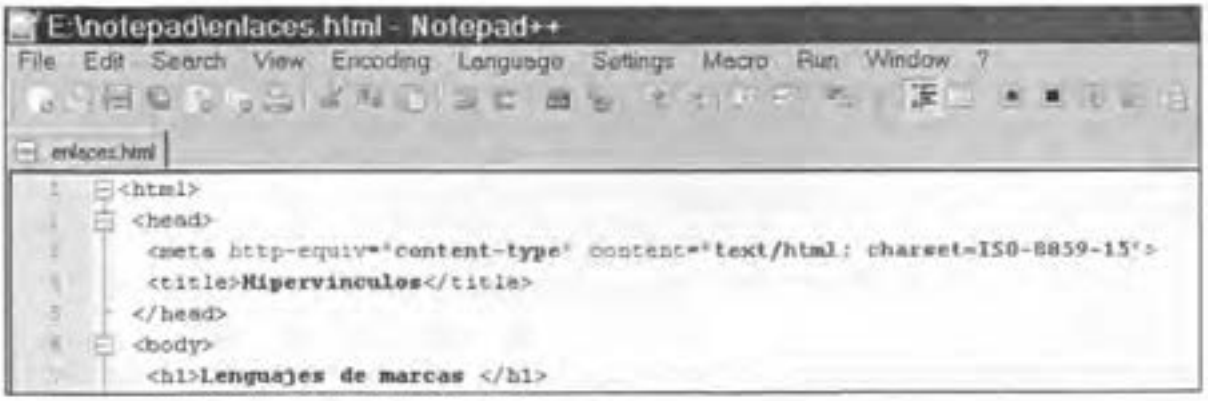


Figura 1.5: Editor de texto Notepad++

[2] Tim Berners-Lee y su equipo desarrollaron el lenguaje HTML, el protocolo http y el sistema de localización de recursos URL, después pusieron en marcha el primer servidor Web llamado httpd, por todo ello es considerado como el padre de la Web.

1.6.2. XML

Es una simplificación y adaptación de SGML que permite definir lenguajes específicos. Por lo tanto, XML no es un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, es decir, lo que hemos llamado un metalenguaje. Para describir la relación con SGML a menudo se utiliza la regla 80/20: 80% de funcionalidad y 20% de complejidad.

Algunos de los lenguajes que se basan en XML para su definición son XHTML, SVG, MathML, RSS, etc.

Como características podemos citar:

- Extensible: se pueden definir nuevas etiquetas

- Versátil: separa contenido, estructura y presentación

- Estructurado: se pueden modelar datos a cualquier nivel de complejidad

- Validable: cada documento se puede validar frente a un DTD/Schema

- Abierto: independiente de empresas, sistemas operativos, lenguajes de programación o entornos de desarrollo.

- Sencillo: fácil de aprender y de usar.

XML no se utiliza solo en Internet, sino que se está convirtiendo en un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos ligeras, editores de texto, hojas de cálculo, transacciones comerciales y en general donde se necesite almacenar información sin las restricciones de un SGBD Relacional.

```
asir.xml
1  <?xml version="1.0" encoding="ISO-8859-15"?>
2
3  <!-- Módulos de ASIR -->
4  <asir>
5      <modulo><titulo>Lenguajes de Marcas</titulo></modulo>
6          <contenido>
7              <unidad>Introducción</unidad>
8              <unidad>HTML</unidad>
9              <unidad>CSS</unidad>
10             <unidad>XML</unidad>
11             <unidad>XLST</unidad>
12             <unidad>Sindicación</unidad>
13             <unidad>ERP</unidad>
14         </contenido>
15     </modulo>
16 </asir>
```

Figura 1.6: asir.xml

CAPÍTULO 2

HTML y XHTML. Lenguajes para la Web

CONTENIDO

Evolución histórica
Estructura del documento
Elementos de HTML
Contenido de la cabecera
Contenido del cuerpo
XHTML
HTML 5
Validación
Ejercicios propuestos

2.1. Evolución histórica

HTML (HyperText Markup Language) aparece a principios de los años noventa cuando el investigador Tim Berners Lee, lo desarrolla para potenciar la colaboración entre físicos e investigadores de la energía nuclear.

El objetivo principal era enlazar los documentos mediante links (hipervínculos), de forma que al hacer clic sobre algún texto, se abriera otro documento relacionado con la información seleccionada.

En 1991 se publica el primer documento formal con el nombre de "HTML tags". Esta primera versión tiene muchas limitaciones, por ello organismos internacionales colaboraron para desarrollar mejoras en el lenguaje.

En 1993 hubo un primer intento de realizar una especificación oficial de HTML, sin embargo no fue reconocida. Se trataba de la versión HTML 1.2.

En 1995 se publica el primer estándar oficial llamado HTML 2.0 por el organismo IETF (Internet Engineering Task Force).

En 1997 se publica la versión HTML 3.2 por la organización W3C (World Wide Web Consortium), que será a partir de ahora y hasta nuestros días, el organismo oficial encargado de publicar los estándares oficiales. Incluye tablas y applets de Java.

En 1998 aparece la versión HTML 4.0 que incluye muchas mejoras como soporte para lenguajes script, hojas de estilo css, facilidades de impresión, accesibilidad para discapacitados, etc. En este mismo año aparece también la primera especificación de XML. Al año siguiente se publica HTML 4.01 la última especificación oficial hasta el momento, se trata de una mera revisión que no incluye novedades significativas.

En el año 2000 el W3C publica XHTML 1.0, se trata de una reformulación de HTML 4.0 basada en XML. Cumpliendo unas directrices de compatibilidad puede funcionar sobre aplicaciones creadas para HTML, pero a la vez es conforme a XML, lo que le proporciona gran flexibilidad para añadir nuevos elementos y módulos.

XHTML 1.0 se ha convertido en un estándar aceptado y ampliamente utilizado por la comunidad de desarrolladores Web.

W3C siguió trabajando por este camino en un nuevo lenguaje XHTML 2.0, pero que ya no era compatible ni con HTML ni siquiera con XHTML 1.0.

En el año 2003 aparecieron los famosos XFORMS, los formularios de XHTML que mejoran considerablemente a los anteriores, gracias a que separan perfectamente el contenido de la presentación. Esta característica les permite funcionar sobre un portátil, teléfono móvil, agenda electrónica, etc.

Los XFORMS despertaron un renovado interés por desarrollar HTML en lugar de sustituirlo, ya que XML dependía de la implementación en los navegadores de nuevas tecnologías como RSS o Atom.

En el año 2004 las empresas Apple, Mozilla y Opera crearon un grupo de trabajo llamado WHATWG con el objetivo de continuar el desarrollo de HTML.

Una de sus premisas de trabajo era mantener total compatibilidad con versiones anteriores.

En el 2007 los grupos de trabajo W3C y WHATWG deciden unirse para el desarrollo de una especificación de HTML 5, lo que permite publicar a principios de 2008 el primer borrador de HTML 5.

En el 2009 el consorcio W3C decide poner fin al desarrollo del estándar XHTML 2.0 a favor de HTML 5.

En la actualidad siguen colaborando juntos y se prevé una especificación oficial en breve.

AÑO	EVENTO
1991	Nacimiento. "HTML tags"
1993	HTML 1.2
1995	HTML 2.0
1994	Se funda W3C
1997	HTML 3.2
1998	HTML 4.0 y XML
1999	**HTML 4.01**
2000	**XHTML 1.0**
2001	XHTML 1.1
2002	XHTML 2.0
2003	XFORMS
2004	Se funda WHATWG
2008	Borrador de HTML 5

Tabla 2.1: Evolución del lenguaje HTML

HTML 4.01 y XHTML 1.0 son las especificaciones más utilizadas.

HTML 5 se va incorporando progresivamente.

La decisión de elegir uno u otro lenguaje depende de varios factores, entre ellos el tamaño y el número de usuarios del sitio Web.

Solo como orientación:

- Para grandes empresas y sitios comerciales con mucho público, sería aconsejable usar XHTML y CSS avanzado en varias hojas de estilos.

- Para sitios Web pequeños y páginas personales se puede utilizar HTML por su simplicidad y una hoja de estilos sencilla.

2.2. Estructura del documento

Los documentos HTML son archivos de texto plano formados por etiquetas de apertura y cierre (por ejemplo: <html>... </html) que permiten dar formato al contenido del documento.

Una página web es el resultado que ofrece un navegador al interpretar dicho documento.

La estructura básica de un documento HTML es:

```
<! Declaración de Tipo de Documento >
<html>

  <head >
    <title> Título de la página </title>
    Elementos opcionales de la cabecera
  </head>

  <body>
   Contenido del cuerpo
  </body>

</html>
```

Tanto la declaración de tipo de documento, como las etiquetas html, head y body son opcionales, pero deben incluirse para que el documento sea correcto y legible.

La etiqueta `<html>` es la raíz del documento por lo que head (cabecera) y body (cuerpo) deben estar totalmente contenidas en ella.

Además head y body no deben solaparse entre sí, por lo tanto, el documento debe tener un gran bloque principal (html) y dos subbloques al mismo nivel.

Estos dos bloques a su vez, pueden tener otros subbloques que tampoco deben solaparse.

Es buena costumbre separar los bloques importantes por líneas de comentarios.

La declaración de tipo de documento se coloca al principio, fuera de todo bloque.

En unidades posteriores estudiaremos las DTD (Document Type Definition - Definición de Tipo de Documento), por ahora es suficiente saber que se trata de un documento externo con extensión .dtd y que existen 3 posibilidades: estricta, de transición y con marcos.

Al indicar la DTD al principio del documento le decimos al navegador cómo interpretar adecuadamente el código.

HTML 4.01 recomienda separar por completo el contenido del formato.

DTD estricta:
Si cumplimos escrupulosamente con esta recomendación, almacenando toda la información sobre el formato en una hoja de estilos CSS, entonces podemos utilizar la declaración estricta: `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">`
DTD de transición:
Si queremos combinar contenido e información de formato en nuestro documento HTML, entonces debemos utilizar la declaración de transición: `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
DTD con marcos:
Si queremos incluir marcos en nuestro documento, entonces tenemos que usar una DTD que es muy similar a la anterior, pero que sustituye el elemento body por frameset, en este caso la declaración sería: `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">`

En HTML 5 la declaración es más simple: `<!DOCTYPE html>`

Ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Mi primera página</title>
   </head>
   <body>
      <h1>Hola Mundo</h1>
   </body>
</html>
```

Actividad 2.1:

Copia este código en un archivo de texto, guárdalo con el nombre mipagina.html y luego observa el resultado en el navegador.

2.3. Elementos de HTML

Los elementos son los componentes básicos del lenguaje HTML. Cada elemento puede tener atributos y contenido. La mayoría de elementos tienen una etiqueta de inicio, una etiqueta de fin y un contenido encerrado entre ambas. Si no tienen contenido se les llama vacíos.

Los atributos de un elemento (casi siempre opcionales) se colocan en la etiqueta de apertura.

Sintaxis.

- La sintaxis básica para un elemento no vacío tiene la forma:

  <nombre_elemento atributo1="valor1" atributo2="valor2"...>

  contenido del elemento

  </nombre_elemento>

- La sintaxis para un elemento vacío (V):

  <nombre_elemento atributo1="valor1" atributo2="valor2"... >

2.3.1. <html>

Definición: delimita el contenido del documento.

Aparición: apertura y cierre opcionales.

Atributos: `dir, lang, version`.

- `dir`: indica el sentido de lectura de texto cuando no corresponde al habitual (árabe).

  Valores: ltr (de izquierda a derecha) y rtl (de derecha a izquierda).

- `lang`: indica el idioma por defecto del documento.

  Valores: de (alemán), en (inglés), es (español), fr (francés), etc.

- `version`: indica la versión del lenguaje HTML, está desaprobado porque ya se proporciona en la declaración `<!doctype>`.

Contenido de html: `<head>` y `<body>`.

Ejemplo:

```
<html lang="es" dir="ltr">
  <head>...</head> <body>...</body>
</html>
```

Actividad 2.2:

Indica que mipagina.html está en español y con dirección rtl, observa el resultado, después deja la dirección definitiva en ltr.

2.3.2. <head>

Definición: delimita la cabecera de un documento.

Aparición: apertura y cierre opcionales.

Atributos: **lang, dir, profile.**

- **profile**: indica donde están los perfiles para interpretar los elementos meta.

  Valores: URI (Uniform Resource Identifier – Identificador Uniforme de Recurso), dirección web completa del recurso.

Contenido de la cabecera: **<title>, <base>, <meta>, <link>, <object>, <script> y <style>.**

Todos los elementos son opcionales excepto **<title>** que es obligatorio.

El contenido de la cabecera es una información general de todo el documento, por eso se escribe al principio y es lo primero que analiza el navegador.

2.3.3. <body>

Definición: delimita el cuerpo de un documento.

Aparición: apertura y cierre opcionales.

Atributos: **lang, dir, id, class, title, style, onload, onunload** y los eventos intrínsecos.

- **id**: sirve para asignar un nombre único a un elemento.

  Se utiliza para identificar elementos en hojas de estilo y en scripts.

- **class**: sirve para asignar un nombre de clase a un elemento.

  Es posible usar el mismo nombre de clase para varios elementos.

  Se usa para mejorar el rendimiento de las hojas de estilo.

- **title**: sirve para agregar un comentario asociado a un elemento.

  Los navegadores muestran este comentario en una ventana emergente cuando el ratón se sitúa sobre el elemento.

- **style**: sirve para aplicar información de estilo a un elemento.

- **onload**: este evento ocurre cuando el navegador finaliza la carga de un documento.

- **onunload**: este evento ocurre cuando el navegador elimina un documento de una ventana o marco.

Atributos desaprobados: **background, text, link, alink, vlink, bgcolor.**

Estos atributos no deben usarse, ya que la forma correcta es a través de hojas de estilo o con estilos en línea como en el siguiente ejemplo.

Ejemplo:

```
<body style="background-color: blue"> ... </body>
```

Actividad 2.3:

Cambia el color de fondo con el atributo style y con el atributo desaprobado bgcolor.

Comprueba que se obtiene idéntico resultado.

2.3.4. Comentarios <!-- -->

Definición: sirve para insertar una o varias líneas con comentarios que no serán interpretadas por el navegador. No se pueden anidar.

Ejemplo:

```
<!--mipagina.html-->
<html>
 <head> ... </head>
 <!-- Aquí empieza el cuerpo del documento -->
 <body> ... </body>
</html>
```

Actividad 2.4:

Agrega comentarios a mipagina.html para separar las partes del documento.

2.3.5. Conjuntos de atributos

Los atributos de los elementos html se organizan en grupos, de modo que se usarán los siguientes conjuntos cada vez que se describa un elemento, para evitar poner la lista completa.

%i18n = **lang, dir**

%coreattrs = **id, class, style, title**

%events = **onload, onunload, onblur, onchange, onfocus, onreset, onselect, onsubmit, onabort, onkeydown, onkeyup, onkeypress, onclick, ondblclick, onmousedown, onmouseover, onmouseout, onmousemove, onmouseup**

%attrs = %i18n + %coreattrs + %events

Listado de eventos intrínsecos (%events) y su descripción:

Eventos de **<body>** y **<frameset>**

Atributo	Descripción
onload	Ocurre cuando termina la carga completa de un documento
onunload	Ocurre cuando el navegador elimina un documento

Eventos de **formulario**

Atributo	Descripción
onblur	Ocurre cuando un campo pierde el foco
onchange	Ocurre cuando un campo pierde el foco y su valor se modifica después
onfocus	Ocurre cuando un campo recibe el foco
onreset	Ocurre cuando se reinicia un formulario
onselect	Ocurre cuando se selecciona un texto de un campo de texto
onsubmit	Ocurre cuando se envía un formulario

Eventos de **imagen**

Atributo	Descripción
onabort	Ocurre cuando la carga de una imagen se interrumpe

Eventos con el teclado

Atributo	Descripción
onkeydown	Ocurre cuando se pulsa una tecla encima de un elemento
onkeypress	Ocurre cuando se pulsa y se suelta una tecla encima de un elemento
onkeyup	Ocurre cuando se suelta una tecla encima de un elemento

Eventos con el ratón

Atributo	Descripción
onclick	Ocurre cuando se hace click sobre un elemento
ondblclick	Ocurre cuando se hace doble click sobre un elemento
onmousedown	Ocurre cuando se pulsa el botón del ratón encima de un elemento
onmousemove	Ocurre cuando el ratón se mueve estando encima de un elemento
onmouseout	Ocurre cuando el ratón sale de un elemento
onmouseover	Ocurre cuando el ratón se sitúa sobre un elemento
onmouseup	Ocurre cuando el botón del ratón se suelta estando encima

(Válidos para todos los elementos excepto <base>, <bdo>, <br>, <frame>, <frameset>, <head>, <html>, <iframe>, <meta>, <param>, <script>, <style> y <title>.)

Para utilizar estos eventos intrínsecos hay que conocer un lenguaje script, que sirva para ejecutar una determinada acción cuando se produzca el evento.

Ejemplo:

```
<body onload="alert('Carga completa');">
```

(Observación: asegúrate de utilizar el carácter comilla simple del teclado)

2.4. Contenido de la cabecera

Elementos:

<title>, <base>, <meta>, <link>, <object>, <script> y <style>.

2.4.1. <title>

Definición: indica el nombre del sitio Web en la barra superior o pestaña del navegador.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **lang, dir.**

Ejemplo:

```
<head>
  <title> Mi primera página </title>
</head>
```

2.4.2. <base>

Definición: indica la dirección raíz del sitio Web, la cual permite resolver las direcciones relativas.

Aparición: sin etiqueta de cierre (V).

Atributos: **href.**

- **href**: se usa siempre para indicar la dirección raíz del documento.

Ejemplo:

```
<base href="http://www.juanmacr.es/">
```

En caso de tener una dirección relativa como "/ASIR.html", se completaría dando lugar a la dirección absoluta: "http://www.juanmacr.es/ASIR.html".

Actividad 2.5:

· Elige un tema para tu sitio Web diferente del resto de compañeros.

· Date de alta en un hosting gratuito.

(Si prefieres un hosting de pago tendrás que elegir también tu dominio)

· Crea un nuevo espacio.

· Sube el contenido de mipagina.html y renómbrala como index.html

· En title indica el nombre de tu empresa.

· Comprueba que tu sitio está publicado en Internet.

2.4.3. <meta>

Definición: indica un conjunto de propiedades generales del documento como el autor, descripción, palabras clave, herramienta utilizada, tipo de contenido, etc.

Estas propiedades evolucionan continuamente, es decir, se crean nuevas propiedades para ampliar la información asociada al documento, las redes sociales son un ejemplo.

Normalmente estas propiedades pasan desapercibidas para el usuario habitual.

Aparición: sin etiqueta de cierre (V).

Atributos: **lang, dir, name, content, http-equiv, scheme.**

- **name:** indica el nombre de una propiedad, no hay una lista oficial de propiedades pero podemos citar los siguientes valores: abstract, author, copyright, date, description, distribution, expires, generator, google-site-verification, keywords, language, no-email-collection, organization, rating, reply-to, revisit-after, robots y la familia de etiquetas Dublin Core.

- **content:** contiene el valor de la propiedad indicada antes por **name**, por tanto, estos dos atributos funcionan simultáneamente proporcionando parejas de valores.

  Ejemplo:

  ```
  <meta name="description" content="Lenguajes de Marcas">
  ```

- **http-equiv**: indica una propiedad al navegador en forma de cabecera http, como si el propio servidor http (Web) hubiera generado dicha cabecera, de ahí viene el nombre "http-equivalent".

  Valores: cache-control, content-type, set-cookie, content-disposition, pics-label, pragma, refresh, resource-type, content-script-type, content-style-type, window-target.

  Tiene muchas utilidades como por ejemplo, indicar el conjunto de caracteres a utilizar en nuestro sitio Web, sin necesidad de modificar el servidor. También funciona en coordinación con el atributo **content** para proporcionar parejas de valores.

  Ejemplo:

  ```
  <meta http-equiv="content-type"
      content="text/html; charset=iso-8859-15">
  ```

  Esto indica que el contenido de la página es texto plano HTML con el conjunto de caracteres iso-8859-15, que corresponde al alfabeto latino nº 9.

- **scheme:** indica al navegador que tiene que interpretar los metadatos atendiendo al perfil especificado, para evitar ambigüedad.

  Ejemplo:

  ```
  <meta scheme="Europe" name="date" content="24-10-2012">
  ```

  Esto indica que la fecha se lee según el perfil "Europe", es decir: "dd-mm-aaaa".

Ahora vamos a estudiar las principales meta etiquetas en función de su utilidad.

Se clasifican en dos grupos (por sus dos atributos principales, aunque hay más):

M E T A

1) **Las <meta name>:** se utilizan principalmente para optimizar el resultado de los motores de búsqueda, es decir, para ayudar a los buscadores de páginas Web.

   La información que manejan no se presenta en pantalla.

2) **Las <meta http-equiv>:** se utilizan para dar instrucciones a los navegadores, es decir, ejercen cierto control sobre ellos.

   Tienen influencia sobre lo que aparece en pantalla.

1) Meta name

· Información general sobre la página:

`abstract, author, copyright, date, generator`

· Información específica para buscadores:

`description, distribution, keywords, locality, rating, revisit-after, robots`

· Para administración del sitio:

`google-site-verification, reply-to`

2) Meta http-equiv

· Tipo de Contenido:

`content-type, content-language, content-script-type`

· Manejo de cookies:

`set-cookie`

· Actualización/Redirección de la página:

`refresh`

· Transiciones de página:

`page-enter, page-exit`

· Control parental:

`pics-label`

· Manejo de la cache:

`cache-control, pragma, last modified, expires`

2.4.3.1. Meta name

- Información general:
  - **abstract**

    Proporciona un resumen muy corto de la etiqueta "description".

    No es utilizada por todos los motores de búsqueda al contrario que description.
  - **author**

    Proporciona el nombre del autor del documento actual.

    Es recomendable indicar el nombre completo con nombre y apellidos.

    ```
    <meta name="author" content="Joel Castro León">
    ```

    (Para páginas desarrolladas por un equipo o para indicar un destinatario de correo diferente al autor se puede utilizar la etiqueta web_author)
  - **copyright**

    Sirve para indicar que hay información bajo derechos de autor.

    ```
    <meta name="copyright" content="Editorial Garceta 2012">
    ```

    Es recomendable sin embargo, utilizar un icono o enlace fácilmente visible para especificar que el documento está bajo derechos de autor.

Figura 2.1: Copyright

Actividad 2.6:

Visita la página `http://es.creativecommons.org/` y crea una licencia para tu sitio Web con objetivos no comerciales, luego inserta el código en tu página.

  - **date**

    Sirve para indicar la fecha de creación de la página.

    ```
    <meta name="date" content="24 de Febrero de 2010">
    ```
  - **generator**

    Sirve para indicar la herramienta utilizada para construir el sitio web.

    ```
    <meta name="generator" content="KompoZer">
    ```

- Información específica para buscadores:

  - **description**

    Sirve para explicar el contenido y objetivos del sitio web.

    Esta información es utilizada por los motores de búsqueda para indexar la página, así que resulta importante el texto que coloquemos.

    ```
    <meta name="description" content="Lenguajes de marcas y sistemas
    de gestión de información. Editorial Garceta. 2012">
    ```

    El texto indicado en la descripción suele aparecer como resultado de la búsqueda realizada por un usuario.

  - **distribution**

    Se utiliza para indicar el nivel de distribución que queremos para nuestro sitio web.

    Niveles: Global (Internet) | Local (reservado para la red local) | IU (Uso interno).

    ```
    <meta name="distribution" content="global">
    ```

    No es recomendable utilizar esta etiqueta, la forma adecuada de restringir el acceso a nuestro sitio es configurar adecuadamente el archivo .htaccess de nuestro servidor http, o bien crear un archivo robots.txt, para que los motores de búsqueda indexen según nuestras preferencias.

  - **keywords**

    Sirve para indicar una lista de palabras clave acerca de nuestro sitio web.

    Es una etiqueta muy útil para que nuestro sitio sea localizable, sin embargo debe utilizarse correctamente. La inclusión de palabras no relacionadas con nuestro sitio con el fin de mejorar el posicionamiento web, se considera spam y puede ser penalizado por los motores de búsqueda.

    ```
    <meta name="keywords" content="Ciclos, Formativos,
                    Informática, FP, DAW, DAM, ASIR, SMR">
    ```

  - **locality**

    Sirve para indicar al robot de indexación la localidad donde se ubica el sitio web.

    ```
    <meta name="locality" content="Madrid, España">
    ```

Actividad 2.7:

Incluye estas etiquetas en tu página cambiando el contenido de acuerdo a tu empresa y visualízalas con el navegador Firefox.

  - **rating**

    Sirve para recomendar el visionado de la página según la edad de un modo similar a las películas en el cine. Existen alternativas mejores como Pics-label.

    ```
    <meta name="rating" content="general">
    ```

- **revisit-after**

Sirve para indicar a los motores de búsqueda, que revisiten nuestra página para indexarla de nuevo, ya que ha sido modificada.

```
<meta name="revisit-after" content="3 days">
```

Los motores de búsqueda suelen tener su propio patrón de retorno a las páginas ya indexadas, así que no siempre cumplen el contenido de esta etiqueta.

- **robots**

Sirve para controlar los robots[1] de los motores de búsqueda.

Los valores posibles son:

Valor	Utilidad
index	Indica al robot que indexe esta página
follow	Indica al robot que siga los hipervínculos de esta página para indexarlos también.
all	Equivale a los dos anteriores juntos.
noindex	No indexa la página.
nofollow	No sigue los hipervínculos.
none	Equivalente a las dos anteriores juntas.
noarchive	No permite a los buscadores almacenar la página en caché
noodp	No permite usar la descripción oficial de la página que hay en Wikipedia (si la hubiera).
noydir	Equivalente a la anterior para Yahoo.

Tabla 2.2: Valores para el archivo robots

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

Además de usar esta etiqueta, los motores de búsqueda se comunican con el sitio Web mediante el archivo robots.txt, que debe crear el administrador del sitio y colocarlo en el directorio raíz.

El archivo robots.txt es de fácil creación, se trata de texto plano que se puede generar con el bloc de notas. Su objetivo primordial es limitar la indexación generalizada de las páginas web de nuestro sitio.

Utiliza unas pocas palabras clave como:

User-agent (robot), **Allow** (permitir), **Disallow** (prohibir)

[1] Robot o araña es un programa de rastreo automático de páginas web, es usado por los buscadores en Internet para

Y los valores:

* (todos los robots), / (todos los directorios).

Para evitar directorios o archivos en particular hay que deshabilitarlos por separado.

El archivo se lee de arriba abajo ordenadamente, para cada robot se pueden prohibir los directorios y archivos deseados y cada user-agent es independiente del siguiente.

Ejemplos:

```
User-agent: *
Allow: /
```

Permite a todos los buscadores indexar todas las páginas, sería equivalente a no crear el archivo.

```
User-agent: *
Disallow:
```

Lo mismo que el anterior.

```
User-agent: *
Disallow: /imagenes/
Disallow: /js/
```

Impide a todos los buscadores indexar los directorios: imagenes y js.

```
User-agent: *
Disallow: /
User-agent: googlebot
Disallow: /
Allow: public
```

Impide a todos los robot indexar todas las carpetas y permite al robot googlebot indexar solamente el directorio /public.

Hay que tener en cuenta que a pesar de evitar la indexación de una carpeta, si algún recurso es referenciado por un enlace, entonces aparecerá en las respuestas del buscador. Para evitar esto hay que utilizar la metaetiqueta robots con "noindex, nofollow".

Actividad 2.8:

- En tu nuevo sitio crea dos carpetas una llamada "public" y otra "private".
- Crea un archivo robots.txt que no permita indexar la carpeta private y súbelo al directorio raíz.
- Inserta una etiqueta meta para que no siga ningún hipervínculo.

- Para administración del sitio

  - **google-site-verification**

    Sirve para identificar al administrador del sitio en las herramientas de administración de sitios de google. Estas herramientas permiten a un administrador analizar varios parámetros: como su visibilidad, las búsquedas que recibe el sitio, sitemaps, errores, etc.

```
<metaname="google-site-verification"
content="8EYm4qdOa4U0L8tRw9clHsjSbo_JOK0Jjlxaa4OtQAA">
```

Figura 2.2: Herramientas para webmasters de Google

Actividad 2.9:

- En herramientas para webmasters de Google solicita un código de verificación, mediante la opción "métodos alternativos", insertar etiqueta meta.
- Insértalo en tu página.
- Comprueba que tu sitio ha sido verificado.
- Visita periódicamente el sitio y observa la variación en las estadísticas.

  - **reply-to**

    Sirve para indicar una dirección email para contactar con el administrador del sitio.

    No es nada recomendable usar esta etiqueta pues puede utilizarse como herramienta de spam, la alternativa es construir un formulario de contacto.

  - **Dublin-Core**

    Este proyecto pretende crear un estándar para el uso de metaetiquetas.

```
<meta name="dc.title" content="Enrique Rojo S.A.">
```

2.4.3.2 Meta http-equiv

- Tipo de Contenido:
  - **`Content-type`**

    Sirve para indicar el tipo de los datos contenidos en el cuerpo de la página y en caso de ser texto también indica el juego de caracteres utilizado.

    Según el RFC 822 el campo **`content-type`** se define:

    `Content-Type := tipo "/" subtipo  [";" atributo = valor]`

Los tipos MIME (Multipurpose Internet Mail Extensions – Extensiones Multipropósito de Correo de Internet) representan los tipos de formato válidos para un archivo.

Tipo MIME	Utilidad
application	Datos binarios o información para programas de correo Subtipos: octect-stream, oda, postscript, x-shockwave-flash, pdf
audio	Archivos de sonido, requieren de un periférico para su salida Subtipos: basic, wav, mp3
image	Imágenes, requieren de un periférico para su salida Subtipos: jpeg, gif, x-icon
message	Un mensaje encapsulado Subtipos: rfc828, partial, external-body
multipart	Datos formados por varias partes con diferentes tipos de datos Subtipos: mixed, alternative, parallel, digest
text	Información en forma de texto Subtipos: plain, richtext, html
video	Datos de video, requiere software y hardware específico Subtipos: mpeg, avi

Tabla 2.3: Tipos mime (solo aparecen los más importantes)

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

En este caso usamos el juego de caracteres latín n°1 que es adecuado para nuestro idioma. Si queremos hacer más internacional nuestra página, escribiendo caracteres que no pertenecen a nuestro alfabeto, necesitamos el charset utf-8.

El UTF-8 (Unicode Transformation Format – Formato de Codificación Universal) contiene todos los caracteres posibles y cada uno tiene un código diferente. Para nuestros caracteres específicos como los acentos o la ñ debemos utilizar los siguientes códigos:

á = á é = é í = í ó = ó ú = ú

ñ = ñ (&=AMPERSAND)

Actividad 2.10:

Cambia el juego de caracteres a utf-8

Inserta en el cuerpo de tu página:

Cuidado con las esdrújulas Cuidado con las esdrújulas

Observa la diferencia.

- **`Content-language`**

  Sirve para indicar el lenguaje principal del documento como en **`lang`**.

  `<meta http-equiv="content-language" content="es_es">`

  Puede ser utilizado por los robots para categorizar el sitio web por el idioma.

  La lista oficial de códigos de idioma está definida por el IANA.

- **`content-script-type, content-style-type`**

  Sirve para indicar el lenguaje de programación por defecto usado en el script y el lenguaje de hoja de estilos por defecto en el documento.

  Normalmente no se usan porque los navegadores los detectan automáticamente.

- **Manejo de cookies:**
  - **`set-cookie`**

    Sirve para guardar una cookie en el navegador del usuario.

    Una cookie es información personal, como el nombre de usuario o sus preferencias, que sirve para mejorar la navegación cuando revisitamos un sitio web.

    Si se especifica la fecha de expiración, la cookie será almacenada hasta el momento indicado, si no se especifica se borrará al cerrar el navegador.

    No es lo más adecuado usar esta etiqueta, el manejo de cookies hoy en día se realiza mediante lenguajes de lado servidor como PHP.

- **Actualización/Redirección de la página:**
  - **refresh**

    Sirve para indicar el tiempo de espera en segundos hasta una actualización o redirección automática de la página.

```
<meta http-equiv="refresh" content="10">
```

    Se utiliza habitualmente en páginas que presentan información en tiempo real.

```
<meta http-equiv="refresh"
      content="10;URL=http://otro_sitio.html">
```

    Puede utilizarse para redireccionar a otro sitio web.

    En el caso de que hayamos cambiado de dominio, es preferible insertar un enlace para que el usuario redireccione o el método de redirección 301 (que requiere insertar código en el lado servidor).

- **Transiciones de página:**
  - **page-enter, page-exit**

    Sirven para realizar transiciones tipo "flash" en la carga de la página y al salir.

```
<meta http-equiv="Page-Enter" content="blendTrans(duration=4)">
<meta http-equiv="Page-Exit" content="revealTrans(duration=1.5,
transition=5)">
```

    Solo tienen finalidad estética, no funcionan en todos los navegadores.

    También puede hacerse mediante CSS3.

Actividad 2.11:

Inserta las etiquetas anteriores modificando los parámetros de duración y transición, comprueba el resultado.

- **Control parental:**
  - **pics-label**

    Sirve para indicar la calificación moral del contenido atendiendo a una serie de parámetros, como desnudez, violencia, lenguaje, comportamientos sociales, etc.

    Existen varias organizaciones de calificación, una de las más conocidas es el ICRA (Internet Content Rating Association – Asociación para Clasificación de Contenidos de Internet), la cual proporciona un generador gratuito de etiquetas.

Ejemplo:

```
<meta http-equiv="pics-Label" content='(pics-1.1
"http://www.icra.org/pics/vocabularyv03/" l gen true for
"http://juanmacr.es" r (n 0 s 0 v 0 l 0 oa 0 ob 0 oc 0 od 0 oe
0 of 0 og 0 oh 0 c 1))'>
```

Es recomendable su uso, sobretodo cuando el contenido no es apto para menores.

Para más información consultar: `http://www.fosi.org/icra/`

- **Manejo de la caché:**

La memoria caché sirve para almacenar la respuesta que envía el servidor, ante una petición del cliente como páginas HTML, imágenes o archivos. De esta forma cuando el cliente solicita de nuevo la misma URI, la caché devuelve la respuesta almacenada en lugar de hacer otra petición al servidor.

El objetivo es mejorar el tiempo de respuesta y reducir el tráfico de red.

Esta memoria puede ubicarse en el navegador del cliente, en servidores Proxy directos como en nuestro ISP (Internet Service Provider - Proveedor de Servicios de Internet) o en servidores Proxy inversos (que sirven de respaldo a sitios web con muchas consultas).

- `cache-control`

Sirve para controlar el cacheo que va a recibir el documento.

Se admiten las siguientes opciones en `content`:

- `max-age` = *nº segundos*

  Indica el número de segundos durante los que el contenido es considerado como válido.

- `s-maxage` = *nº segundos*

  Similar a la directiva max-age, pero aplicable solo para cachés compartidas.

- `public`

  Indica que el documento puede ser guardado por proxies intermedios.

- `private`

  Indica que el archivo puede ser almacenado por el navegador del usuario, pero no por proxies intermedios (la página está personalizada para cada usuario).

- `no-cache`

  Significa que no se debe consultar la caché, por tanto, obliga al navegador a realizar una petición al servidor.

- `no-store`

  Indica al navegador que no guarde el documento en caché después de visualizarlo.

- **must-revalidate**

  Comunica a las cachés que deben cumplir todas nuestras reglas de cacheo.

- **proxy-revalidate**

  Similar a la anterior, para servidores Proxy.

Ejemplos:

```
<meta http-equiv="cache-control" content="max-age=3600">
<meta http-equiv="cache-control" content="no-cache">
```

- **pragma**

Sirve para evitar que la página sea consultada en memoria caché, por tanto, el navegador debe solicitar la página al servidor cada vez que queremos visualizarla.

Ejemplo:

```
<meta http-equiv="pragma" content="no-cache">
```

Equivale a cache-control="no-cache", pero se utiliza por si hay servidores proxy intermedios con protocolo http 1.0.

- **last modified**

Con esta etiqueta el servidor comprueba si la respuesta que el navegador tiene almacenada en caché es válida. El desarrollador web debe modificar la fecha cada vez que actualice la página.

Ejemplo:

```
<meta http-equiv="last-modified"
    content="Mon, 12 Nov 2012 11:35:00 GMT">
```

El navegador envía la fecha "last-modified" junto con la petición de página al servidor web, y éste comprueba si la página está obsoleta o es válida.

Si está obsoleta, envía la página al cliente, pero si es válida envía un mensaje "not modified", entonces el navegador sirve la página que tiene en caché.

- **expires**

Se utiliza para indicar en qué fecha expira la página.

Ejemplo:

```
<meta name="expires" content="Mon, 29 Oct 2012">
```

El formato de fecha según RFC 1123: "día de la semana con tres letras, día del mes con tres letras, año con 4 dígitos, hh:mm:ss GMT".

A veces se utiliza con el valor `content="0"` para evitar que el navegador consulte la caché, pero esto se debe hacer con `cache-control`.

2.4.3.3. El protocolo Open Graph

Es un protocolo que permite identificar los elementos que contiene nuestra página web dentro de una red social. Por ejemplo, para permitir que nuestra web sea conocida en facebook®, y entonces disponga de las mismas ventajas que el resto de páginas.

Podemos incluir un botón "Me gusta" para enviar el contenido de las metaetiquetas. Una vez enviadas los recursos de la página estarán accesibles desde facebook.

Para empezar hay que incluir los siguientes atributos en html:

```
xmlns:fb="http://www.facebook.com/2008/fbml"
xmlns:og="http://ogp.me/ns#"
```

Después las etiquetas meta con la sintaxis:

```
<meta property="og:nombre" content="valor" />
```

- Las propiedades básicas son:

  og:title = El título tal y como aparecerá en la red.

  og:type = El tipo de objeto, por ejemplo "article".

  og:image = La URL de una imagen representativa del objeto.

  og:url – La URL canónica del objeto que será su identificador en la red, por ejemplo, "http://www.mipagina.es/facebook/1618033988".

- Algunas propiedades opcionales:

  og:audio = La URI a un fichero de audio.

  og:description = La descripción con un par de frases.

  og:locality = La localidad del sitio.

  og:site_name = Nombre del sitio.

  og:video = La URI a un fichero de video.

  fb:admins = El identificador del administrador del sitio en Facebook.

Ejemplos:

```
<meta property="og:title" content="mipagina" />
<meta property="og:type" content="article" />
<meta property="og:url" content="http://mipagina.es" />
<meta property="og:image" content="http://foto.jpg" />
<meta property="fb:admins" content="1258775867" />
```

2.4.4. <link>

Definición: define un vínculo a otro documento indicado por **href**.

Aparición: sin etiqueta de cierre (V).

Atributos: **lang, dir, class, id, style, title, eventos intrínsecos, charset, href, hreflang, type, rel, rev y media.**

- **charset**: establece el conjunto de caracteres usado para el documento.
- **href**: indica la URI del documento vinculado.
- **hreflang**: indica el lenguaje del documento vinculado.
- **type**: indica el tipo de contenido del documento vinculado, para que el navegador no lo abra en caso de no tener soporte.

  Los tipos más comunes son: text/html, text/css, image/png, image/gif, image/x-icon, video/mpeg y audio/basic.

  Tipos para documentos xml: application/rss+xml, application/atom+xml.

  Otras: application/x-shockwave-flash, application/opensearchdescription+xml.
- **rel**: establece la relación entre los documentos origen y destino.
- **rev**: indica un vínculo inverso, es decir, desde el destino al origen.
- **media**: indica el medio al que se refiere los datos de estilo en el documento destino.

  Tipos de medios: screen, tty, tv, projection, handheld, print, braille, aural, all.

  Valores básicos para el atributo **rel**:

  · **stylesheet** = se refiere a una hoja de estilos externa.

  · **alternate** = designa una versión alternativa del documento actual.

  Cuando se usa con el atributo **hreflang** implica que hay una versión traducida del documento. Cuando se usa con el atributo **media**, implica que hay una versión diseñada para un medio diferente. Cuando se usa con **type = application** se trata de un feed.

  · **start** = se refiere al primer documento de un conjunto de documentos.

  · **next** = se refiere al siguiente documento en una serie ordenada de documentos.

  · **prev** = se refiere al documento anterior en una serie ordenada de documentos.

  · **contents** = se refiere a un documento que sirve como tabla de contenidos.

  · **index** = se refiere a un documento que es un índice para el documento actual.

  · **glossary** = se refiere a un documento que proporciona un glosario de términos.

  · **copyright** = se refiere al aviso de copyright del documento actual.

  · **chapter** = se refiere a un documento que actúa como capítulo en una serie.

- **section** = se refiere a un documento que actúa como sección en una serie.
- **subsection** = se refiere a un documento que actúa como subsección en una serie.
- **appendix** = se refiere a un documento que actúa como apéndice en una serie.
- **help** = se refiere a un documento que ofrece ayuda.
- **bookmark** = se refiere a una señal de lectura, es decir, un vínculo a una entrada importante dentro de un documento extenso.

Otros valores no predefinidos para usar con **rel**:

- **shortcut icon** = sirve para incluir un logo en la barra/pestaña del navegador.
- **canonical** = indica la página principal en caso de duplicación con alternate.
- **offline** = establece un fichero de tipo CDF (Channel Definition Format).
- **search** = indica una página de material relacionado con el documento actual.
- **service.feed**, **service.post**, **editUri**, etc. para las tecnologías RSS y Atom.

Ejemplos:

Para indicar que hay una versión de la página en otro idioma:

```
<link rel="alternate" hreflang="en" title="Mi página en Inglés"
   type="text/html"  href="http://mipagina.es/ingles.html">
```

Para indicar que hay un archivo rss:

```
<link rel="alternate" type="application/rss+xml"
   title="Resumen de todas las secciones" href="/feed.xml">
```

Para insertar un icono en la barra de título o pestaña del navegador:

```
<link rel="shortcut icon" type="image/x-icon"
   href="http://mipagina.es/imagenes/logo.ico">
```

Para vincular una hoja de estilos con el documento actual:

```
<link rel="stylesheet" type"text/css"
   href="http://mipagina.es/estilos.css">
```

Para especificar la página principal cuando hay duplicación en diferentes idiomas:

```
<link rel="canonical" href="http://mipagina.es/index.html">
```

(En todos los casos se puede especificar una ruta relativa en **href** dependiendo de la estructura de directorios)

Actividad 2.12:

- En tu sitio crea la carpeta "imagenes" (no uses tilde ni espacio para nombres de carpeta).
- Crea un logo relacionado con tu empresa, dale tamaño 80x80 px, llámalo logo.ico y súbelo a la carpeta /imagenes.
- Inserta el enlace tipo "shortcut icon" en tu página.
- Comprueba que aparece el logo junto al título, en la pestaña del navegador.

<object>

Su uso dentro de la cabecera es poco habitual, trataremos este elemento en el siguiente apartado, dentro del cuerpo del documento.

2.4.5. <script>

Definición: inserta un script dentro del documento.

Puede aparecer varias veces, tanto en la cabecera como en el cuerpo y puede ser interno al documento, o bien un fichero externo.

Aparición: las etiquetas de inicio y cierre son obligatorias.

Atributos: **charset, type, src, defer**.

- **charset**: codificación de caracteres del script indicado por **src**.
- **type**: indica el lenguaje de programación en el cual está escrito el script.

  Como ya sabemos, con la etiqueta **<meta>** se puede determinar el lenguaje script por defecto para el documento. Si no se hace así, es conveniente indicar el lenguaje cada vez mediante el atributo type.
- **src**: indica la URI donde está alojado el script externo (del inglés source).
- **defer**: es un booleano, su presencia indica al navegador que el script no va a generar ningún contenido en el documento.

Ejemplos:

- Interno:

```
<script type="text/javascript">
   ...
</script>
```

- Externo:

```
<script type="text/javascript" src="/js/banner.js"></script>
<script type="text/javascript"
        src="https://www.google.com/jsapi"></script>
```

Actividad 2.13:

· Busca un código javascript que permita visualizar fecha y hora del sistema.

· Inserta el código para la fecha en tu página.

· Guarda el código para la hora en un archivo externo hora.js

· Inserta la llamada al archivo externo mediante la etiqueta <script... src>

· Visualiza el resultado de la forma FECHA: ..... HORA: .....

2.4.6. <style>

Definición: permite insertar una hoja de estilo interna en la cabecera del documento.

Aparición: etiquetas de inicio y fin obligatorias.

Atributos: **lang, dir, type, media, title.**

- **type**: indica el tipo de lenguaje de estilos.

  Normalmente será "text/css".

- **media**: indica el tipo de medio al que se dirige la información de estilo.

  Por defecto será "screen". Otros valores: tty, tv, handheld, print, braille, aural, all.

  · handheld = para dispositivos móviles.

  · print = para impresoras.

- **title**: es un texto descriptivo de la información de estilo.

Ejemplo:

```
<style type="text/css">
  body {
      margin-left: 40px;
      margin-top: 40px;
      margin-right: 40px;
  }
</style>
```

Actividad 2.14:

Inserta este código en tu página y observa los cambios producidos.

2.5. Contenido del cuerpo

En HTML la mayoría de elementos del cuerpo se pueden clasificar en:

- **Elementos de bloque.**

  Son elementos que crean estructuras más grandes que pueden contener a los elementos de línea y a otros elementos de bloque. Además siempre comienzan en una línea nueva.

- **Elementos de línea.**

  Estos elementos solo pueden contener datos y a otros elementos de línea.

  No comienzan con línea nueva.

2.5.1. Manejo del texto

<h1 | h2 | h3 | h4 | h5 | h6>

Para empezar vamos a ver los elementos **h1** al **h6** que sirven para establecer encabezados, es decir, el tamaño de los títulos.

Hay seis niveles de encabezados: **h1, h2, h3, h4, h5** y **h6** ordenados estrictamente por el tamaño. Los encabezados son elementos de bloque.

Aparición: las etiquetas de apertura y cierre son obligatorias.

Atributos: **%attrs, align.**

Ejemplo:

```
<body>
  <h1>Encabezado H1</h1><h2>Encabezado H2</h2>
  <h3>Encabezado H3</h3><h4>Encabezado H4</h4>
  <h5>Encabezado H5</h5><h6>Encabezado H6</h6>
</body>
```

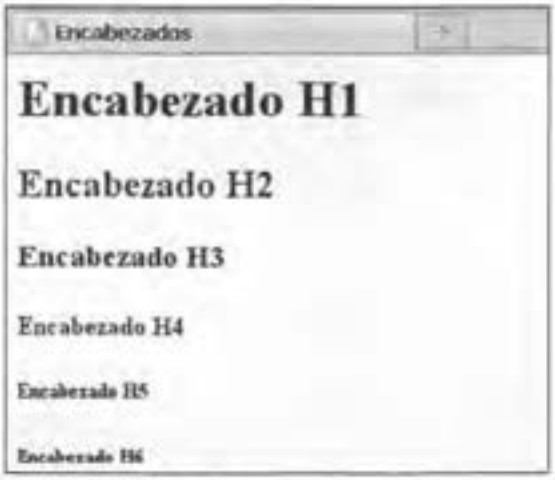


Figura 2.3: Encabezados en HTML

<p>

Definición: Otro de los elementos de bloque más típicos es **<p>** de párrafo.

De la misma forma que distribuimos el texto mediante párrafos en un documento Word, las líneas de texto en un documento HTML deben agruparse dentro de **<p>**. Cada párrafo en HTML se compone de una única línea de texto, salvo que introduzcamos un retorno de carro explícito usando el elemento **
**. El navegador se encarga de que la línea de texto ocupe todo el ancho disponible, y si el ancho cambia, el texto se ajusta al nuevo ancho ocupando más o menos líneas.

Todo texto de un documento HTML, que no sea encabezado, lista o celda de tabla debe estar dentro del elemento **<p>**.

Aparición: la etiqueta de cierre es opcional. No puede contener elementos de bloque.

Atributos: **%attrs, align.**

Ejemplo:

```
<body>
 <h1>Evolución de Networking</h1>
 <h3>Fase1: Ordenadores personales</h3>
 <p>La aparición de ordenadores personales en las empresas fue
    lenta al principio, pero el lanzamiento de programas de
    gestión como Lotus y otras aplicaciones especificas de
    empresa, impulsaron el desarrollo del PC.</p>
 <h3>Fase2: Red a pie</h3>
 <p>Al principio los ordenadores eran dispositivos autónomos a
    los que a veces se conectaban impresoras. Cuando un empleado
    que no tenía impresora conectada a su ordenador, tenía que
    copiar los ficheros a disquete, cargarlos en el ordenador de
    algún compañero que tuviera impresora e imprimirlos desde
    allí. A este tipo de red rudimentaria, se la conoció con el
    nombre de "red a pie".</p>
 <h3>Fase 3: Red LAN</h3>
 <p>Para resolver este inconveniente, las empresas instalaron
    redes de área local (LAN). Esto permitía a los usuarios de
    un mismo departamento transferir rápidamente información a
    través de la red. Las impresoras locales fueron sustituidas
    por impresoras de red.</p>
</body>
```

El resultado se ve en la Figura 2.4.

Un asunto a tener en cuenta es el espacio en blanco, si queremos insertar un espacio más del normal entre dos palabras, no vale utilizar la barra espaciadora porque es ignorada por el navegador. Es necesario insertar el carácter ** ** (no-break space) por cada espacio que queramos añadir.

Evolución de Networking

Fase1: Ordenadores personales

La aparición de ordenadores personales en las empresas fue lenta al principio, pero el lanzamiento de programas de gestión como Lotus y otras aplicaciones específicas de empresa, impulsaron el desarrollo del PC.

Fase2: Red a pie

Al principio los ordenadores eran dispositivos autónomos a los que a veces se conectaban impresoras. Cuando un empleado no tenía impresora conectada a su ordenador, tenía que copiar los ficheros a disquete, cargarlos en el ordenador de algún compañero que tuviera impresora e imprimirlos desde allí. A este tipo de red rudimentaria, se la conoció con el nombre de "red a pie".

Fase 3: Red LAN

Para resolver este inconveniente, las empresas instalaron redes de área local (LAN). Esto permitía a los usuarios de un mismo departamento transferir rápidamente información a través de la red. Las impresoras locales fueron sustituidas por impresoras de red.

Figura 2.4: Texto con la etiqueta <p>

A veces es preciso introducir un salto de línea dentro del mismo párrafo, en ese caso debemos insertar la etiqueta `<br>` de "break" que sirve para "romper" la línea de texto. No sirve insertar un salto de línea usando enter porque también es ignorado por el navegador.

Usando el ejemplo anterior, para evitar que una frase quede partida y mejorar la legibilidad del texto, podríamos introducir saltos de línea detrás de punto y seguido.

Aparición: sin etiqueta de cierre (V).

Evolución de Networking

Fase1: Ordenadores personales

La aparición de ordenadores personales en las empresas fue lenta al principio, pero el lanzamiento de programas de gestión como Lotus y otras aplicaciones específicas de empresa, impulsaron el desarrollo del PC.

Fase2: Red a pie

Al principio los ordenadores eran dispositivos autónomos a los que a veces se conectaban impresoras.
Cuando un empleado no tenía impresora conectada a su ordenador, tenía que copiar los ficheros a disquete, cargarlos en el ordenador de algún compañero que tuviera impresora e imprimirlos desde allí.
A este tipo de red rudimentaria, se la conoció con el nombre de "red a pie".

Fase 3: Red LAN

Para resolver este inconveniente, las empresas instalaron redes de área local (LAN).
Esto permitía a los usuarios de un mismo departamento transferir rápidamente información a través de la red.
Las impresoras locales fueron sustituidas por impresoras de red.

Figura 2.5: Texto con las etiquetas <p> y

<hr>

Otra forma posible de separar el texto es utilizar una línea visible horizontal.

Aparición: sin etiqueta de cierre (V).

Atributos: **%attrs, align, noshade, size y width.**

<b> <i> <big> <small> <s> <strike> <tt> <u>

Definición: son los llamados estilos de fuente.

b: negrita, i: itálica o cursiva, big: tamaño grande, small: tamaño pequeño,

s y strike: texto tachado mediante una línea a media altura,

tt: texto con apariencia de teletipo, u: subrayado con una línea por debajo.

Aparición: las etiquetas de apertura y de cierre son obligatorias.

Atributos: **%attrs.**

Se trata de elementos de línea, por tanto, no hacen salto de línea y se pueden introducir dentro de un párrafo **<p>**. También pueden contener a otros combinándose entre sí.

<font>

Definición: sirve para establecer el tamaño, el color y la fuente para el texto, está desaprobado.

Aparición: apertura y cierre obligatorios.

Atributos: **%i18, %coreattrs, size, color, face.**

Ejemplo:

```
<body>
  <h1>Estilos de Fuente</h1> <hr>
  <ul>
   <li><h2>Estilos simples:</h2>
    <b>Negrita</b><i>Cursiva</i><big>Tamaño Grande</big>
    <small>Tamaño pequeño</small><tt>Teletipo</tt>
    <strike>Tachada</strike><u>Subrayado</u>
   </li>
   <li><h2>Tipos de letra:</h2>
    <font face="Verdana" size="4">Font face Verdana</font>
    <font face="Courier New">Font face Courier</font>
   </li>
  </ul>
</body>
```

Las posibilidades usando hojas de estilos son mucho mayores, por ello, todos los estilos de fuente están en desuso.

<pre> Texto con formato previo

Definición: sirve para que el navegador visualice el texto tal y como aparece en el contenido de esta etiqueta, respetando tabuladores, espacios y saltos de línea.

La única excepción es que introduzcamos en el contenido, otros elementos HTML, los cuales se ejecutarán también.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs y width.`

- `width`: indica el ancho del bloque con formato previo.

Ejemplo:

```
<body>
  <h1>Programación en Java</h1>
  <h2>
   <pre>
   class HolaMundo{
     public static void main (String arg[]){
       System.out.println(<&lt;Hola Mundo&gt;>)
     }
   }
   </pre>
  </h2>
</body>
```

Para evitar que los caracteres "<" y ">" se interpreten como etiqueta debemos utilizar:

`<` = "<" (*less than*)

`>` = ">" (*greater than*)

Programación en Java

```
class HolaMundo{
  public static void main (String arg[]){
    System.out.println(<<Hola Mundo>>)
  }
}
```

Figura 2.6: Texto formateado con <pre>

Si no utilizamos `<pre>` se visualizaría todo el texto en una sola línea.

2.5.2. Listas

Tenemos 2 tipos fundamentales de listas: listas de elementos y listas de definiciones.

A su vez las listas de elementos pueden estar ordenadas o no.

Las listas son un recurso muy común en los textos, como se puede apreciar en este esquema, en el que se han utilizado listas no ordenadas.

- Listas
  - Listas de elementos
    - Listas no ordenadas (`<ul>` `<li>`)
    - Listas ordenadas (`<ol>` `<li>`)
  - Listas de definiciones (`<dl>` `<dt>` `<dd>`)

- **Listas de elementos.**

<ul> Listas de elementos no ordenados.

Definición: se construyen con el elemento `<ul>` y su contenido está formado exclusivamente por elementos `<li>`.

Estos elementos `<li>` son los que contienen a su vez el texto.

Aparición: las etiquetas de apertura y cierre son obligatorias.

Atributos: `%attrs, type y compact.`

- `type:` indica el tipo de símbolo utilizado y puede tomar los valores "disc" (disco), "square" (cuadrado) y "circle" (círculo). Desaprobado.
- `compact:` si aparece indica al navegador que la lista debe compactarse.

<ol> Listas de elementos ordenados.

Definición: se construyen con el elemento `<ol>` y su contenido está formado exclusivamente por elementos `<li>`.

Aparición: las etiquetas de apertura y cierre son obligatorias.

Atributos: `%attrs, type y start.`

- `type:` indica el tipo de secuencia de lista y puede tomar los valores 1 (números decimales), a (letras minúsculas), A (letras mayúsculas), i (números romanos minúsculas), I (números romanos mayúsculas). Desaprobado.
- `start:` indica el número del primer elemento de la lista. Desaprobado.

<li> Elemento de lista.

Definición: objeto de lista, que puede contener otros elementos de bloque como párrafos o directamente el texto.

Aparición: El cierre es opcional.

Atributos: **%attrs y value.**

- **value**: establece el número del objeto de lista actual. Desaprobado.

Ejemplo:

```
<body>
  <h1>Lenguajes de Marcas</h1> <hr>
  <h2>Lista ordenada</h2>
  <ol>
   <li>HTML</li>
   <li>XHTML</li>
   <li>XML</li>
  </ol>
  <h2>Lista no ordenada</h2>
  <ul>
   <li>HTML</li>
   <li>XHTML</li>
   <li>XML</li>
  </ul>
</body>
```

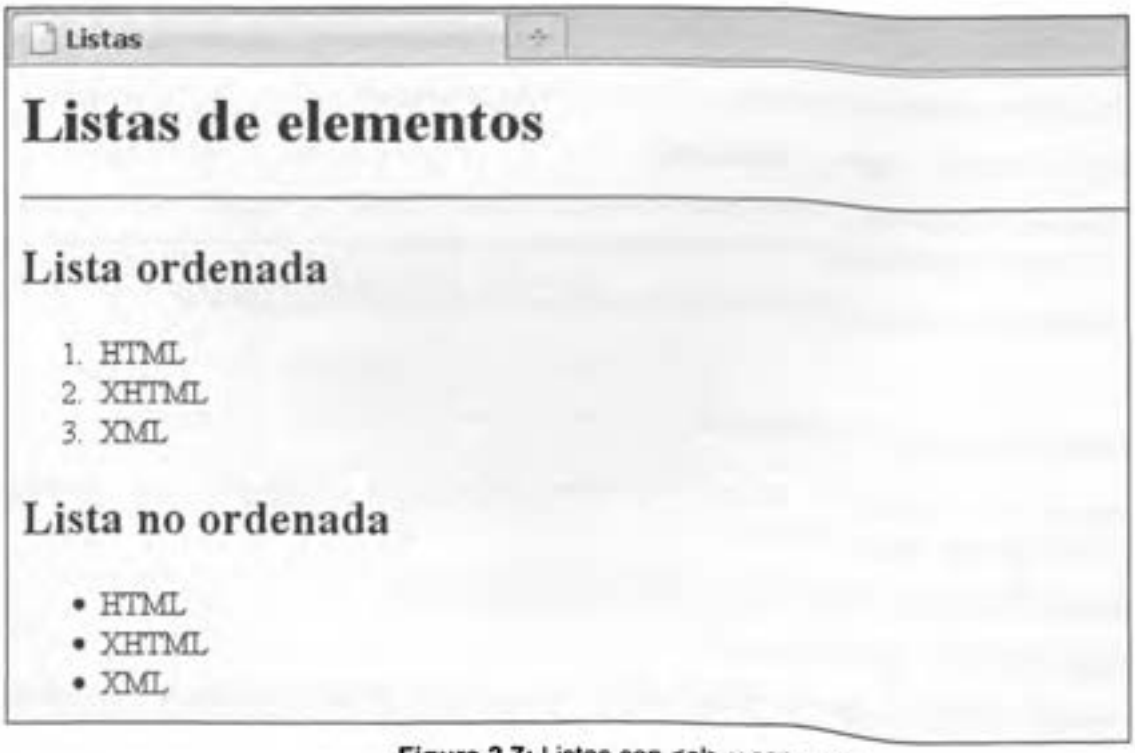


Figura 2.7: Listas con <ol> y con <ul>

- **Listas anidadas.**

La mayor utilidad de las listas resulta cuando se combinan, es decir, cuando cada elemento de una lista es a su vez otra lista. En estos casos los niveles de anidación deben ser de tipos distintos, para evitar la confusión del lector.

Ejemplo:

```
<body>
 <h1>Listas Anidadas</h1> <hr>
 <h2>Lenguajes de Marcas</h2>
 <ol>
  <li><b>Páginas Web</b> <ul><li>HTML</li> <li>XHTML</li> </ul> </li>
  <li><b>Dispositivos móviles</b>
          <ul> <li>WML</li> <li>XHTML MP</li> </ul> </li>
  <li><b>Sindicación</b> <ul> <li>RSS</li> <li>Atom</li> </ul></li>
  <li><b>Documentos</b>
          <ul> <li>RTF</li> <li>TeX</li> <li>MathML</li> </ul> </li>
  <li><b>Gráficos</b> <ul> <li>SVG</li> <li>STEP</li> </ul></li>
  <li><b>Específicos</b> <ul> <li>SMIL</li> <li>WSDL</li> </ul></li>
 </ol>
</body>
```

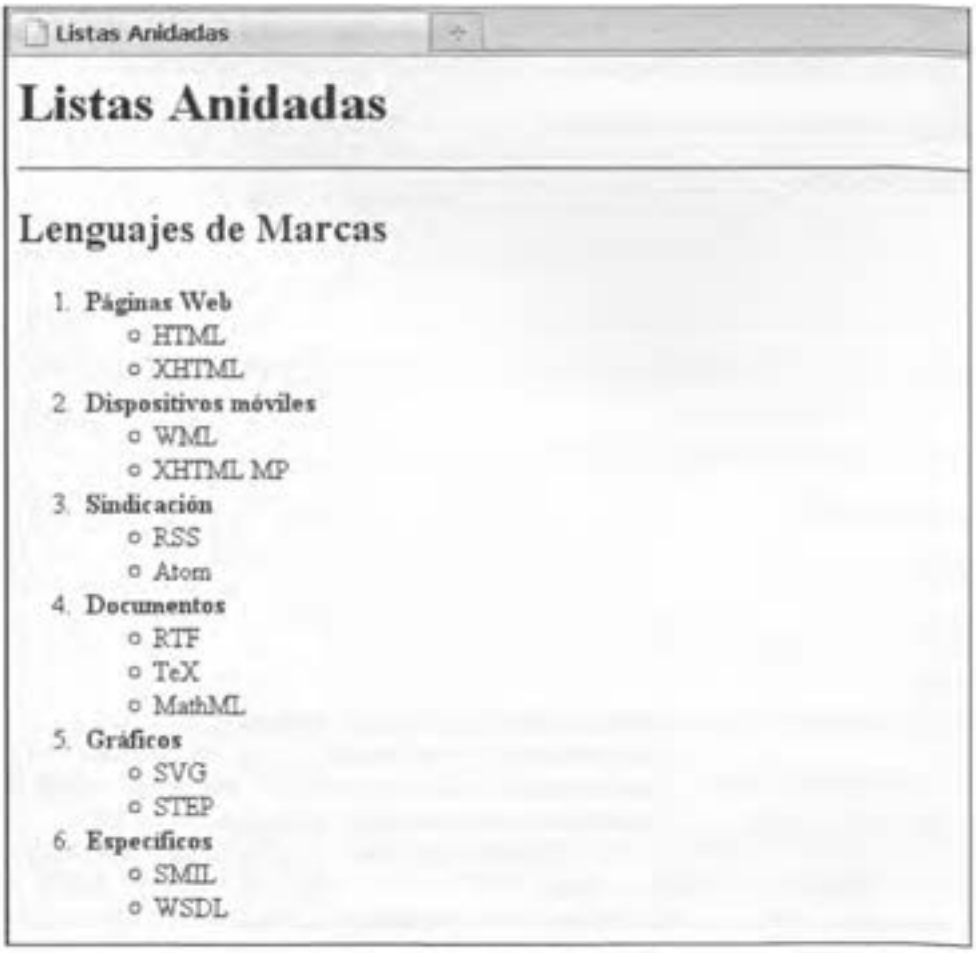


Figura 2.8: Listas anidadas

Actividad 2.15:

Construye una lista anidada, que refleje los módulos que estás estudiando ordenados numéricamente y los contenidos de cada módulo sin ordenar.

- **Listas de Definiciones.**

<dl>

Se trata de listas que no tienen índices ni símbolos de ninguna clase como las anteriores, sino que se componen de dos elementos: el término y la definición, a modo de diccionario. Todo el texto aparece con sangría.

Definición: se construyen con el elemento `<dl>` y contienen solamente elementos de tipo `<dt>` y `<dd>` en número variable.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: `%attrs.`

<dt>

Definición: es el término que se quiere definir, normalmente una palabra o un texto breve y se trata de un elemento de línea.

Aparición: etiqueta de cierre opcional.

Atributos: `%attrs.`

<dd>

Definición: es la definición del término anterior, normalmente un texto largo y se trata de un elemento de bloque.

Aparición: etiqueta de cierre opcional.

Atributos: `%attrs`

Ejemplo:

```
<body>
  <h1>Lenguajes de Marcas</h1>
  <dl>
   <dt>HTML</dt> <dd>HyperText Markup Language </dd>
                 <dd>Lenguaje de Marcas de Hipertexto</dd>
   <dt>XHTML</dt> <dd>Extensible HyperText Markup Languaje </dd>
   <dt>WML</dt>   <dd>Wireless Markup Language </dd>
   <dt>XHTML MP</dt> <dd>Extensible HyperText Markup Language
       Mobile Policy</dd>
   <dt>RSS</dt>  <dd>Rich Site Summary</dd>
                 <dd>Really Simple Sindication</dd>
```

```
    <dt>RTF</dt> <dd>Rich Text Format</dd>
    <dt>SVG</dt> <dd>Scalable Vector Graphics </dd>
    <dt>SMIL</dt> <dd>Sichronized Multimedia Integration Language
                  </dd>
    <dt>WSDL</dt> <dd>Web Services Description Language </dd>
   </dl>
</body>
```

Listas de Definiciones

Lenguajes de Marcas

HTML
    HyperText Markup Language
    Lenguaje de Marcas de Hipertexto
XHTML
    Extensible HyperText Markup Languaje
WML
    Wireless Markup Language
XHTML MP
    Extensible HyperText Markup Language Mobile Profile
RSS
    Rich Site Summary
    Really Simple Sindication
RTF
    Rich Text Format
SVG
    Scalable Vector Graphics
SMIL
    Sichronized Multimedia Integration Language
WSDL
    Web Services Description Language

Figura 2.9: Listas de definiciones

Actividad 2.16:

Construye un mini diccionario de inglés-español mediante una lista de definiciones, con los términos: ace, dbms, geek, ide, nerd, nerf, owned, p2p, raid, uri. En caso de tener varios significados incluye el contexto al que pertenece cada uno.

2.5.3. Enlaces

Los enlaces o hipervínculos son los elementos más característicos del HTML, ya que fueron el motivo que inspiró su desarrollo, para vincular información procedente de muchas fuentes distintas. Un hipervínculo es en realidad una zona del documento, que al realizar alguna acción sobre ella (como hacer clic o colocar el ratón encima) nos redirecciona a otro punto del mismo documento o de otro distinto.

El hipervínculo en principio es invisible, lo que vemos son los elementos que insertamos en su contenido, como texto, imagen, objetos, etc.

<a>

Definición: este elemento sirve para establecer el origen o el destino del hipervínculo.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **%attrs, charset, type, name, href, hreflang, rel, rev, acceskey, shape, coords, tabindex, onfocus, onblur.**

- **name**: sirve para nombrar un ancla o marcador que será el destino del enlace.
- **href**: sirve para especificar la URI destino del hipervínculo.
- **acceskey**: asigna una tecla única de acceso al hipervínculo.
- **shape**: establece la forma de la imagen que contiene el hipervínculo.
- **coords**: establece la posición en pantalla de la imagen.
- **target**: establece el marco de destino (por defecto se toma la ventana actual).

  Valores posibles: _blank, _self, _parent, _top.
- **tabindex**: establece un número de orden.

Se trata de un elemento en línea por lo que puede estar dentro de un párrafo, lista, etc., a su vez **<a>** puede contener otros elementos en línea excepto a sí mismo. Los atributos **name** y **href** son opuestos en el sentido de que, o bien creamos el origen con **<a href>**, o bien el destino con **<a name>**.

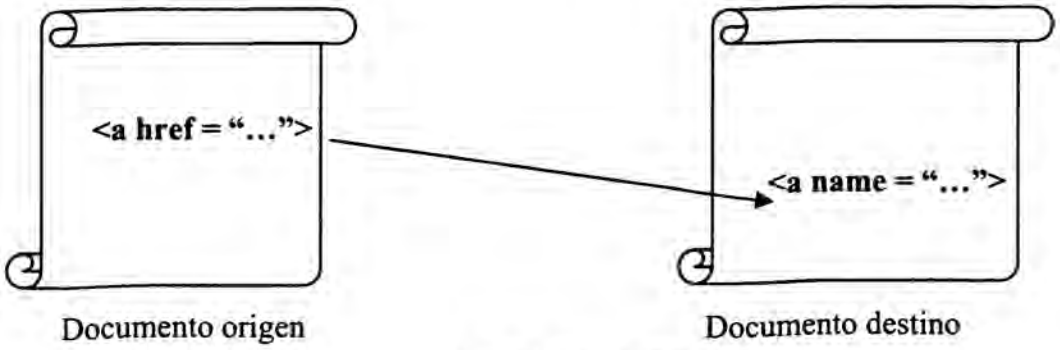


Figura 2.10: Hipervínculo

Vamos a ver cómo se utilizan los hipervínculos, mediante 3 situaciones posibles: destino en la misma página, destino en otra página del mismo servidor y destino en distinto servidor.

- Destino en la misma página.

  Insertamos el marcador en el destino: `<a name="X"></a>`

  Insertar el hipervínculo en el origen: `<a href="#X">`

  Es preciso usar el carácter almohadilla "#" para indicar, que el destino es un marcador de posición interno al documento.

  Ejemplo:

```
<body>
  <h1>Lenguajes de marcas </h1>
  <ul>
    <li> <a href="#H"> HTML </a>  </li>
    <li> <a href="#X"> XHTML </a>  </li>
  </ul>

  <a name="H"></a>
  <h2>HTML</h2>
  <p>HTML (HyperText Markup Language)
    Aparece a principios de los años 90.
    El objetivo principal era enlazar los documentos mediante
    enlaces de forma que al hacer clic sobre algún texto, se
    abriera otro documento relacionado con la información
    seleccionada.
  </p>
  <a name="X"></a>
  <h2>XHTML</h2>
  <p>XHTML (Extensible HyperText Markup Language)
    Es un dialecto de XML que contiene todos los elementos de
    HTML pero que se ajusta a las normas sintácticas de XML.
  </p>
...
</body>
```

También es posible utilizar como marcador de posición, un elemento ya existente en la página, en lugar de crear un nuevo marcador. En este ejemplo usamos el atributo `id`.

Por ejemplo: `<h2 id="X">  XHTML  </h2>`

El resultado sería el mismo que con el marcador `<a name>`.

Actividad 2.17:

Inserta un enlace a la lista de módulos y otro a la lista del diccionario, creadas anteriormente.

- Destino en otra página del mismo servidor.

Supongamos ahora que tenemos una página indice.html que es el origen de todos los hipervínculos. También tenemos una segunda página llamada definiciones.html que contiene los marcadores de posición, que son el destino de los hipervínculos.

El marcador estará en la página destino: `<a name="X"></a>`

El hipervínculo estará en el origen: `<a href="Definiciones.html#X">`

Simplemente hay que añadir el nombre de la página destino en el atributo **href**.

```
<ul>
  <li> <a href="Definiciones.html#H"> HTML </a>  </li>
  <li> <a href="Definiciones.html#X"> XHTML </a>  </li>
  <li> <a href="Definiciones.html#W"> WML </a>  </li>
</ul>
```

Indice.html

```
<a name="X"></a>
<h2>XHTML</h2>
<p>XHTML (Extensible HyperText Markup Language)
   Es un lenguaje que contiene todos los elementos de HTML
   pero que se ajusta a las normas sintácticas de XML.
</p>
```

Definiciones.html

Actividad 2.18:

Crea una segunda página .html para tu empresa, e inserta un enlace en index.html que lleve al comienzo de la segunda página y otro al final.

- Destino en otro servidor (hipervínculo externo)

En este caso no necesitamos marcadores sino especificar la URI completa de la página destino.

Ejemplo:

```
<ul>
 <li> <a href="http://www.w3.org/TR/html401/"> HTML</a> </li>
 <li> <a href="http://www.w3.org/TR/xhtml1/"> XHTML</a> </li>
 <li> <a href="http://www.wapforum.org"> WML</a> </li>
</ul>
```

Estructura de una URI (Uniform Resource Identifier) completa:

Protocolo	:	Parte Jerárquica	?	Solicitud	#	Fragmento

· **Protocolo**: hay muchos, como file, ftp, http, https, imap, ldap, mailto, news, snmp, telnet, xmpp, etc.

· **Parte Jerárquica**: dominio, subdominio, carpeta, subcarpeta, recurso, puerto y las barras (/) para el nivel de jerarquía. (Si el puerto es 80 se omite)

· **Solicitud**: variables $_GET que se pasan al recurso. (Suelen utilizarse para páginas web dinámicas con PHP o similares)

· **Fragmento**: posición del recurso dentro del documento.

Características de una URI:

· La solicitud y el fragmento no son obligatorios.

· El protocolo y el (sub.) dominio de la parte jerárquica son insensibles a mayúsculas y por lo tanto el navegador las convierte en minúsculas.

· La ruta de la parte jerárquica, la solicitud y el fragmento sí son sensibles a mayúsculas.

Esta sensibilidad depende también del servidor que esté procesando la dirección, ya que puede estar configurado para normalizar todo a minúsculas. Sin embargo, la mayoría de los servidores como Apache, distinguen las mayúsculas en esta parte.

· Una URL (Uniform Resource Locator) consta de las tres primeras partes, protocolo, parte jerárquica y solicitud, pero no tiene fragmento.

Ejemplos:

ftp://ftp.rediris.es

http://www.w3c.org

http://www.garceta.es/catalogo/area.php?ID_CO=CF&ID_AR=INFCF

http://www.w3.org/TR/REC-html40/struct/links.html#h-12.1.1

http://www.example.com/ruta/?pagina=2#final

Actividad 2.19:

Estructura de una URI (Uniform Resource Identifier) completa:

Protocolo	:	Parte Jerárquica	?	Solicitud	#	Fragmento

· **Protocolo**: hay muchos, como file, ftp, http, https, imap, ldap, mailto, news, snmp, telnet, xmpp, etc.

· **Parte Jerárquica**: dominio, subdominio, carpeta, subcarpeta, recurso, puerto y las barras (/) para el nivel de jerarquía. (Si el puerto es 80 se omite)

· **Solicitud**: variables $_GET que se pasan al recurso. (Suelen utilizarse para páginas web dinámicas con PHP o similares)

· **Fragmento**: posición del recurso dentro del documento.

Características de una URI:

· La solicitud y el fragmento no son obligatorios.

· El protocolo y el (sub.) dominio de la parte jerárquica son insensibles a mayúsculas y por lo tanto el navegador las convierte en minúsculas.

· La ruta de la parte jerárquica, la solicitud y el fragmento sí son sensibles a mayúsculas.

Esta sensibilidad depende también del servidor que esté procesando la dirección, ya que puede estar configurado para normalizar todo a minúsculas. Sin embargo, la mayoría de los servidores como Apache, distinguen las mayúsculas en esta parte.

· Una URL (Uniform Resource Locator) consta de las tres primeras partes, protocolo, parte jerárquica y solicitud, pero no tiene fragmento.

Ejemplos:

ftp://ftp.rediris.es

http://www.w3c.org

http://www.garceta.es/catalogo/area.php?ID_CO=CF&ID_AR=INFCF

http://www.w3.org/TR/REC-html40/struct/links.html#h-12.1.1

http://www.example.com/ruta/?pagina=2#final

Actividad 2.19:

Inserta en index.html dos hipervínculos externos relacionados con tu empresa.

2.5.4. Agrupación del contenido

Hasta ahora, hemos visto elementos que sirven para agregar contenido o formato al documento, en este apartado vamos a introducir dos elementos, cuyo fin es darle una estructura a ese contenido. Se trata de los elementos ** ** (división) y ****.

Definición: sirve para agrupar varios elementos dentro de un bloque y de esta forma asignarle nombre, clase y estilo.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **%attrs y align.**

Se trata de un elemento de bloque, por tanto, puede contener elementos en línea y otros elementos de bloque, incluyendo otro elemento ** **.

Ejemplo:

```
<html>
  <body>
            <!-- Primer bloque llamado Header  -->

    <div id="Header" style="background-color: #D5EDB3"
     align="center">
      <h2>LATON MAN</h2>
      <p>Industrias StarFuerte</p>
    </div>

            <!-- Segundo bloque llamado Content -->

    <div id="Content" style="background-color: #EE3" align="left">
      <p>Industrias StarFuerte presenta su nuevo prototipo de
         armadura para combate cuerpo a cuerpo contra múltiples
         enemigos. <br>
         Características:
          <ul>
            <li>Resistencia al fuego</li>
            <li>Capacidad para volar</li>
            <li>Gran arsenal de proyectiles</li>
          <ul>
      </p>
    </div>

    <p>Solo se puede adquirir por teléfono. <br>
       De regalo un set de cuchillos. <br>
       Aproveche esta magnífica oferta.
    </p>
  </body>
</html>
```

La paleta de colores RGB consta, básicamente, de tres colores primarios aditivos: Rojo-Verde-Azul, o bien **Red-Green-Blue**.

Estos colores primarios en HTML, están representados por tres pares hexadecimales HH-HH-HH según el formato **#RRGGBB.**

Por tanto, los códigos de los colores básicos son:

· ROJO = #FF0000

· VERDE = #00FF00

· AZUL = #0000FF

El resto de colores se obtienen por combinación de los primarios.

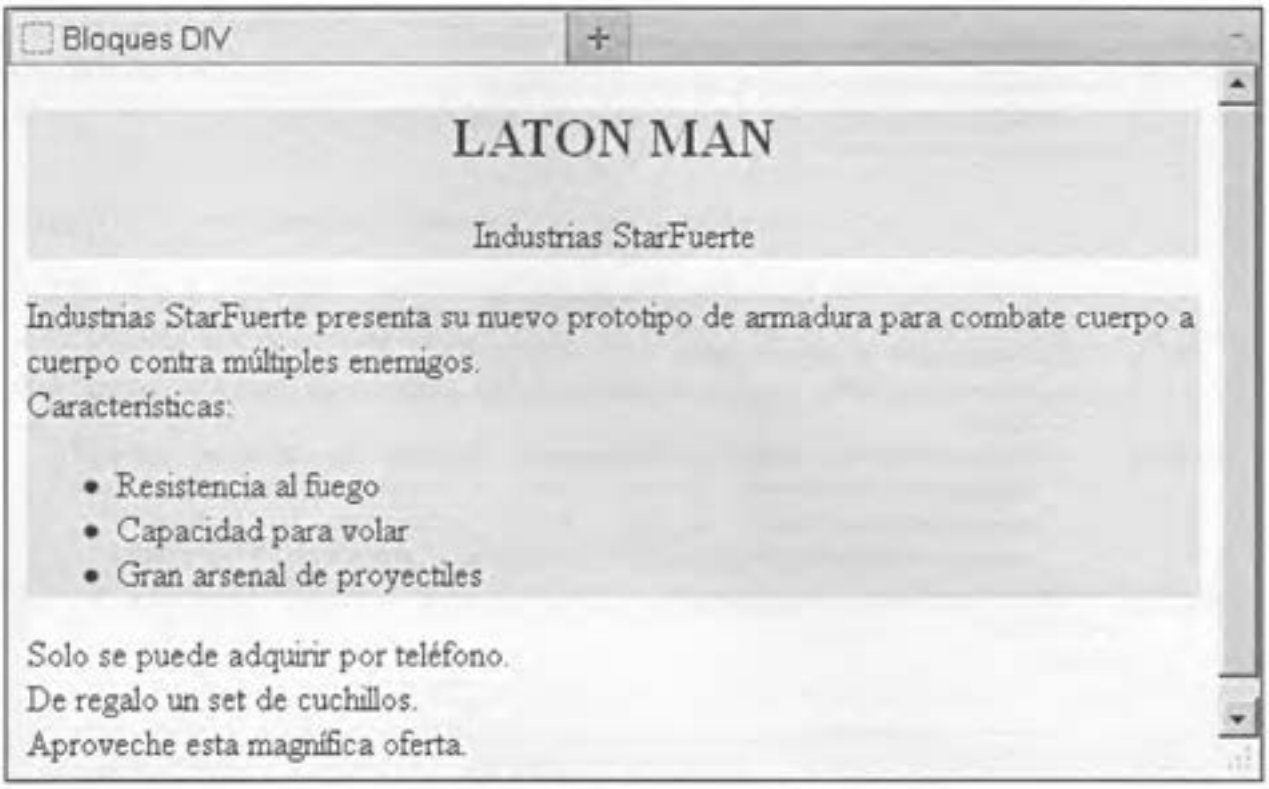


Figura 2.11: Bloques o capas

Cada bloque creado con **`<div>`** define un área dentro del documento, cuyas medidas dependen principalmente del contenido que hayamos insertado en el bloque.

Muy habitualmente a estas áreas del documento se les llama capas, porque pueden ocultarse y solaparse entre sí.

Algunas propiedades de estas capas son: **`left`**, **`top`** y **`position`** para la posición en pantalla, **`width`** y **`height`** para ancho y alto, **`z-index`** para ordenar el solapamiento, **`visibility`** para mostrar u ocultar, **`background-color`**, etc.

Pero todas estas propiedades deben establecerse a través de hojas de estilo, por lo que dejamos este tema para más adelante.

Definición: sirve para identificar un elemento en línea y de esta forma asignarle nombre, clase y estilo.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **%attrs y align.**

Con podemos cambiar entre otros, el tamaño de la fuente, el color, el alto de línea, la posición del texto dentro de la línea, etc.

Ejemplo:

```
<body>
            <!-- Primer bloque llamado Header  -->

  <div id="Header" style="background-color: #D5EDB3"
  align="center">
   <h2>Primer Bloque</h2>
   <p>Este párrafo no tiene estilo específico</p>
  </div>

            <!-- Segundo bloque llamado Content -->

  <div id="Content" style="background-color: #EE3"
  align="center">
   <h2>Segundo Bloque</h2>
   <p>Este párrafo tiene elementos en línea con estilo propio:
      <br>
      <span style="font-size: 16pt"> Tamaño de fuente 16 |
      </span>
      <span style="font-style: italic">Cursiva | </span>
      <span style="font-weight: bolder"> Negrita | </span>
   </p>
  </div>
</body>
```

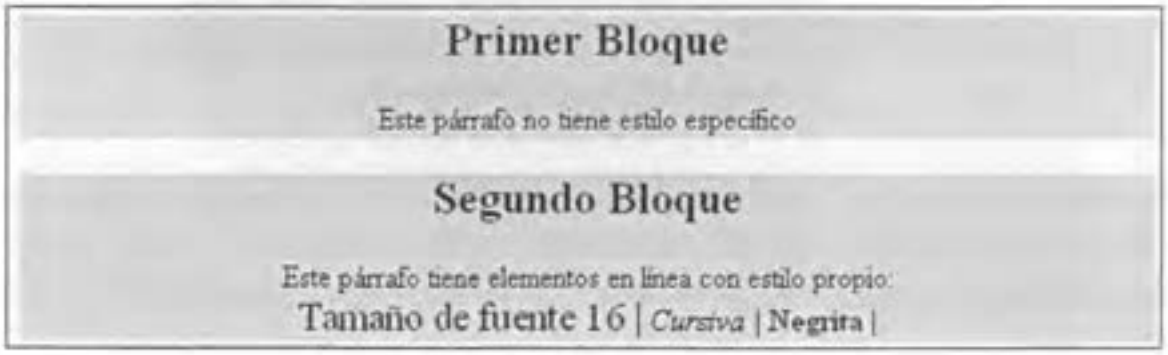


Figura 2.12: Etiqueta

Mediante la etiqueta **** podemos distinguir partes concretas del texto y aplicarle un estilo diferente a cada una, sin necesidad de cambiar de línea.

2.5.5. Imágenes

La mayoría de los navegadores aceptan 3 formatos de imagen: JPG, GIF y PNG.

JPG	GIF	PNG
- Adecuado para fotos	- Adecuado para dibujos	- Adecuado en general
- Compresión con pérdida	- Compresión sin pérdida	- Compresión sin pérdida
- Conserva color original	- Usa el algoritmo LZW	- Millones de colores
- 16 Millones de colores	- 256 colores	- Permite transparencia
- Permite modo progresivo de aparición	- Permite transparencia	- Permite metadatos
	- Permite animación	- Editable
- No permite transparencia	- No degrada el color	- Las fotos pesan más
- No permite animación	- Tiene derechos de autor	- No permite animación

<img>

Descripción: sirve para insertar la imagen determinada por el atributo **src**.

Se trata de un elemento en línea.

Aparición: sin etiqueta de cierre (V).

Atributos: **%attrs, src, alt, longdesc, name, height, width, usemap, ismap. (hspace, vspace, align y border desaprobados).**

- **src**: indica la URI de acceso a la imagen.
- **alt**: texto con breve descripción de la imagen, que se presenta en caso de no poder visualizar la imagen.
- **longdesc**: indica la URI de una descripción larga de la imagen, en caso de tener asociado un mapa de imágenes, debe describir su contenido.
- **name**: identifica la imagen para su manejo posterior en hoja de estilo o script.
- **height, width**: establecen la altura y ancho respectivamente, cuando se especifican el navegador debe modificar las medidas originales de la imagen.
- **usemap**: sirve para asociar a la imagen un mapa de imágenes (como un marcador).
- **ismap**: sirve para definir un mapa de imágenes.

Ejemplo:

```
<body>
 <img src="imagenes/monitor.gif" alt ="monitor" width="113"
     height="110">
</body>
```

<map>

Definición: sirve para crear un mapa de imágenes en el lado del cliente.

Aparición: etiquetas inicial y final obligatorias.

Atributos: **%attrs** y **name**.

<area>

Definición: sirve para especificar una región geométrica del mapa y el vínculo asociado a esa región.

Aparición: sin etiqueta final (V).

Atributos: **%attrs**, **shape**, **coords**, **href**, **nohref**, **alt**, **tabindex**, **accesskey**, **onfocus** y **onblur**.

- **shape**: especifica el tipo de región con los valores: default, rect, circle y poly.
  - · default (región completa)
  - · rect (rectangular) = (x,y) vértice superior izquierdo, (x,y) vértice inferior derecho.
  - · circle (circular) = (x, y) centro, radio.
  - · poly (poligonal) = (x,y) 1° punto, (x,y) 2° punto, ..., (x,y) último punto.
- **coords**: indica la posición en pantalla, son valores de longitud partiendo de la esquina superior izquierda de la pantalla y separados por comas.

Ejemplo:

```
<body>
  <img src="mapa.png" usemap="#mapa1" alt="mapa 4 colores">
  <map name="mapa1">
    <area href="amarillo.html" alt="AMARILLO" shape="rect"
          coords="0,0,190,190">

    <area href="verde.html" alt="VERDE" shape="rect"
          coords="210,0,400,190">

    <area href="azul.html" alt="AZUL" shape="rect"
          coords="0,210,190,400">

    <area href="rojo.html" alt="ROJO" shape="rect"
          coords="210,210,400,400">
  </map>
</body>
```

Observa que el marcador "#mapa1" usado en el atributo **usemap** de la imagen, debe ser igual que el usado en el atributo **name** del mapa.

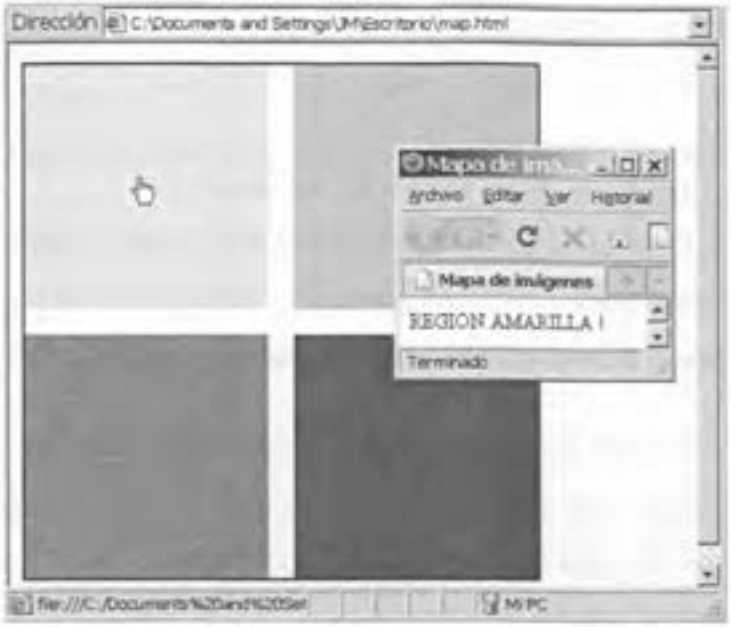


Figura 2.13: Mapa de imágenes con shape="rect"

Cuando pinchamos sobre el área de color amarillo se abre el archivo amarillo.html que contiene el texto "REGION AMARILLA!".

Uno de los usos más extendidos para los mapas de imágenes es la localización de áreas geográficas, en este caso se trata de polígonos con muchos vértices.

Figura 2.14: Mapa de imágenes con shape="poly"

```
<area href="#" shape="poly" alt="Cáceres"
 coords="64,120,65,117,73,118,81,118,88,117,92,119,94,124,99,129,101,
 135,104,136,103,139,103,142,104,146,105,149,99,152,95,156,88,156,81,
 158,76,152,69,152,60,148,56,147,50,150,46,143,52,142,55,138,57,131,
 59, 124">
```

2.5.6. Tablas

<table>

Definición: inserta una tabla de tamaño variable que puede contener filas, columnas, fila/s de encabezado, fila/s de pie, fila/s de cuerpo y resumen.

A su vez cada fila y columna puede estar formada por cero o más celdas.

Aparición: etiquetas inicial y final obligatorias.

Atributos: **`%attrs, summary, width, border, frame, rules, cellspacing, cellpadding.`**

- **`summary:`** resumen del contenido y estructura de la tabla (para los buscadores).
- **`width:`** anchura total de la tabla en pixeles (px) o porcentaje (%).
- **`frame`**: especifica qué lados de la tabla serán visibles.
  - · void (ninguna, valor por defecto)
  - · above, below, lhs, rhs (solo arriba, solo abajo, solo izquierda, solo derecha)
  - · box (todos los lados)
- **`rules`**: especifica qué líneas de división entre celdas serán visibles.
  - · none (ninguna, valor por defecto)
  - · groups (solo entre grupos de filas o grupos de columnas)
  - · rows, cols (solo entre filas, solo entre columnas)
  - · all (todas)
- **`border`**: especifica el ancho del borde exterior de la tabla en px.
- **`cellspacing`**: especifica el espacio entre celdas.
- **`cellpadding`**: especifica el espacio entre el borde de una celda y su contenido.

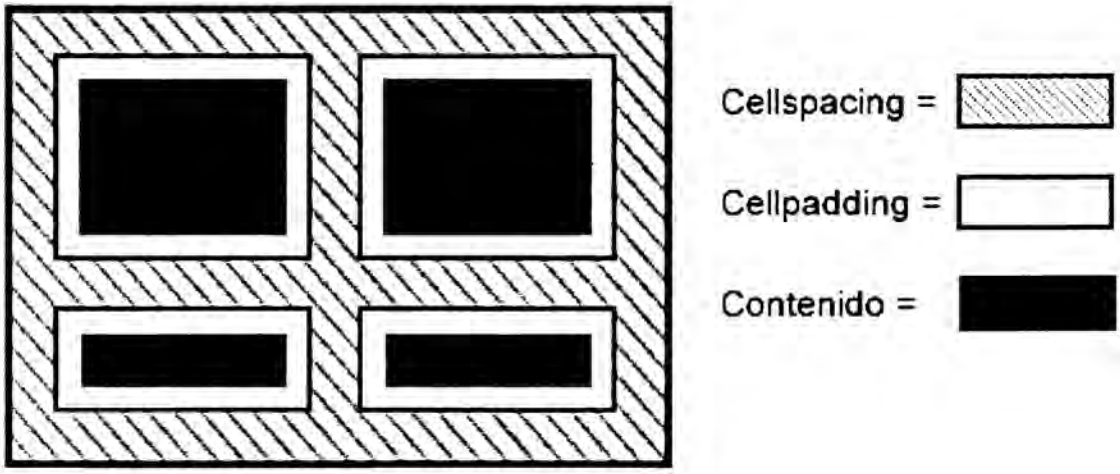


Figura 2.15: Border, cellspacing, cellpadding y contenido.

Contenido de table: **<tr>**, **<td>**, **<th>**, **<thead>**, **<tfoot>**, **<tbody>**, **<colgroup>**, **<col>** y **<caption>**.

<tr>

Definición: sirve para crear una fila dentro de una tabla.

Aparición: etiqueta final opcional.

Atributos: **%attrs**, **align**, **valign**, **char**, **charoff** y (bgcolor desaprobado).

<td> <th>

Definición: sirve para crear una celda dentro de una fila (**<th>** celda de cabecera).

Aparición: etiqueta final opcional.

Atributos: **%attrs**, **abbr**, **axis**, **headers**, **scope**, **rowspan**, **colspan**, **align**, **valign**, **char**, **charoff**, (bgcolor, nowrap, width, height desaprobados)

- **align**: indica el alineamiento del texto (left, right, center, justify, char).
- **valign**: indica la posición vertical de los datos (top, middle, bottom, baseline).
- **char**: indica que un carácter concreto del texto actúa de eje de alineación.
- **charoff**: indica la distancia entre el borde y el carácter de alineación.
- **abbr**: indica de forma breve el contenido de una celda.
- **axis**: indica la categoría a la que pertenece la celda.
- **headers**: indica mediante una lista, las celdas que son de encabezado.
- **scope**: indica las celdas de encabezado mediante row, col, rowgroup y colgroup.
- **rowspan**: indica el número de filas que ocupa la celda (combinar filas).
- **colspan**: indica el número de columnas que ocupa la celda (combinar columnas).

<thead>, <tfoot>, <tbody>

Definición: sirven para agrupar filas en la cabecera, el pie o en el cuerpo de una tabla.

Cada grupo de filas debe tener al menos una fila definida por **<tr>**.

Son útiles para tablas muy largas que ocupan varias páginas, el navegador puede repetir en cada página el encabezado y el pie para facilitar su lectura.

Aparición: etiqueta final opcional.

Atributos: Tienen los mismos atributos que **<tr>**.

<colgroup>

Definición: sirve para crear un grupo de columnas.

Aparición: etiqueta final opcional.

Atributos: **%attrs, span, width, align, valign, char y charoff.**

- **span**: determina el número de columnas del grupo (por defecto 1).
- **width**: determina una anchura por defecto para cada columna del grupo.

<col>

Definición: sirve para agrupar atributos.

Aparición: sin etiqueta final (V).

Atributos: **%attrs, span, width, align, valign, char y charoff.**

Ejemplo:

```
<table border="3" cellspacing="0" cellpadding="0" width="230">
  <tr> <td colspan="3" height="10"></td> </tr>
  <tr>
    <td> </td>
    <td width="10" height="44"> </td>
    <td width="210" align="center">
      <p>
        <span><strong>Mens Sana in Corpore Sano</strong></span>
      </p>
    </td>
    <td width="10"> </td>
  </tr>
  <tr> <td colspan="3" height="10"></td> </tr>
</table>
```

Figura 2.16: Columnas combinadas mediante "colspan"

Actividad 2.20:

Construye una tabla 2x2. Asigna cellpadding=10px, cellspacing =20px, borde de tabla, color de fondo para la tabla, color para las celdas y rellena las celdas con texto.

Ejemplo:

```
<table border="3" cellspacing="0" cellpadding="5" width="390">
   <tr>
     <th width="10">Cod</th><th width="80">Nombre</th>
     <th width="300">Significado</th>
   </tr>
   <tr><td>1</td><td>SGML</td><td>Standard Generalized Markup
                                    Language</td> </tr>
   <tr><td>2</td><td>HTML</td><td>HyperText Markup
                                    Language</td></tr>
   <tr><td>3</td><td>XML</td><td>Extended Markup Language</td>
   </tr>
</table>
```

Cod	Nombre	Significado
1	SGML	Standard Generalized Markup Language
2	HTML	HyperText Markup Language
3	XML	Extended Markup Language

Figura 2.17: La cabecera <th> centra su contenido automáticamente

<caption>

Definición: sirve para poner un título a la tabla, aparece justo encima.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs` (align desaprobado).

Diseño (Layout) a 3 columnas

LOGO		
Menú Izquierdo	Contenido	Menú Derecho
PIE DE PÁGINA		

Figura 2.18: Tabla con <caption>

2.5.7. Marcos

Antiguamente casi todas las páginas contenían marcos, porque son una manera fácil de crear diferentes áreas de navegación.

La separación entre cabecera, pie, menú e información se hacía mediante marcos. Sin embargo, la aparición de las hojas de estilo CSS los ha relegado a segundo plano.

Con las hojas de estilo podemos conseguir diseños más flexibles y variados, además tienen ciertos inconvenientes que los han relegado casi por completo.

Un documento con marcos no tiene sección body, en su lugar tiene una sección **<frameset>**, y una sección **<noframes>** en caso de no poder visualizar los marcos.

<frameset>

Definición: divide la ventana en rectángulos (marcos) independientes y redimensionables.

Aparición: etiquetas inicial y final obligatorias.

Atributos: **%coreattrs, rows, cols, onload, onunload.**

- **rows**: especifica los marcos horizontales mediante una lista de valores separados por comas. Los valores son longitudes expresadas en px o porcentajes.
- **cols**: especifica los marcos verticales de igual forma.

<frame>

Definición: define el contenido y apariencia de un marco.

Aparición: sin etiqueta de cierre (V).

Atributos: **%coreattrs, longdesc, name, src, frameborder, marginwidth, marginheight, noresize, scrolling.**

- **name**: nombre del marco.
- **longdesc**: URI a una descripción larga.
- **src**: contenido inicial del marco (un documento HTML, una imagen, etc).
- **frameborder**: 0 no incluye borde, 1 incluye borde.
- **marginwidth, marginheight**: es el padding del marco.
- **noresize**: es un booleano para impedir que la ventana sea redimensionable.
- **scrolling**: sirve para incluir una barra de desplazamiento (auto | yes | no).

Cada marco es una ventana o página independiente con sus propiedades y con una URI diferente. Esto supone una diferencia apreciable frente a un diseño hecho mediante capas, donde todas las capas pertenecen a la misma página. Los marcos pueden dificultar la navegación porque dejan sin utilidad a los botones de documento previo (back) y documento siguiente (forward), ya que ambos nos trasladarán fuera del documento con marcos.

Ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
      "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
   <title>Marcos</title>
   </head>
            <!-- Definición de marcos -->

   <frameset rows="50, 300, 50">
      <frame src="Cabecera.html">
      <frameset cols="10%, 90%">
         <frame src="Menu.html">
         <frame src="Informacion.html">
      </frameset>
      <frame src="Pie.html">
   </frameset>
</html>
```

Se construye de forma similar a una tabla con 3 filas y 2 columnas en la fila central.

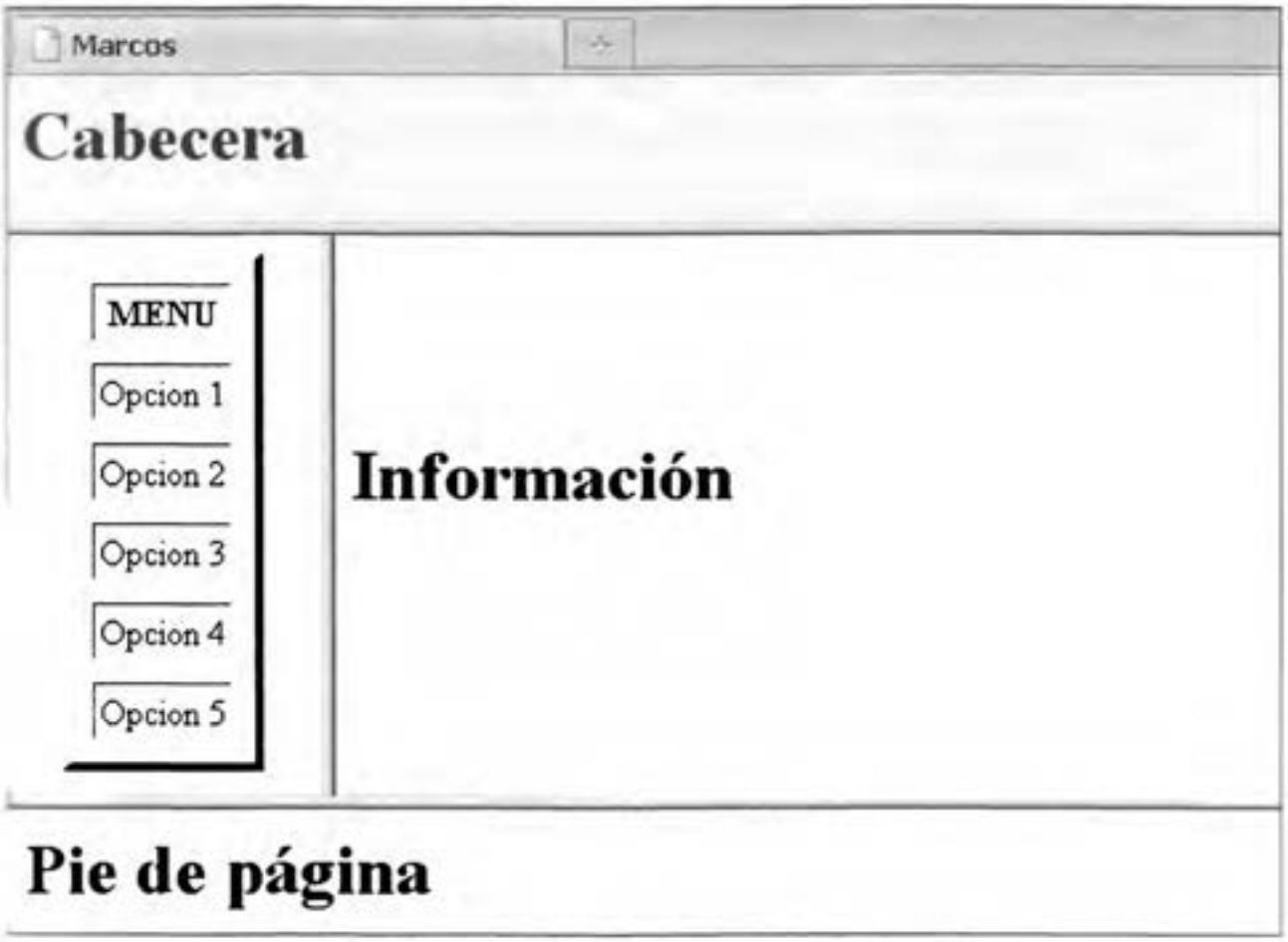


Figura 2.19: Diseño de página a 2 columnas mediante marcos

2.5.8. Objetos Multimedia

<iframe>

Definición: inserta un marco en línea dentro de un documento, es equivalente a insertar una página dentro de otra con el tamaño establecido.

El iFrame es un objeto que se manipula con la misma libertad que cualquier otro elemento dentro del documento. Se suele utilizar para publicidad o sitios de colaboración.

Aparición: etiquetas inicial y final obligatorias.

Atributos: **%coreattrs, longdesc, name, src, frameborder, marginwidth, marginheight, scrolling, width, height** (align desaprobado).

<object>

Definición: inserta un objeto en el documento.

El tipo de objeto viene determinado por sus atributos y puede ser una imagen, un applet de Java, un vídeo, una animación flash, otro documento HTML, etc.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **%attrs, declare, classid, codebase, data, type, codetype, archive, standby, height, width, usemap, name, tabindex.**

- **declare**: es un booleano, cuando aparece indica que solo se trata de una declaración del objeto, por tanto, la instanciación debe producirse después en otra línea con object.
- **classid**: indica la URI donde está la implementación del objeto.
- **codebase**: indica la dirección base para completar los URIs de los demás atributos (por defecto es la misma que el documento).
- **data**: indica la URI donde están los datos del objeto.
- **type**: indica el tipo de datos especificados en data.

  La sintaxis es type="categoría/formato". Ejemplos: audio/basic, audio/mp3. video/mpeg, video/quicktime, application/x-shockwave-flash, etc.
- **codetype**: indica el tipo de datos especificados por classid.
- **archive**: es un la lista de URIs (separada por espacios) donde hay archivos útiles para el objeto.
- **standby**: es un texto que presenta el navegador durante la carga de los datos.
- **height, width**: alto y ancho del objeto.
- **usemap**: indica al objeto que utilice el mapa de imágenes del cliente.
- **name**: nombre para designar el elemento.

<param>

Definición: sirve para inicializar variables de objetos, que serán utilizadas en tiempo de ejecución.

Aparición: sin etiqueta de cierre (V).

Atributos: **id, name, value, valuetype, type.**

- **name**: indica el nombre de un parámetro de ejecución.
- **value**: indica el valor inicial del parámetro especificado en name.
- **valuetype**: indica el tipo de atributo **value** (data | ref | object).
- **type**: indica el tipo del recurso cuando **valuetype** = "ref".

Sintaxis:

```
<param name="nombre de parámetro" value="valor de parámetro">
```

La lista de parámetros depende del tipo de objeto, algunos son: FileName para el nombre del archivo, quality para la calidad de imagen, wmode para el fondo, allowFullScreen para pantalla completa, swfversion para la versión de Flash Player, etc.

Ejemplo 1: Insertar un video local.

```
<object type="video/mpeg" id="hamaca" width="300" height="200"
        data="videos/hamaca.mpg">
  <param name="FileName" value="videos/hamaca.mpg">
  <param name="quality" value="high">
</object>
```

Figura 2.20: Insertar un video local

Actividad 2.21:

Descarga un archivo .swf de Internet, ajusta el ancho y alto, e insértalo en tu página.

Ejemplo 2: Insertar un video de YouTube®.

Se trata de archivos flash, por tanto, utilizamos **type** (application/x-shockwave-flash), en **data** especificamos la URI que aparece en YouTube.

```
<body>
  <h2>Un video de YouTube</h2>
  <object type="application/x-shockwave-flash"
     data="http://www.youtube.com/v/en0EfNXmL6M?version=3"
     width=425 height=350>
  </object>
</body>
```

Figura 2.21: Insertar un video de YouTube

El código que proporciona YouTube:

```
<iframe width="420" height="315"
   src=http://www.youtube.com/embed/en0EfNXmL6M
   frameborder="0" allowfullscreen>
</iframe>
```

Ejemplo 3. Insertar un plugin social.

Vamos a insertar un botón "Me gusta" de facebook®.

En http://developers.facebook.com/

Iniciamos sesión en facebook DEVELOPERS, Social Plugins, Like Button.

En el paso 1, introducimos los datos para configurar el botón a nuestro gusto: la URL de nuestra página, el diseño con o sin contador, ancho, perfiles, color, fuente, etc.

Por último, debemos elegir el tipo de etiquetas con las que se genera el código, para HTML 5, para XFBML o con **<iframe>**. Elegimos la última que es la más simple y la más compatible con versiones anteriores de los navegadores.

El código que nos proporciona la herramienta:

```
<iframe
  src="//www.facebook.com/plugins/like.php?href=http://www.juanm
  acr.es/index.html&amp;send=false&amp;layout=standard&amp;width
  =450&amp;show_faces=false&amp;action=like&amp;colorscheme=ligh
  t&amp;font&amp;height=80"scrolling="no" frameborder="0"
  style="border:none; overflow:hidden; width:450px;
  height:80px;" allowtransparency="true">
</iframe>
```

En el paso 2, vamos a obtener las etiquetas **<meta>** que vinculan nuestra web con facebook, se trata de las **<meta property>** que ya vimos anteriormente. Introducimos los datos asociados a la página (o el objeto en general) como el título, tipo, URL, imagen, etc.

Ejemplo:

```
<meta property="og:url" content="http://www.juanmacr.es" />
<meta property="og:title" content="Sitio Web de ayuda al
      estudiante de ciclos de informática" />
```

Añadimos el espacio de nombres como atributo de html:

```
xmlns:og="http://ogp.me/ns#"
```

El resultado estándar, después de publicarlo, sería:

Figura 2.22: Botón Me gusta

De forma similar podemos agregar el botón +1 de Google.

En http://www.google.com/webmasters/+1/button/

Rellenamos el formulario de opciones para configurar el botón, y debemos elegir de nuevo las etiquetas para generar el código, válido para HTML 5 o como en este ejemplo:

```
<div class="g-plusone" data-annotation="inline"></div>
<script>…</script>
```

El resultado sería:

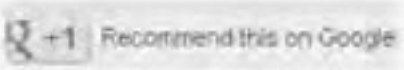


Figura 2.23: Botón +1

Ahora vamos con los botones de Twitter.

En Twitter developers, https://dev.twitter.com/

Elegimos uno de los botones, por ejemplo, Twittear:

Rellenamos las opciones de configuración y copiamos el código que debemos pegar en la página donde aparecerá el botón.

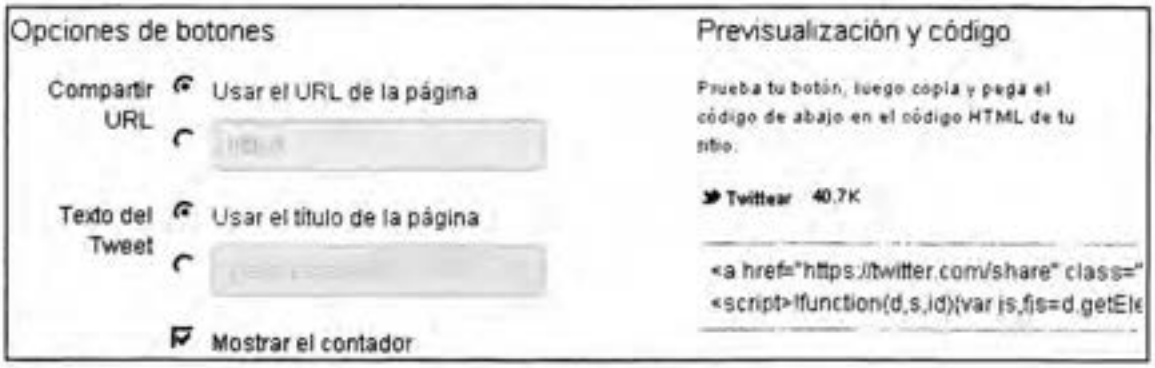


Figura 2.24: Opciones de Botón twittear

```
<a href="https://twitter.com/share" class="twitter-share-button"
   data-lang="es">
     Twittear
</a>
<script> … </script>
```

El resultado sería:

Figura 2.25:. Botón Twittear

2.5.9. Formularios

Inicialmente HTML estaba diseñado para que el usuario recibiera datos del servidor, pero no para que el usuario enviara datos al servidor, con el fin de solucionar esta deficiencia se diseñaron los formularios.

Un formulario es un área del documento en la que insertamos elementos de entrada de información, llamados comúnmente controles de formulario, que permiten introducir datos, marcar opciones, elegir una opción de un grupo, etc.

Es conveniente indicar que el formulario solo sirve para recabar información del usuario, pero el tratamiento posterior de esta información lo hace normalmente un programa externo al documento llamado CGI (Common Gateway Interface).

Estos programas realizan diversas tareas como:

· Verificar y manipular los datos.

· Almacenar los datos en un archivo o en una base de datos.

· Reenviar los datos por correo electrónico.

· Leer los datos y devolver algún resultado al usuario.

Hay varias formas de construir estos programas, pero la más habitual es utilizar un lenguaje de programación para crear un script en el servidor, los lenguajes más utilizados son C, Perl, Java, ASP y sobretodo PHP.

<form>

Definición: sirve para insertar un formulario en el documento.

Aparición: etiquetas inicial y final obligatorias.

Atributos: **%attrs, action, method, enctype, accept, accept-charset, name, target, onsubmit, onreset.**

- **action**: indica la URI del programa encargado de tratar los datos del formulario.

  (Por ejemplo: **action="registro/registroCliente.php"**)

- **method**: indica el método usado para pasar los datos al programa (get | post).

- **enctype**: indica el tipo de contenido usado para enviar los datos.

  (Solo para method="post").

- **accept**: es una lista de tipos de contenido aceptados por el servidor.

  (Sirve para filtrar archivos de tipos no aceptables).

- **accept-charset**: codificación de caracteres usada para los datos.

- **onsubmit**: sirve para ejecutar alguna acción cuando el formulario es enviado.

- **onreset**: sirve para realizar alguna acción cuando el formulario es reseteado.

<Form> es un elemento de bloque que puede contener a su vez a los elementos:

- **<input>**
- **<button>**
- **<select> <option> <optgroup>**
- **<textarea>**

<input>

Definición: sirve para insertar un elemento, definido por 'type', dentro del formulario.

Aparición: sin etiqueta de cierre (V).

Atributos: **%attrs, type, name, value, size, maxlenght, checked, src, alt, disabled, readonly, usemap, ismap, tabindex, acceskey, onfocus, onblur, onselect, onchange, accept** (align desaprobado).

- **type**: indica el tipo de control de formulario.

  (text | password | checkbox | radio | submit | reset | file| hidden | image | button).
- **name**: nombre del control.
- **size**: establece el ancho del control en pixels o en caracteres.
- **maxlenght**: establece el nº máximo de caracteres para control de tipo texto o password.
- **checked**: marca la casilla para un control de tipo checkbox o radio.
- **src**: indica la URI donde se encuentra la imagen para type = "image".
- **disabled**: este booleano deshabilita el control y no se puede introducir datos.
- **readonly**: este booleano impide que el usuario haga cambios en el control.
- **accept**: lista de tipos mime válidos para subir ficheros.
- **value**: · para cuadros de texto es el valor inicial que aparece en el recuadro
  · para botones de tipo radio y checkbox es el valor que se envia al servidor
  · para botones de acción es el título del botón

Ejemplo:

```
<form action="manejador.html" enctype="multipart/form-data"
      method="post">
    USUARIO:
    <input type="text" name="usuario" value="user"> <br>
    CONTRASEÑA: <input type="password" name="pass"> <br>
    SOCIO: <input type="checkbox" name="socio" checked> <br>
    SEXO: <input type="radio" name="genero" value="m"> Masculino
    <input type="radio" name="genero" value="f"> Femenino <br>
```

```
    TEMAS PREFERIDOS:
    <input type="radio" name="aventuras" value="a"> Acción
    <input type="radio" name="historica" value="h"> Histórica
    <input type="radio" name="ficcion" value="fi"> Ficción <br>
    DESCUENTO:
    <input name ="descuento" value="15%" size="5" readonly> <br>
    SUBIR FOTOGRAFIA: <br>
    <input type="file" name="fichero"
           accept="application/x-zip-compressed"> <br>
    <input type="hidden" name="fecha" value=""> <br>
    <input type="submit" value="enviar">
    <input type="reset" value="reset">
    <input type="button" value="salir"
           onclick="window.close();">
</form>
```

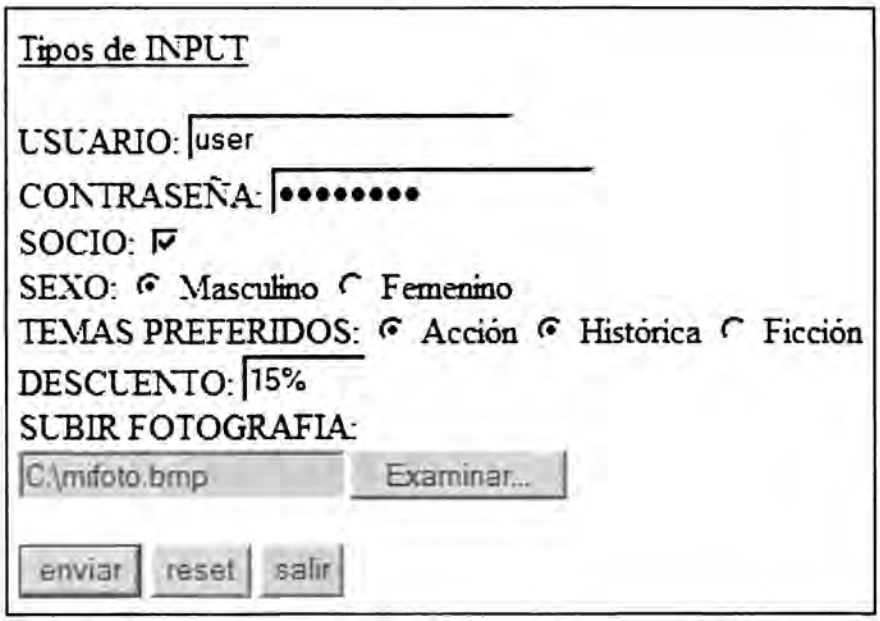


Figura 2.26: Controles de tipo <input>

El control de tipo text tiene el valor por defecto "user".

El control de tipo password sustituye los caracteres que introduce el usuario por símbolos para ocultarlos.

El control de tipo checkbox o casilla de verificación, está marcado por defecto mediante el atributo **checked**, pudiendo ser desmarcado por el usuario.

El primer conjunto botones tipo radio tienen el mismo **name ="genero"**, lo cual los hace mutuamente excluyentes, por tanto, el usuario solo puede marcar uno de ellos.

El segundo conjunto de botones tipo radio tienen diferentes nombres, por tanto, es posible seleccionar varios a la vez.

El control que contiene el descuento tiene el atributo **readonly** que impide al usuario modificar el valor.

El control de tipo file inserta un botón, para realizar una búsqueda en el disco del fichero que se enviará con el formulario. Hay que definir el atributo **enctype**.

El control de tipo hidden no se ve en el formulario, por tanto, no es percibido por el usuario, su finalidad es la de enviar ciertos datos al servidor que no son útiles para el cliente. Por ejemplo, cuando enviamos el formulario anterior podemos enviar la fecha y hora junto con el resto de información, sin que el usuario lo vea expresamente.

Para obtener al fecha y hora del sistema, sería preciso crear un script con lenguaje cliente (como Javascript) y modificar el atributo **value** del control hidden.

Por otro lado, el hecho de que el control sea oculto para el usuario no significa que se envíe de forma segura, todos los datos son enviados sin cifrar salvo que utilicemos un protocolo seguro como HTTPS.

El control de tipo submit envía los datos del formulario a la URI especificada en el atributo **action**.

El control de tipo reset recupera los valores por defecto de todos los controles.

El control de tipo button define un botón genérico, para asociarle una determinada acción con un script, por ejemplo, cerrar la ventana del formulario.

Actividad 2.22:

Crea la página manejador.html con el texto "PEDIDO RECIBIDO" (simulando la respuesta del servidor). Copia el código anterior en una página nueva llamada form.html y comprueba cómo funcionan los controles.

<button>

Definición: sirve para insertar un botón como los creados por **<input>** pero con más posibilidades de formato (por ejemplo, incluir imágenes dentro del botón).

Aparición: etiqueta inicial y final obligatorias.

Atributos: **%attrs, type, name, value, disabled, tabindex, acceskey, onfocus, onblur.**

- **type**: indica el tipo de botón (submit | reset | button).

Ejemplo:

Fig. 2.27: Botón

```
<button type="submit">
  <img src="imagenes/enviar.jpg">
</img></button>
```

Actividad 2.23:

Inserta un botón de enviar con una imagen sonriente en form.html.

<select>

Definición: sirve para insertar un menú de formulario con varias opciones.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs, name, size, multiple, disabled, tabindex, onfocus, onblur, onchange`

- `size`: especifica el nº de filas del menú que se ven al mismo tiempo.
- `multiple`: este booleano permite una selección múltiple de opciones.

Figura 2.28: Control <select>

<option>

Definición: sirve para indicar cada opción de un menú de formulario.

Aparición: etiqueta final opcional.

Atributos: `%attrs, selected, disabled, label, value`.

- `selected`: este booleano indica que la opcion está preseleccionada.
- `label`: especifica un rótulo para la opción de menú.

<optgroup>

Definición: sirve para definir grupos de opciones.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs, disabled, label.`

<textarea>

Definición: sirve para insertar un cuadro de entrada de texto con muchas líneas.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs, name, rows, cols, disabled, readonly, tabindex, acceskey, onfocus, onblur, onselect, onchange`.

- `rows`: indica el nº de líneas de texto visibles.
- `cols`: indica el ancho del cuadro en caracteres.

Comentarios

```
Escriba aquí sus comentarios
```

Figura 2.29: Control <textarea>

<label>

Definición: sirve para asignar un título a un control.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs, for, acceskey, onfocus, onblur`.

- `for`: especifica el nombre de otro control para el que servirá de rótulo. (Coincide con el atributo `id` del control relacionado).

<fieldset>

Definición: sirve para agrupar los controles y sus títulos por temas.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs`.

<legend>

Definición: sirve para asignar un rótulo a un fieldset.

Aparición: etiquetas inicial y final obligatorias.

Atributos: `%attrs, acceskey`.

Ejemplo conjunto: pedido.html

```
<body>
 <h1>INDUSTRIAS STARFUERTE</h1>
 <table width="1000"><tr><td>
 <p>
 <form action="manejador.html" method="post">
  <fieldset>
    <legend>FORMULARIO DE PEDIDO</legend>
    <table width=700 cellpadding=5><tr>
    <tr><td><strong> > Paso 1. Identificación </strong></td></tr>
    <td><fieldset>
         <legend>Datos del Cliente</legend><br>
         <label for="email">Correo Electrónico</label>
         <input type="text" id="email" size=20 maxlength=40>
         <br><label for="pass">Contraseña</label>
         <input type="password" id="pass" size=10 maxlength=30>
         <br>(Longitud mínima 6 caracteres)
        </fieldset>
    </td>
    <td><fieldset>
         <legend>Tipo de Cliente</legend><br>
         <input type="radio" id="particular" name="tipo" checked>
         <label for ="particular">Particular</label><br>
         <input type="radio" id="asociacion" name="tipo">
         <label for ="asociacion">Asociación</label><br>
         <input type="radio" id="empresa" name="tipo">
         <label for ="empresa">Empresa</label>
        </fieldset>
    </td></table>

    <table width=980 cellpadding=5>
    <tr><td><strong> > Paso 2. Pedido <br> </strong></td></tr>
    <tr>
    <td>
        <fieldset>
        <legend>Datos del Producto</legend>
         Producto<br>
         <select id="producto" name="producto" multiple="si" size=8>
          <optgroup label="Armaduras">
          <option value="AR001" selected="selected">
           AR001 Oro Reluciente
          </option>
          <option value="AR002">
           AR002 Plata Pulida
          </option>
          <option value="AR003">
           AR003 Bronce Reforzado
          </option>
          </optgroup>
         <optgroup label="Extras">
          <option value="EX001" selected="selected">
           EX001 Escudo
```

```
            </option>
            <option value="EX002">EX002 Invisibilidad<option>
           </optgroup>
          </select>
          </fieldset>
       </td>
        <td><fieldset>
          <legend>Información de interés</legend>
          Comentarios<br>
          <textarea rows=6 cols=60>Escriba aquí sus comentarios
        </textarea>
          <br><input type="checkbox" id="publi" value="si" checked>
          <label for "publi">Acepto recibir información publicitaria
          </label>
          </fieldset>
        </td>
       </tr>
       </table>

       <table width=980 cellpadding=5>
       <tr>
        <td><fieldset>
            <legend>Datos de Envío</legend><br>
            <label for "dire">Dirección</label>
            <input type="text" id="dire" size=40>
            <br><label for "ciudad">Ciudad</label>
            <input type="text" id="ciudad" size=20>
            <label for "cp">Codigo Postal</label>
            <input type="text" id="cp" size=5>
           </fieldset>
        </td>
        <td><fieldset>
            <legend>Datos Bancarios</legend><br>
            <label for "entidad">Nombre de la Entidad</label>
            <input type="text" id="entidad" size=20><br>
            <label for "cuenta">N° de cuenta</label>
            <input type="text" id="entidad" size=20>
           </fieldset>
        </td>
       </tr>
       <tr>
        <td><button type="submit">ENVIAR<br>PEDIDO</button></td>
        <td><button type="reset">RESET</button></td>
       </tr>
       </table>

     </fieldset>
     </p>
    </form>
    </table>
   </body>
```

INDUSTRIAS STARFUERTE

FORMULARIO DE PEDIDO

> **Paso 1. Identificación**

Datos del Cliente

Correo Electrónico

Contraseña

(Longitud mínima 6 caracteres)

Tipo de Cliente

Particular

Asociación

Empresa

> **Paso 2. Pedido**

Datos del Producto

Producto

Armaduras

AR001 Oro Reluciente

AR002 Plata Pulida

AR003 Bronce Reforzado

Extras

EX001 Escudo

EX002 Invisibilidad

Información de interés

Comentarios

Escriba aquí sus comentarios

Acepto recibir información publicitaria

Datos de Envío

Dirección

Ciudad

Codigo Postal

Datos Bancarios

Nombre de la Entidad

Nº de cuenta

ENVIAR PEDIDO

RESET

Figura 2.30: Formulario completo "pedido.html"

Mediante `<fieldset>` podemos agrupar los controles por grupos, lo que facilita al usuario la tarea de rellenar el formulario.

2.6. XHTML

Como ya sabemos XHTML 1.0 vio la luz por primera vez de forma oficial en 1998 y se convirtió en recomendación para construir páginas web en enero del año 2000.

El objetivo era mejorar las deficiencias que tenía el HTML de aquel momento, incorporando las ventajas que aportaba un nuevo lenguaje recién creado, el XML.

XML es un lenguaje derivado de SGML como todos, pero mucho más sencillo, cuya principal finalidad es la de transmitir datos con una cierta estructura a través de la Web. Los documentos XML no aportan información sobre como van a ser tratados esos datos, ni cómo se van a presentar y esta separación constituye, en sí misma, una gran ventaja para su desarrollo y mantenimiento.

Por otro lado, XML es un lenguaje más estricto en su sintaxis, ya que exige por ejemplo, escribir todos los elementos y sus atributos en minúsculas, los valores de atributos deben ir entre comillas, todo elemento debe tener su marca de cierre, el anidamiento de elementos debe ser correcto sin solapamiento, etc. Esto es una ventaja cuando hay un equipo de desarrolladores trabajando en el mismo sitio Web, así mismo permite que el parser (intérprete) sea más simple.

Por último indicar, que también era imprescindible mantener la compatibilidad con HTML 4, porque estaba muy extendido y los navegadores que no se habían adaptado al nuevo XHTML debían ser capaces de interpretar estos documentos como si se tratase de documentos HTML.

En resumen, XHTML es un lenguaje formado por los mismos elementos que HTML pero que se ajusta a las normas sintácticas de XML.

(Por esta razón, se dice habitualmente que XHTML es una reconstrucción de HTML usando XML).

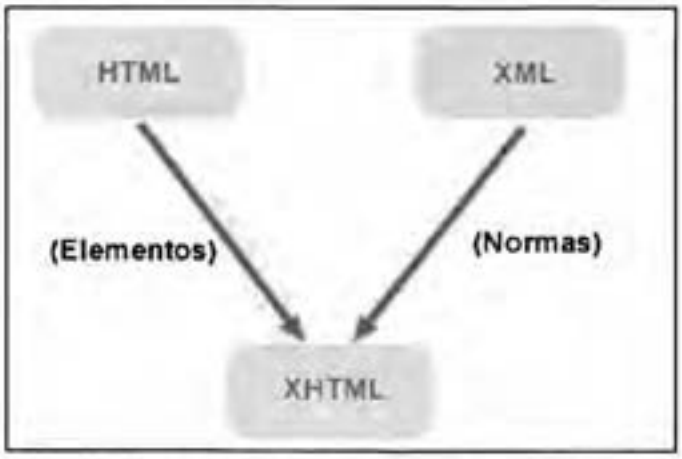


Figura 2.31: Origen del lenguaje XHTML

Otra ventaja de XHTML es la posibilidad de incorporar elementos procedentes de otros espacios de nombres como MathML y SVG. Esto permitirá construir documentos con contenido científico o gráficos vectoriales, de una forma mucho más simple y eficaz.

2.6.1. Estructura y elementos del documento

La estructura es exactamente la misma que la de un documento HTML, por tanto, tiene el conocido aspecto:

```
<! Declaración de DTD >
<html>
  <head>
    Contenido de cabecera
  </head>
  <body>
    Contenido del cuerpo
  </body>
</html
```

La principal diferencia está en la declaración de DTD y en el elemento `<html>`.

Para empezar la declaración de DTD en XHTML es obligatoria mientras que en HTML no lo era, pero seguimos teniendo tres definiciones equivalentes a las de HTML:

Strict, Transitional y Frameset.

DTD Strict
· Si no vamos a utilizar elementos de estilo (considerados obsoletos) dentro del documento, podemos usar la declaración estricta: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
DTD Transitional
· Si queremos permitir el uso de elementos de estilo, utilizaremos la declaración de transición: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
DTD Frameset
· Si queremos usar marcos en el documento, utilizaremos la declaración de frameset: `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`

En cuanto al elemento `<html>` es también obligatorio y tiene dos atributos nuevos: `xmlns` y `xml:lang`.

· `xmlns="http://www.w3.org/1999/xhtml"` se trata del espacio de nombres, que se tratará más adelante en la unidad correspondiente a XML.

El valor que toma este atributo en este contexto es la URI fija indicada.

· `xml:lang="es"` que indica el idioma al estilo xml.

Ejemplo:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
```

Además el elemento `<html>` debe llevar obligatoriamente los elementos `<head>`, `<body>` y a su vez `<head>` debe tener un `<title>`.

2.6.2. Normas en XHTML

Para que un documento XHTML esté bien formado debe cumplir una serie de normas sintácticas heredades de XML:

- Todas las etiquetas y todos los atributos debe ir en minúsculas.

Incorrecto	Correcto

`<Head>`
`<TITLE>Bienvenidos</TITLE>`
`<Head>`
	`<head>`
	`<title> Bienvenidos </title>`
	`</head>`

- Todos los elementos deben cerrarse, incluso los elementos vacíos.
  (Con éstos se puede utilizar la forma abreviada de cierre, que consiste en añadir la barra "/" al final, en lugar de poner la etiqueta de cierre)

Incorrecto	Correcto
` ` `<img src="grafico.gif"` `alt="gráfico">`	` ` `<img src="grafico.gif"` `alt="gráfico" />`

- No se permiten errores de anidamiento, es decir, las etiquetas no deben solaparse.

Incorrecto	Correcto

`<p> Bienvenidos a`
`  <a href="www.garceta.es">`
`</p>`
`  </a>`
	`<p>Bienvenidos a`
	`  <a href="www.garceta.es" />`
	`</p>`

- Todos los valores de atributos deben estar entre comillas, simples o dobles y las dos del mismo tipo, (recomendamos usar las dobles para distinguir otros lenguajes).

Incorrecto	Correcto

`<table width=800>`
`...`
`</table>`
	`<table width="800">`
	`...`
	`</table>`

- Todos los atributos deben tener un valor.

  Para los atributos de tipo booleano, que en HTML no especificamos su valor, se repite el nombre del atributo como valor.

Incorrecto	Correcto

`<input type="checkbox"`
`  value="1" checked>`
`  Acepto las condiciones`
`</input>`
	`<input type="checkbox"`
	`  value="1" checked="checked">`
	`  Acepto las condiciones`
	`</input>`

(Lista de atributos booleanos: compact, checked, declare, readonly, disabled, selected, defer, ismap, nohref, noshade, nowrap, multiple, noresize)

Actividad 2.24:

Reescribe el formulario pedido.html en XHTML y guárdalo en otra página.

2.7. HTML 5

Esta versión aporta nuevos elementos enfocados a distintas tareas como la presentación, el diseño de la página (layout), facilitar la inclusión de audio y video, mejorar los formularios, almacenar los datos de sesión (evitando usar cookies), generar eventos "server-sent" desde el servidor, permitir la geolocalización del sitio web, construir una superficie de dibujo llamada "canvas", arrastrar objetos de un lugar a otro de la página, entre otras.

Pero no se trata solamente de incluir nuevos elementos, esta versión tiene el objetivo más ambicioso de convertirse en una plataforma de desarrollo, que integre todas las tecnologías web. Hoy en día para desarrollar una web deben combinarse diferentes lenguajes (como HTML, CSS, Javascript...), diferentes formatos de audio y video, animaciones, etc. A veces esto crea problemas de compatibilidad con algunos navegadores y diferentes resultados al visualizar una página según el navegador o la versión utilizada. Además, exige al usuario la descarga y actualización de los plug-in para reproducción de animaciones y videos.

HTML 5 pretende ser una solución a esto incorporando una API (Application Program Interface), es decir, una biblioteca de funciones lo bastante amplia para que todos los navegadores se comporten de igual forma.

También se produce una simplificación importante a nivel de atributos, todos los elementos de HTML 5 tienen un conjunto común llamado **global attributes**.

En cuanto a la sintaxis, es muy flexible y pone fin a la bifurcación entre HTML y XHTML convirtiéndose en un estándar para los dos lenguajes.

Características:

- Se permiten las mayúsculas en las etiquetas.
- La etiqueta de cierre es opcional en los elementos vacíos.
- Es opcional establecer un valor a los atributos.
- Las comillas son opcionales en los atributos.

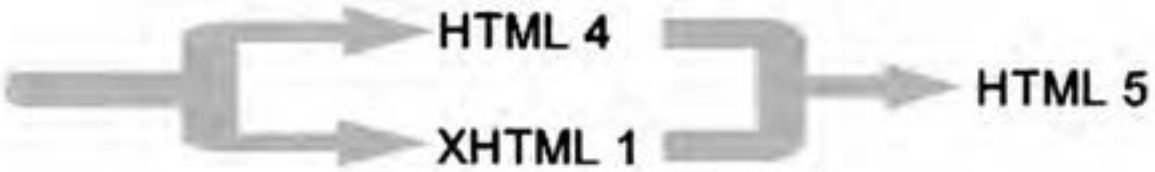


Figura 2.32: Origen del lenguaje HTML 5

Dado que HTML 5 es todavía un lenguaje en construcción, algunos de sus elementos o atributos no son soportados por todos los navegadores, lo cual significa que algunos objetos, por ejemplo, los controles de formularios no se visualicen de igual forma.

2.7.1. Elementos de sección

Para evitar el uso indiscriminado de etiquetas `<div>` como secciones de página, aparecen nuevas etiquetas más específicas que facilitan la comprensión del código, y permitirán a los navegadores y a los buscadores, clasificar dichas secciones según su utilidad e importancia.

<header>

Sirve para insertar información introductoria, como el título, el logo, o algún enlace secundario.

<nav>

Sirve para insertar una sección con enlaces a otras páginas o bloques importantes dentro del documento.

<section>

Sirve para insertar una sección general dentro de un documento.

Suele contener los elementos `<h1-h6>` para crear una jerarquía de contenidos.

Puede anidarse para crear subsecciones.

<article>

Sirve para crear un componente autónomo con el objetivo de que pueda ser reutilizado en otro lugar, por ejemplo, un post de foro, un artículo en un periódico, una entrada en un blog, un comentario de usuario, etc.

Pueden anidarse, de modo que los artículos internos estén asociados al artículo externo, como por ejemplo, los comentarios de usuarios sobre un artículo de periódico.

<aside>

Sirve para insertar una sección relacionada indirectamente con el contenido que tiene a su alrededor, por lo que se considera contenido independiente.

Puede utilizarse por ejemplo para una barra lateral, publicidad o enlaces de navegación.

<footer>

Sirve para insertar un pie de página o de sección, que suele contener el autor, el copyright, el año, disclaimer, mapa del sitio, etc.

<hgroup>

Sirve para agrupar un conjunto de encabezamientos h1-h6.

Estructura general:

```
<body>
  <header>...</header>
  <nav>...</nav>
  <section>
    <article>...</article>
    <article>...</article>
    <article>...</article>
    <aside>...</aside>
  </section>
  <footer>...</footer>
</body>
```

A su vez cada artículo puede reproducir la estructura general, es decir, puede tener un `<header>`, varios `<section>` y un `<footer>`.

Ejemplo de `<article>`:

```
<article>
 <hgroup>
  <h1>Desarrollo web</h1>
  <h2>Tipos de lenguajes</h2>
 </hgroup>
 <p>Todos los lenguajes web provienen de SGML.</p>
 <section>
  <h1>HTML</h1>
  <p>Lenguaje de Marcas de HiperTexto.</p>
 </section>
 <section>
  <h1>XML</h1>
  <p>Lenguaje de Marcas Extensible.</p>
 </section>
</article>
```

Global attributes:

`Acceskey`, `class`, `contenteditable`, `contextmenu`, `dir`, `draggable`, `hidden`, `id`, `lang`, `spellcheck`, `style`, `tabindex`, `title` y los atributos para manejar eventos.

(Para la lista completa: http://dev.w3.org/html5/spec/Overview.html#global-attributes)

HTML 4

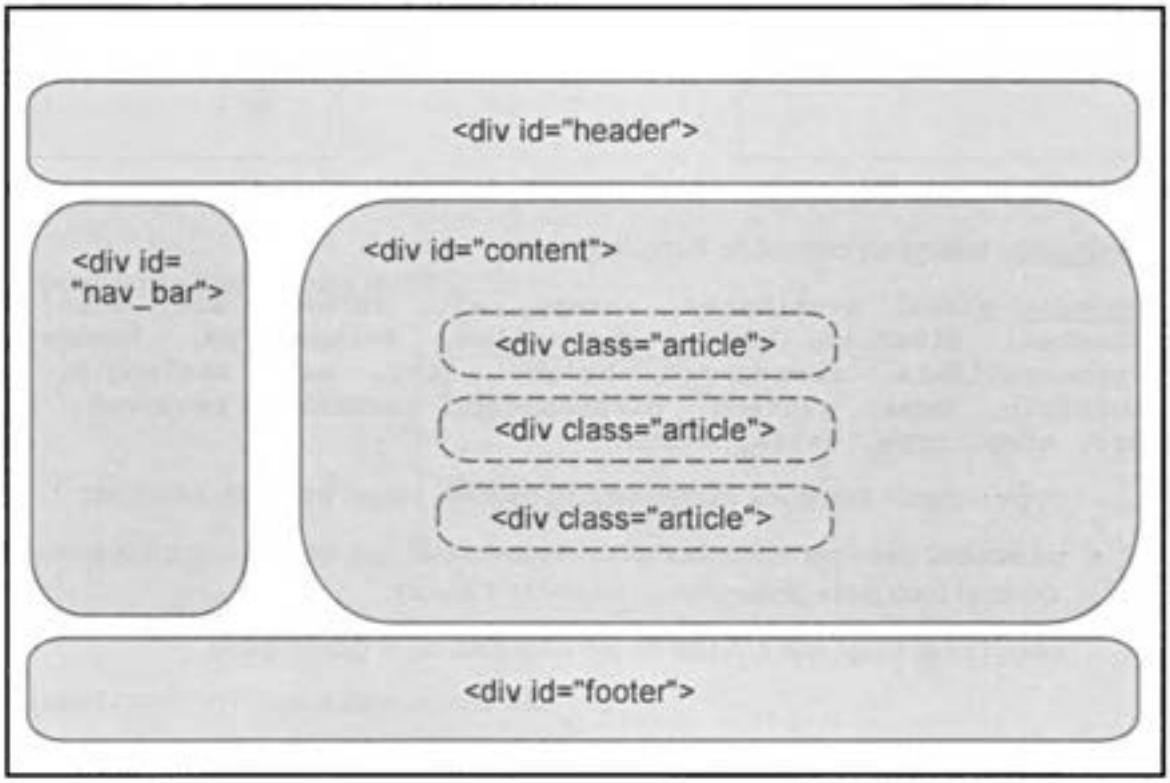


Figura 2.33 a: Layout típico de página web en HTML 4

HTML 5

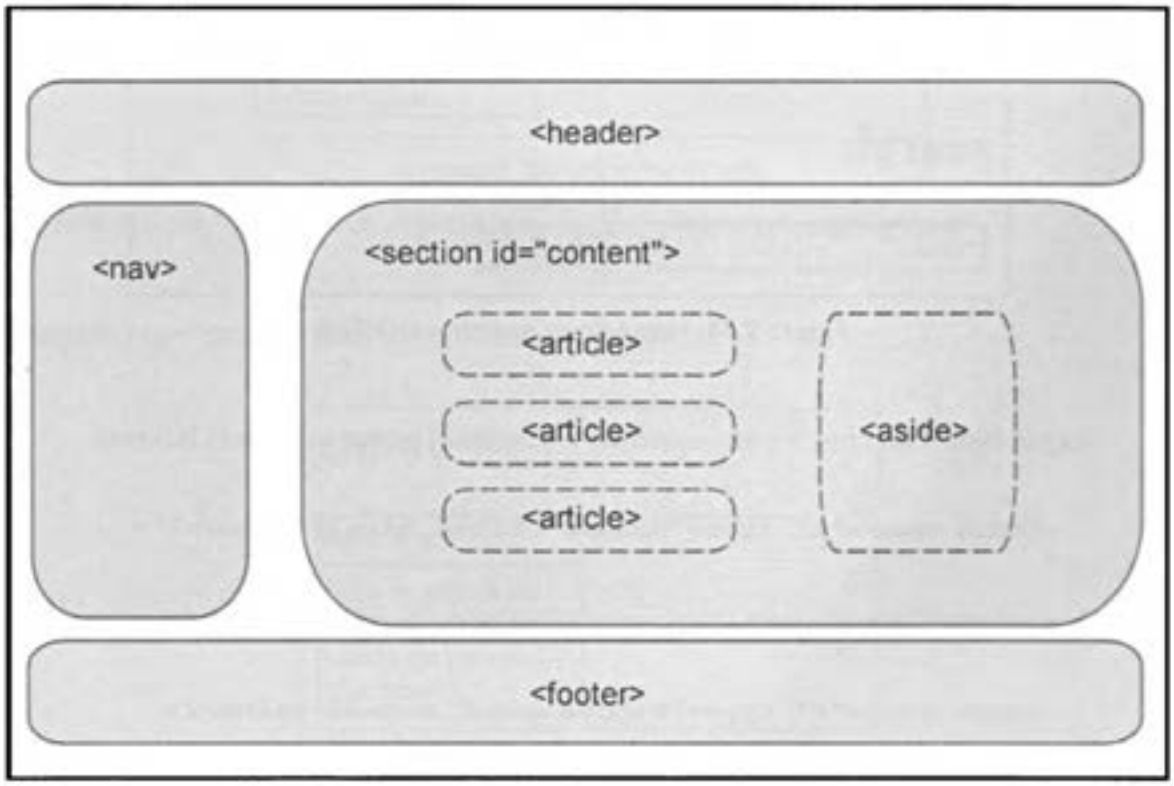


Figura 2.33 b: Layout equivalente en HTML 5

2.7.2. Formularios

El elemento `<input>` incorpora nuevos "type" y nuevos atributos que facilitan la validación de los datos de entrada.

<input>

Definición: inserta un control de formulario.

Atributos: **`global attributes, accept, alt, autocomplete, autofocus, checked, disabled, form, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, multiple, name, pattern, placeholder, readonly, required, size, src, step, type, value, width.`**

- **`type:`** puede tomar los valores search, number, range, time, url, email, etc.
- **`placeholder:`** permite insertar un texto inicial que se borra automáticamente al situar el foco en la caja.
- **`required:`** exige que el valor de ese elemento no se quede vacío.

Ejemplos:

<**input** type="search"> para cajas de búsqueda.

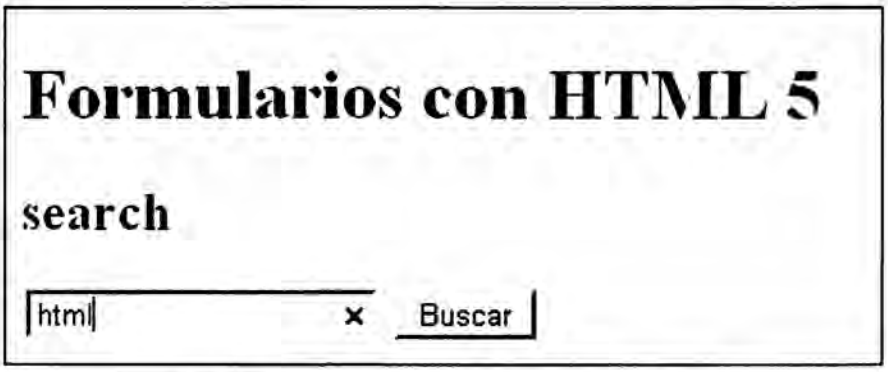


Figura 2.34: <input type="search"> en HTML 5

<**input** type="number"> para aumentar o disminuir números usando botones.

```
<input name="n" type="number" min=0 max=20 value=2">
```

<**input** type="range"> para seleccionar un valor en un intervalo usando botones.

```
<input name="r" type="range" min=0 max=20 value=2>
```

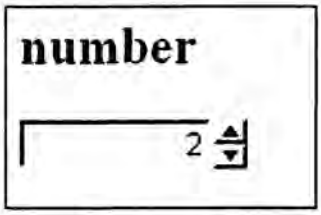


Figura 2.35: <input type="number">

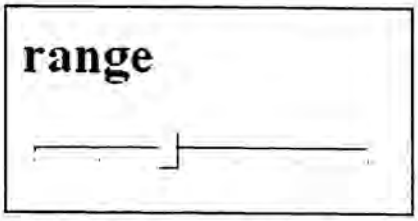


Figura 2.36: <input type="range">

<**input** type="time"> para la hora.

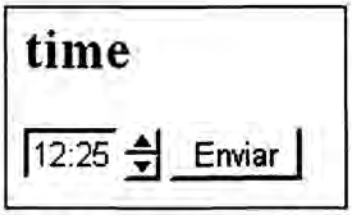


Figura 2.37: <input type="time">

<**input** type="url"> para direcciones web.

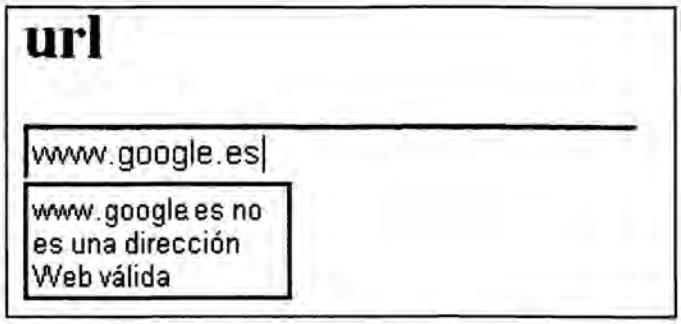


Figura 2.38: <input type="url">

El formato correcto incluye el protocolo, es decir, http://www.google.es

<**input** type="email"> para direcciones de correo.

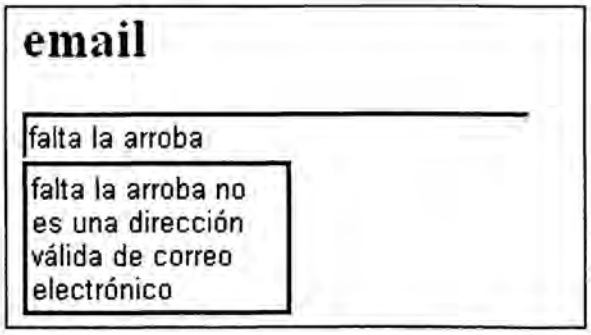


Figura 2.39: <input type="email">

El formato correcto es "nombre" + "@" + "servidor de correo".

<input type="date"> para seleccionar un día en un calendario.

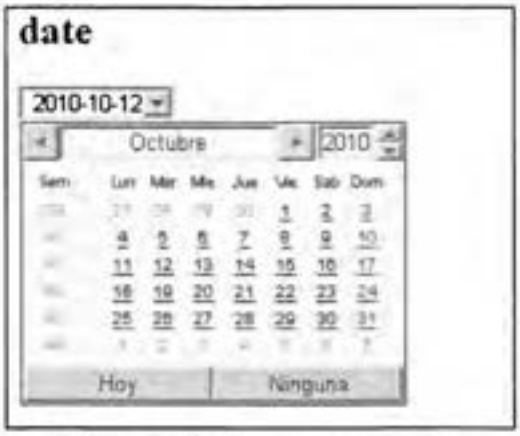


Figura 2.40: <input type="date">

<input type="datetime"> para una fecha y hora absolutas según UTC.

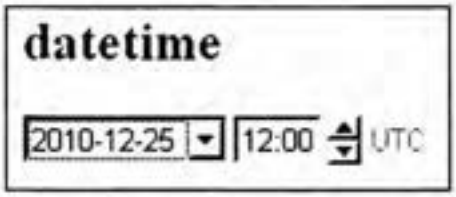


Figura 2.41: <input type="datetime">

Figura 2.42: Atributo placeholder

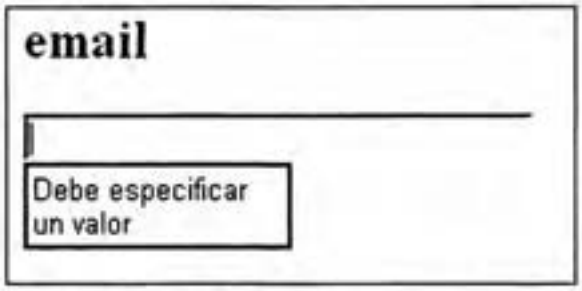


Figura 2.43: Atributo required

Actividad 2.25:

Reescribe el formulario pedido.html utilizando los controles de HTML5.

2.7.3. Objetos multimedia

HTML 5 incluye soporte nativo multimedia, es decir, permite reproducir videos, animaciones, sonido, etc. sin necesidad de tener instalados plugin de ninguna clase. Las nuevas etiquetas **<audio>** y **<video>** permiten incorporar multimedia fácilmente en nuestra página web, y los nuevos atributos como "**controls**" permiten al usuario iniciar o detener la ejecución.

<video>

Definición: sirve para insertar un video o animación.

Atributos: **global attributes, src, poster, preload, autoplay, loop, controls, width, height**.

- **src**: proporciona la dirección URL del video a reproducir.
- **poster**: sirve para visualizar una imagen mientras no está disponible el video.
- **preload**: sirve para mejorar la visualización (valores none, metadata y auto).
- **autoplay**: es un booleano que indica la reproducción automática del video.
- **loop**: es un booleano que indica la repetición automática del video.
- **controls**: es un booleano que indica al navegador que incorpore controles de usuario.

La especificación actual de HTML 5 no indica qué formatos de video se deben utilizar, pero los tipos más habituales son:

· **ogg**: se trata de un formato contenedor, es decir, contiene diferentes datos de audio y video no comprimidos en un solo archivo. Fue desarrollado por Xiph.org como código abierto. El codec de video es Thedora y el codec de audio es Vorbis.

Extensiones de archivo: .ogg, .ogv, .oga, .ogx.

· **mp4**: se trata de un formato contenedor de datos multimedia especificado en el estándar mpeg-4. El codec de video es H.264 y el codec de audio es AAC.

Extensiones de archivo: .mp4, .m4a, .m4p, .m4v, .3gp, .3g2.

· **matroska**: se trata de un formato contenedor de datos multimedia, que pretende ser la alternativa de código abierto a otros formatos como avi, mov, mp4.

Extensiones de archivo: mkv, mka, mks.

Ejemplo:

```
<video src="mivideo.ogv" width="300" height="200" controls>
  Tu navegador no soporta la etiqueta "video".
</video>
```

<audio>

Definición: inserta un archivo de sonido en el documento.

Atributos: **global atttributes, src, poster, preload, autoplay, loop, controls.**

La especificación actual de HTML 5 tampoco indica qué formatos de audio se deben utilizar, pero los tipos más habituales son: ogg, mp3 y wav.

Ejemplo:

```
<audio src="miaudio.wav" autoplay controls>
  Tu navegador no soporta la etiqueta "audio".
</audio>
```

<source>

Definición: sirve para especificar con mayor detalle el archivo fuente multimedia. Permite especificar distintos archivos fuente, en ese caso el navegador visualiza el primero con un formato reconocible. Debe utilizarse en combinación con **<video>** o **<audio>.**

Atributos: **global attributes, src, type, media.**

- **type**: indica el tipo de formato contenedor.

  Valores posibles: ogg, mp4, x-matroska…

Ejemplo:

```
<video controls poster="downloading.jpg">
   <source src="mivideo.ogv" type="video/ogg">
   <source src="mivideo.mp4" type="video/mp4">
   <source src="mivideo.mkv" type="video/x-matroska">
</video>
```

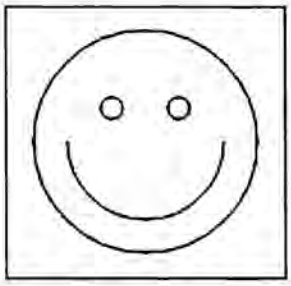
Figura 2.44: Etiqueta <video> con controles en HTML 5

<canvas>

Definición: define un área para contener gráficos, que pueden ser simples imágenes, animaciones o gráficos vectoriales.

Atributos: **global attributes, width, height.**

Ejemplo: para dibujar en el **canvas** tenemos que utilizar el lenguaje Javascript.



Métodos usados:

ctx = canvas.getContext() para establecer contexto 2d
ctx.beginPath() indica el comienzo de los dibujos
ctx.arc() para dibujar un arco
ctx.moveTo() para desplazar el pincel
ctx.fillStyle(), ctx.fill() para colorear un área
ctx.stroke() realiza el dibujo

Figura 2.45: Gráfico con canvas

```
<html>
 <head>
  <script>
   function dibuja(){
          // usamos var1 para manejar el canvas
    var var1 = document.getElementById('smile');

    if (var1.getContext)
     {
          // usamos getContext para dibujar en 2 dimensiones
      var ctx = var1.getContext('2d');

      ctx.beginPath();
      ctx.arc(80,80,50,0,Math.PI*2,false); // Circunf. exterior
      ctx.moveTo(115,80);
      ctx.arc(80,80,35,0,Math.PI,false);   // Sonrisa
      ctx.moveTo(70,65);
      ctx.arc(65,65,5,0,Math.PI*2,false);  // Ojo izquierdo
      ctx.moveTo(100,65);
      ctx.arc(95,65,5,0,Math.PI*2,false);  // Ojo derecho
      ctx.fillStyle='#efd';
      ctx.fill();                    // Colorea
      ctx.stroke();            // Dibuja
     }
    else {
     alert('El navegador actual no soporta canvas');}
   }
  </script>
 </head>
 <body onload="dibuja();">
  <canvas id="smile" width=160, height=160>
 </body>
</html>
```

2.8. Validación

La validación es un procedimiento bastante sencillo, que consiste en comparar nuestro documento con un modelo, que especifica las normas para escribir en un determinado lenguaje. Dicho modelo es lo que llamamos Definición de Tipo de Documento (DTD).

Es cierto que la validación no es necesaria, de hecho, podemos encontrar muchas páginas famosas de Internet que no son válidas, pero para comenzar es bastante recomendable validar nuestros documentos. Por otro lado, cuando estudiemos XML veremos que la validación toma una importancia relevante.

Ahora ha llegado el momento de aclarar en que consiste una DTD.

Para empezar las palabras reservadas y símbolos que se usan son:

- **<!ENTITY>** : variable para definir un enlace a un objeto
- **<!ELEMENT>** : descripción de un elemento
- **<!ATTLIST>** : descripción de atributos de un elemento
- ***** : cero o más ocurrencias
- **+** : una o más ocurrencias
- **?** : cero o una ocurrencia
- **|** : alternativas
- **#PCDATA** : texto sin etiquetas o marcas, sólo datos.
- **<!-- -->** : comentarios

Cada declaración tiene el siguiente formato:

<!ELEMENT *nombre_elemento* *opciones_de_aparición* --comentarios-->

opciones_de_aparición:

- - significa que las etiquetas inicial y final son obligatorias.

- O significa que la etiqueta final es opcional.

O O significa que las dos etiquetas son opcionales.

- O EMPTY significa que la etiqueta final está prohibida (elemento vacío).

<!ATTLIST *nombre_elemento* *definiciones_de_atributos* >

definiciones_de_atributos:

Contiene el nombre del atributo, el tipo de atributo o un conjunto de posibles valores y las claves #IMPLIED | #REQUIRED | #FIXED dependiendo de si el valor por defecto es implícito, requerido o fijado respectivamente.

(#IMPLIED es para atributos no obligatorios y en caso de que no le demos un valor, no toma valor por defecto)

Ejemplo:

```
<!ENTITY % html.content "HEAD, BODY">

<!ELEMENT HTML O O (%html.content;)     -- document root element -->
<!ATTLIST HTML
  %i18n;                                -- lang, dir --
  >
```

Esta declaración significa que el elemento **<html>** tiene como contenido los elementos **<head>** y **<body>**, las etiquetas inicial y final son opcionales, y los atributos son **lang** y **dir**.

```
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- document title -->
<!ATTLIST TITLE %i18n>
```

Esta declaración significa que el elemento **<title>** tiene etiquetas inicial y final obligatorias, y los atributos disponibles son **lang** y **dir**.

```
<!ELEMENT META - O EMPTY                -- generic metainformation -->
<!ATTLIST META
  %i18n;                                -- lang, dir use with content--
  http-equiv  NAME        #IMPLIED  -- HTTP response header name --
  name        NAME        #IMPLIED  -- metainformation name --
  content     CDATA       #REQUIRED -- associated information --
  scheme      CDATA       #IMPLIED  -- select form of content --
  >
```

Esta declaración significa que el elemento **<meta>** es vacío, por tanto, no debemos poner etiqueta de cierre. Dispone de los atributos **%i18n;**, **http-equiv**, **name**, **content** y **scheme**, de los cuales **content** es obligatorio.

En la práctica podemos validar nuestro documento de dos maneras, en línea utilizando el validador del W3C o en local usando diversas herramientas.

Tipos de validación:

- Validación en línea (W3C)
- Validación local
  - Complementos del navegador (HTML validator)
  - Entornos de desarrollo (Adobe Dreamweaver)
  - Validadores específicos (TidyGUI)

2.8.1. Validación en línea

El consorcio W3C ha creado una herramienta que se puede utilizar gratuitamente a través de Internet ► http://validator.w3.org/

Figura 2.46: Validación en línea mediante URI

Dentro de la validación en línea, disponemos de tres opciones:

- **Validate by URI**, permite escribir la dirección de la página que se quiere validar.

  Esta opción es la más sencilla para validar las páginas que ya están publicadas en Internet.
- **Validate by File Upload**, muestra un formulario mediante el que se puede subir el archivo HTML correspondiente a la página que se quiere validar.

  Esta opción es la adecuada para validar páginas web terminadas pero que aún no se han publicado en Internet.
- **Validate by Direct Input**, permite validar código HTML, insertándolo de forma directa en el cuadro de texto.

  Se trata de la opción más rápida para validar trozos o páginas HTML pequeñas.

La siguiente imagen muestra el resultado de la validación de fallida realizada mediante la opción: Validate by URI.

Errors found while checking this document as HTML 4.01 Strict!

Result:	2 Errors, 3 warning(s)	

Source :	`<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
	`<html>`
	`  <head>`
	`  <title> titulo de la página </title>`
	`  </head>`
	`  <body>`
	`  Contenido del documento`
	`  </body>`
	`</html>`	

Encoding :	utf-8	(detect automatically)
Doctype :	HTML 4.01 Strict	HTML 4.01 Strict
Root Element:	html	

Figura 2.47: Errores de validación

Si la página no pasa correctamente la prueba de validación, se muestra el listado completo de errores junto con la ayuda necesaria para resolverlos.

Aviso:

No Character encoding declared at document level

No se ha especificado un juego de caracteres para el documento, el validador asumirá UTF-8.

Lista de errores:

Line 7, Column 5: **character data is not allowed here**

Line 8, Column 8: **end tag for "BODY" which is not finished**

En este caso, al utilizar la DTD strict para validar, no es aceptable escribir texto suelto en el cuerpo, sin encerrarlo en un elemento tal como `<p>texto</p>`.

La siguiente imagen muestra una validación con éxito:

This document was successfully checked as HTML 4.01 Transitional!

Result:	Passed, 2 warning(s)
Source :	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html> <head> <title> titulo de la página </title> </head> <body> Contenido del documento </body> </html>
Encoding :	utf-8 (detect automatically)
Doctype :	HTML 4.01 Transitional (detect automatically)
Root Element:	HTML

Figura 2.48: Documento válido

En este caso podemos insertar una imagen que informa a los usuarios de que nuestra página es válida, bastaría con añadir el código que nos proporcionan:

```
<a href="http://validator.w3.org/check?uri=referer">
    img src=http://www.w3.org/Icons/valid-html401
    alt="Valid HTML 4.01 Transitional" height="31" width="88">
</a>
```

Figura 2.49: Icono de código válido

Actividad 2.26:

Valida tu página publicada, mediante el validador en línea de W3C.

Corrige los errores por orden estricto de aparición y observa los warning.

Inserta la imagen que indica código validado.

2.8.2. Validación local

- **Complementos del navegador**

En el navegador Firefox podemos instalar el validador mediante un complemento llamado HTML Validator (https://addons.mozilla.org/es-ES/firefox/addon/249).

Tras su instalación, la primera vez que se reinicia Firefox se muestra la siguiente ventana de configuración:

Figura 2.50: Validación mediante complementos de navegador

En ella se solicita al usuario que configure el tipo de validación que se va a realizar.

Las opciones para elegir son:

· HTML Tidy (que ofrece ayuda para resolver los errores y es mejor para HTML).

· SGML Parser (ofrece menos ayuda, pero es el mismo que el validador del W3C).

· Serial (que realiza las dos validaciones de forma consecutiva).

Una vez configurado el validador, abre cualquier página web y verás cómo en la esquina inferior derecha de Firefox se muestra un pequeño icono que indica si la página es válida o no. Cuando la página no es válida, aparece el icono con forma de cruz, si colocamos el puntero del ratón sobre el icono, se muestra la siguiente información:

Figura 2.51: Errores de validación

Haciendo doble clic se muestran todos los errores detalladamente.

- **Validación con Adobe Dreamweaver®**

El validador se encuentra integrado en la propia herramienta. Podemos acceder a la configuración del validador desde la opción: Edición > Preferencias > Validador

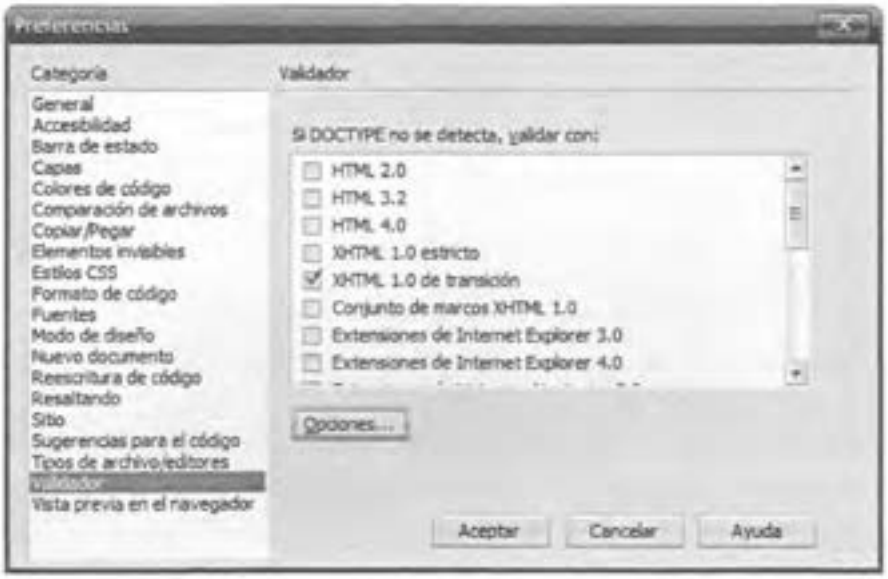


Figura 2.52: Preferencias de validación con Dreamweaver

En esta ventana de configuración se puede elegir el DTD por defecto que se utiliza en caso de que la página web no la indique.

Para ejecutar la herramienta de validación de Dreamweaver usamos el icono:

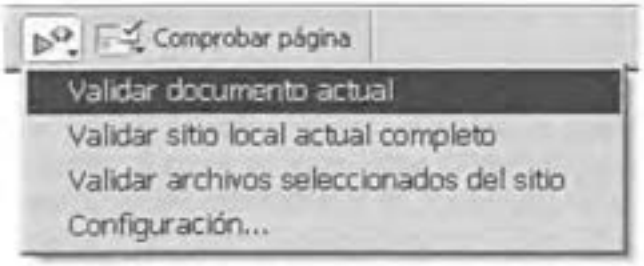


Figura 2.53: Validar documento actual

Si no se han producido errores al validar la página, Dreamweaver lo indica mediante un mensaje que declara a la página como válida. Si se produce algún error, la página no es válida y Dreamweaver muestra la lista de todos los errores encontrados junto con sus posibles soluciones:

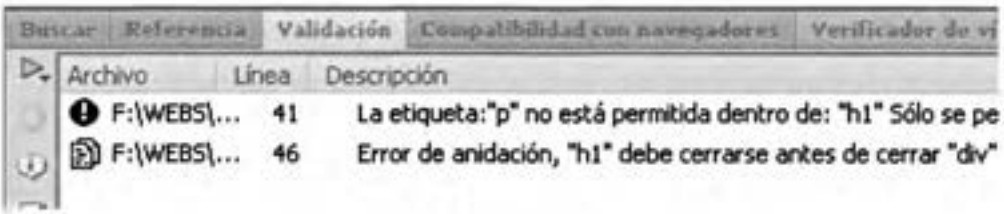


Figura 2.54: Errores de validación

- **Validadores específicos**

En caso de necesitar un validador ligero sin conexión a Internet y sin depender de ninguna herramienta de diseño, disponemos de TidyGUI.

Su uso es muy simple, basta con introducir el fichero fuente y ejecutar Tidy!

A continuación se muestra un ejemplo con 2 warning

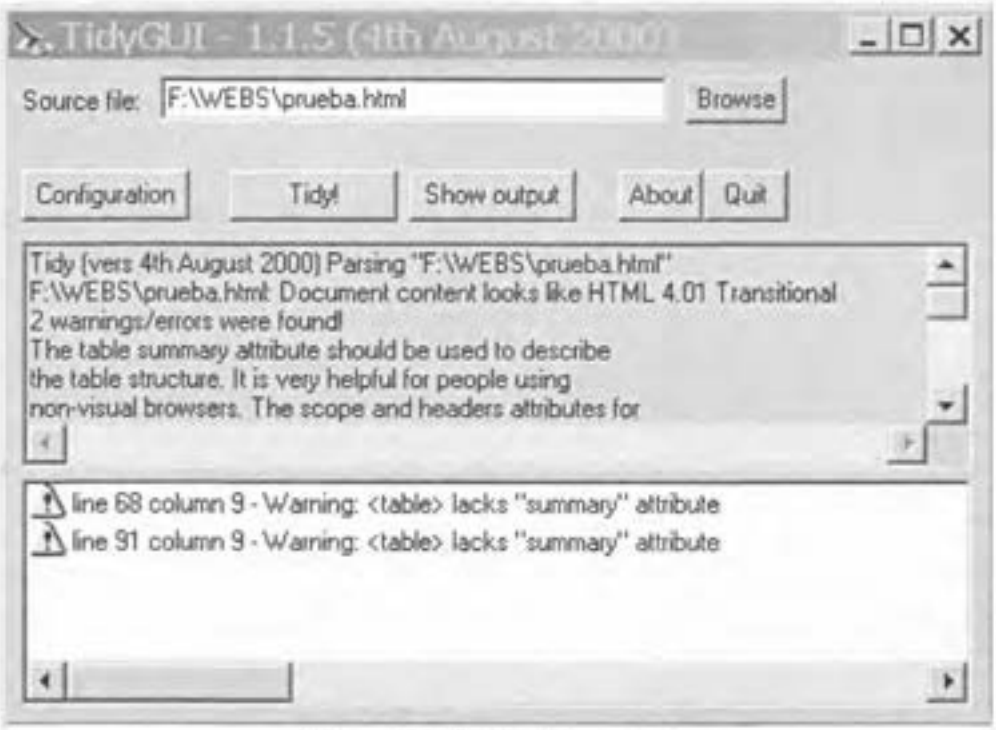


Figura 2.55: Validación con Tidy

Una vez resueltos:

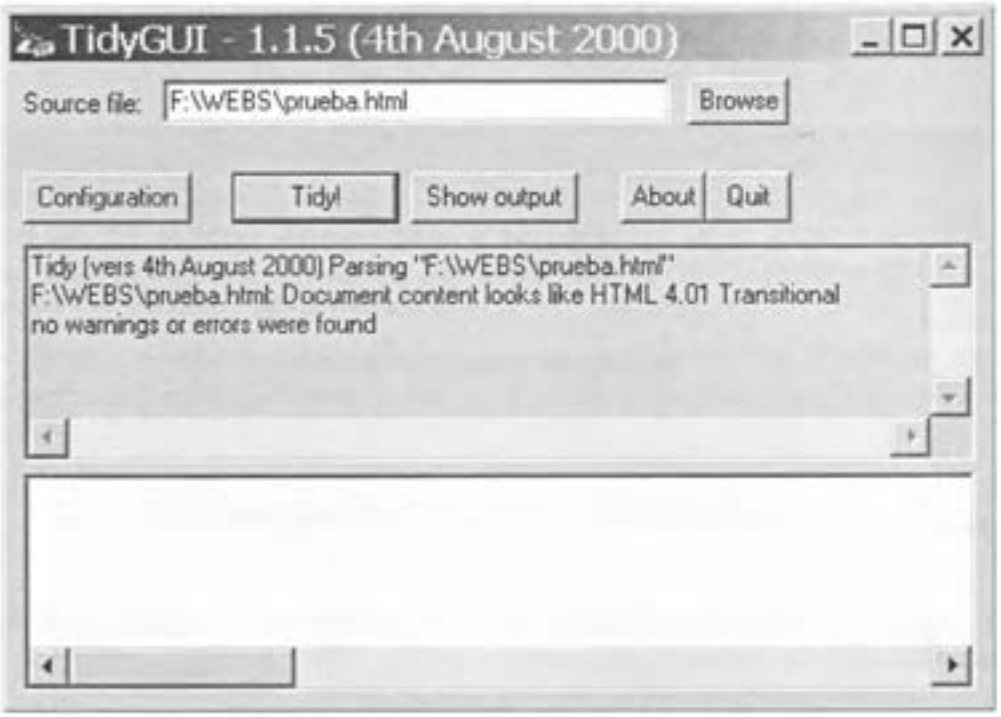


Figura 2.56: Documento válido

Ejercicios propuestos

Ejercicio 1.

Construye una página con el siguiente contenido formateado:

Programación orientada a objetos

La **Programación Orientada a Objetos** (POO, o en inglés OOP, *Object Oriented Programming*) es un paradigma de programación que pretende desarrollar aplicaciones basándose en el comportamiento de los objetos del mundo real.

Conceptos fundamentales

- **Clase**: modelo, molde o patrón a partir del cual se crearán instancias concretas (objetos). Al crearse se definirán su nombre, sus atributos, sus métodos y si es construida a partir de otra clase ya existente (*herencia*).
- **Objeto**: instancia concreta de una clase. Tendrá un estado concreto y ocupará un espacio en memoria.
- **Atributo**: característica de un objeto (o clase). Los atributos de un objeto se definen al construir la clase de la que luego se instanciará el objeto. También se denomina en ocasiones *propiedad*.
- **Método**: funcionalidad asociada a un objeto (o clase) y que se definen al construir la clase de la que luego se instanciará el objeto.

Java: un ejemplo

El lenguaje de programación Java es un exponente de la programación orientada a objetos. Un pequeño ejemplo de un código en Java sería:

```
public class Circulo {
    Punto centro;
    int radio;
    Circulo(Punto centro, int radio) {
        this.centro= centro;
        this.radio= radio;
    }
}
```

Figura 2.57: POO

Ejercicio 2.

Realiza una lista que muestre en su primer nivel los nombres de los planetas del sistema solar y, en un segundo nivel, los satélites de cada planeta.

Ejercicio 3.

Realiza mediante una tabla el siguiente mosaico:

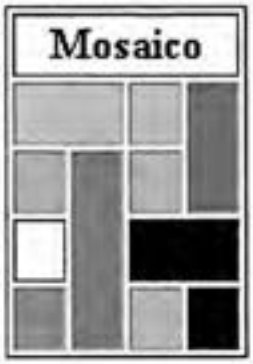


Figura 2.58: Mosaico

Ejercicio 4.

Realiza un mapa de imágenes a partir de las siguientes figuras.

Figura 2.59: Mapa ΔOX□

Cada una de las figuras será un enlace que conectará con las páginas triangulo.html, circulo.html, aspa.html y cuadrado.html respectivamente (deberán ser creadas). Cada una de estas cuatro páginas tendrá el siguiente aspecto:

Triángulo

Figura 2.60: Triangulo.html

Ejercicio 5.

Diseña un formulario de alta de usuario con la siguiente apariencia:

Figura 2.61. Formulario de alta.

Los campos tendrán las siguientes características:

Identificador de nuevo usuario: texto; sólo lectura; valor por defecto usu_012345

Nombre, Apellidos y E-Mail: texto

Contraseña y Confirmar contraseña: contraseña

Sexo [Varón | Mujer]: botones de radio; valor por defecto seleccionado Mujer

Aficiones [Deporte | Lectura | Cine]: cuadros de chequeo; por defecto seleccionado Lectura

País [España | Francia | Portugal]: menú desplegable; por defecto seleccionado Francia

Colores favoritos [Rojo | Verde | Azul]: menú desplegable con selección múltiple; por defecto seleccionados Rojo y Azul

Subir su foto: campo de tipo fichero

Comentarios personales: área de texto con 40 columnas y 5 filas; por defecto aparecerá Escriba aquí sus comentarios

Resetear: botón de reseteo Enviar: botón de envío

Modificar datos del usuario: botón deshabilitado

Ejercicio 6.

Diseña un sencillo formulario con los siguientes campos:

Usuario: campo de texto de tamaño 20 y longitud máxima de 12

Contraseña: campo de contraseña de tamaño 12 y longitud máxima de 12

Servicio [http| ftp| file]: menú desplegable; selección por defecto ftp

Enviar: imagen de submit

Los campos se agruparán con una etiqueta fieldset con título *Datos personales*. Los campos deben poder recorrerse en el orden en que se visualizan mediante la tecla tabulador (usa el atributo tabindex). Cada campo dispondrá de una tecla de atajo para acceder a él rápidamente (usa atributo accesskey).

Ejercicio 7.

Realiza una página que disponga de un índice de contenidos ubicado al principio de la misma. Cada una de las entradas del índice de contenidos será un enlace que conecte con una sección dentro del documento. Al final de cada sección, aparecerá un enlace que lleve al principio del documento. Como modelo de índice consulta alguna página de Wikipedia, como por ejemplo la referida a la CPU:

(http://es.wikipedia.org/wiki/Unidad_Central_de_Procesamiento).

Ejercicio 8.

Realiza una página que utilice una tabla (con el borde oculto) como forma de maquetación. La tabla dispondrá de dos filas:

- La primera fila se expandirá dos columnas y contendrá el título de un cuento, como "Hansel y Gretel".
- La segunda dispondrá de dos columnas. La primera celda contendrá un índice de capítulos de un libro (3 capítulos), enlazados respectivamente al documento capitulo1.html, capitulo2.html y capitulo3.html. La segunda celda contendrá el contenido del capítulo 1. Al documento generado se le llamará capitulo1.html.

Una vez se disponga de esta página, copia renombrándola como capitulo2.html y sustituye la celda del contenido con el del capítulo 2.

Repite de nuevo el proceso con el capítulo 3.

Figura 2.62: Hansel y Gretel

Ejercicio 9.

Realiza mediante marcos el siguiente diseño:

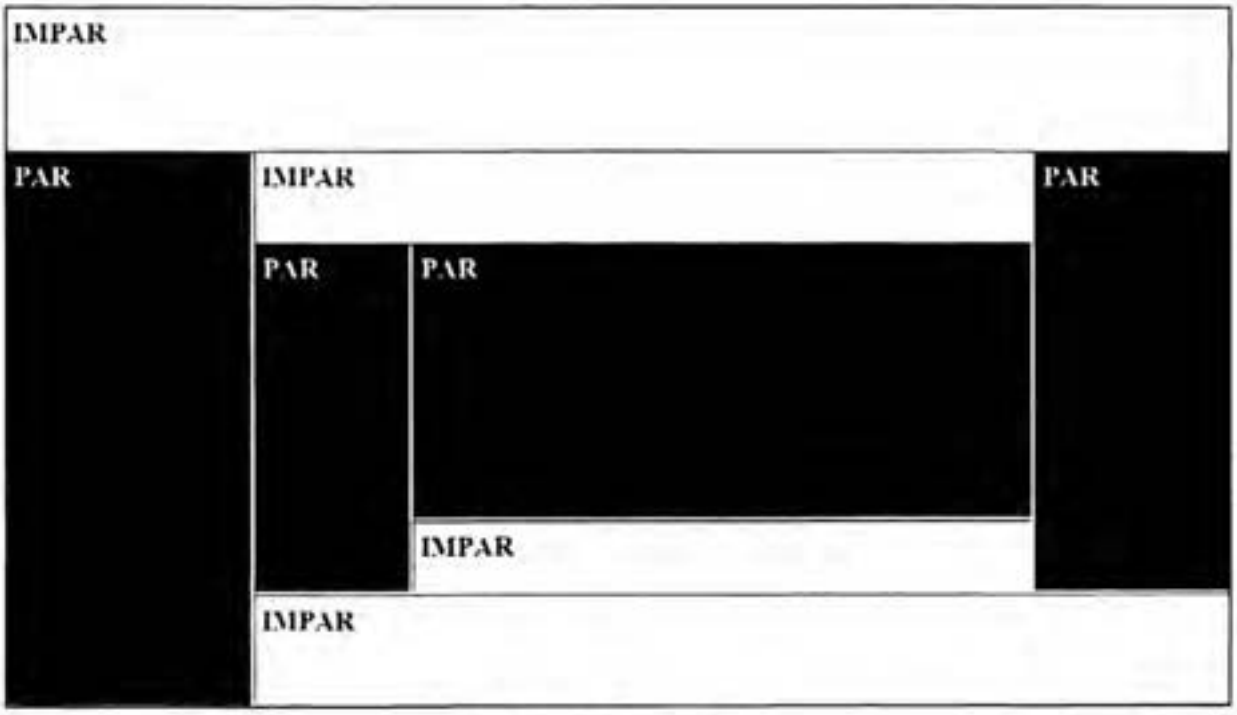


Figura 2.63: Marcos

Las páginas que se cargarán en cada marco se llamarán respectivamente par.html e impar.html y deberán ser creadas igualmente.

Ejercicio 10.

Descarga los siguientes objetos multimedia: oso.mp4, lemniscata.swf, risa.wav y aviso.jpg de la dirección www.juanmacr.es/recursos (user: Karpov, sin contraseña).

Inserta estos objetos utilizando los elementos <object>, <iframe>, <audio> y <video>.

Abre la página en varios navegadores comprobando el diferente soporte de que disponen para los elementos multimedia.

Ejercicio multimedia

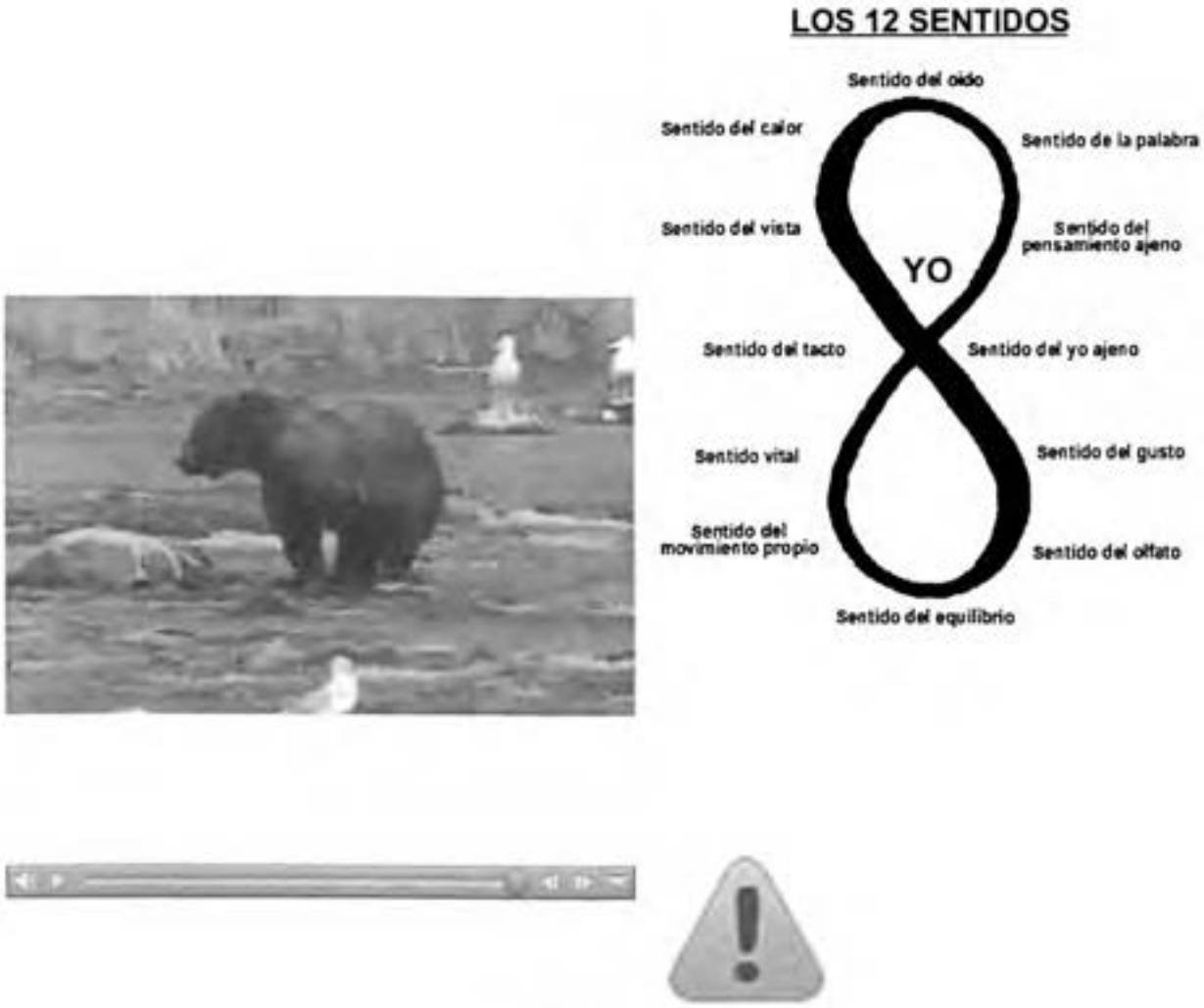


Figura 2.64: Multimedia

CAPÍTULO 3

CSS. Hojas de estilo

CONTENIDO

Introducción
Sintaxis
Selectores
Modelo de cajas
Texto
Listas
Tablas
Formularios
Layout
Prioridad
Miscelánea
Ejercicios propuestos

3.1. Introducción

En primer lugar debe quedar claro que CSS (Cascading Style Sheets - Hojas de Estilo en Cascada) no es un lenguaje de marcas sino un lenguaje de presentación, por tanto, su estructura y sintaxis distan mucho de HTML.

Lo estudiamos porque CSS está íntimamente ligado a los lenguajes de marcas, hasta el punto de que resulta imprescindible para crear sitios web con calidad.

El objetivo principal de CSS es manejar el aspecto y formato de los documentos, liberando de esta forma a HTML de las tareas de presentación.

Esta separación entre contenidos y presentación supone muchas ventajas, la más evidente es que evita la repetición innecesaria de código, como podemos ver en el siguiente ejemplo:

```
<body>
 <h1><font face="Arial" size="5">UNIDAD 2. HTML y XHTML</font></h1>
 <h2><font face="Verdana" size="4">2.5. Contenido del cuerpo</font>
 </h2>
 <h3><font face="Garamond" size="3">2.5.1. Manejo del texto</font>
 </h3>
 <h3><font face="Garamond" size="3">2.5.2. Listas</font></h3>
 <h2><font face="Verdana" size="4">2.7. HTML 5</font></h2>
 <h3><font face="Garamond" size="3">2.7.1. Elementos de sección
     </font></h3>
</body>
```

Que se puede mejorar con este otro:

```
<html>
  <head>
    <style type="text/css">
      h1 {font-family: Arial;}
      h2 {font-family: Verdana;}
      h3 {font-family: Garamond;}
    </style>
  </head>
  <body>
    <h1>UNIDAD 2. HTML y XHTML</h1>
    <h2>2.5. Contenido del cuerpo</h2>
    <h3>2.5.1. Manejo del texto</h3>
    <h3>2.5.2. Enlaces</h3>
    <h2>2.7. HTML 5</h2>
    <h3>2.7.1. Elementos de sección</h3>
  </body>
</html>
```

Con este último código, cada vez que incluyamos un elemento de cabecera **<h1>** al **<h3>**, el tipo de fuente ya estará establecido de antemano por las reglas css, contenidas en **<style>**.

Así pues, podemos citar como ventajas en el uso de CSS las siguientes:

- Disminuye el código a escribir.
- Facilita la generación de código y su mantenimiento.
- Mejora la legibilidad de los documentos.

En cuanto a su evolución histórica, el inicio de las hojas de estilo tiene lugar en torno a 1970, pero el uso generalizado se produjo más tarde con el auge de Internet.

La ausencia de un estándar oficial provocó diferentes formas de establecer reglas de estilo y como consecuencia, las habituales diferencias de visualización entre los navegadores. Como en el caso de HTML, fue el consorcio W3C el encargado de encontrar una especificación oficial que unificara la forma de definir los estilos.

Fue en 1996 cuando surgió la primera versión de CSS llamada CSS nivel 1 partiendo de los trabajos de Bert Bos.

Pero esta versión contenía bastantes problemas que fueron solucionándose con el tiempo, lo que dio lugar a la aparición de la segunda versión en el año 1997.

Entonces, el consorcio W3C decidió crear grupos específicos de trabajo para HTML, CSS y DOM. El estándar oficial CSS nivel 2 se publica en 1998.

Este estándar ha sufrido una revisión llamada **CSS 2.1** que es la especificación actual (consultar en http://www.w3.org/TR/CSS2/).

Desde 1998 hasta hoy el W3C sigue trabajando para establecer la versión CSS 3.

En cuanto al comportamiento de los navegadores respecto las reglas de CSS, podemos decir que todos los navegadores tienen un soporte muy amplio para CSS 2.1, pero no tan completo en su mayoría para la versión CSS 3, aunque progresivamente van incorporando las nuevas propiedades y reglas que contiene.

Para comprobar la compatibilidad de nuestro navegador con las versiones CSS consultar en http://www.css3.info/selectors-test/

En la siguiente tabla, se puede ver que las primeras versiones de Internet Explorer tienen problemas de compatibilidad con la versión CSS 2.1 y casi nula con CSS 3. Los demás navegadores se han adaptado con mayor rapidez a la evolución de CSS.

Navegador: / CSS:	IE6	IE7	IE8	IE9	FF3	FF3.6	FF4	Chrome
CSS 1	Parcial	Total	Total	Total	Total	Total	Total	Total
CSS 2.1	Muy Pobre	Parcial	Total	Total	Buena	Total	Total	Total
CSS 3	Nula	Pobre	Parcial	Total	Parcial	Buena	Total	Total

Tabla 3.1: Compatibilidad de navegadores con CSS

3.2. Sintaxis

3.2.1. Cómo incluir CSS en el documento

Vamos a ver 3 formas distintas de integrar hojas de estilo en documentos HTML.

La primera ya se utilizó en el capítulo anterior para el color de fondo y consiste en utilizar el atributo style en la propia línea. La segunda consiste en incluir el código css en la cabecera del documento, dentro de la etiqueta **<style>**. La tercera consiste en crear un archivo css externo y vincularlo al documento mediante un enlace **<link>** o @import.

- CSS en línea.

```
<body>
 <p style="font-family: Verdana; font-size: medium;">HOLA MUNDO</p>
</body>
```

Se utiliza para indicar el estilo particular de la línea (no utiliza llaves).

- CSS interno.

```
<html>
  <head>
    <style type="text/css">
      body {font-family: Courier New;}
      h1 {font-family: Arial; font-size: x-large;}
      p  {font-family: Verdana; font-size: medium;}
    </style>
  </head>
  <body>
    <h1>Tipo de fuente Arial y tamaño grande</h1>
    <p>Tipo de fuente Verdana y tamaño medio</p>
  </body>
</html>
```

Se utiliza para indicar los estilos propios de esta página, que se complementan con los estilos globales del sitio web, y suelen definirse en un archivo css externo.

Se coloca dentro de la etiqueta **<style>**.

Tipo de fuente Arial y tamaño grande

Tipo de fuente Verdana y tamaño medio

Figura 3.1: CSS interno

- CSS externo.

Creamos el archivo estilos.css en una carpeta como /css:

```
/* Estilos globales */

body {margin: 0px;}

td    {color: #000000;
       font-size: 12px;}

a     {color: #FF6600;
       font-weight: bold;}

a:hover {color: #3366CC;}
```

Después insertamos en cada documento el enlace mediante <link>:

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

Hay otra forma de enlazar el archivo .css, mediante la regla @import:

```
<style type="text/css">
 @import "/css/estilos.css";
</style>
```

Se utiliza para indicar los estilos globales que van a compartir todas las páginas web del sitio.

La prioridad de las reglas en caso de colisión, aumenta cuanto más particular sea la regla, es decir:

+ ▲ 1º css en línea
  | 2º css interno
- | 3º css externo

Dentro de cada una de estas formas, en caso de repetición es la última regla la que se aplica, por ejemplo en este caso, se aplicaría tipo de fuente Verdana:

```
p {font-family: Arial;
   font-family: Verdana;}
```

Podemos aumentar la prioridad utilizando la palabra reservada `!important` , en el ejemplo anterior permitiría aplicar la fuente Arial, a pesar de no ser la última regla.

3.2.2. Cómo construir las reglas CSS

Cada regla CSS sirve para definir un estilo y consiste en la unión de un selector más una declaración entre llaves. A su vez, cada declaración se compone de una o más parejas de propiedades con su valor, terminadas en punto y coma.

Regla CSS

selector { propiedad: valor; propiedad: valor; ...; }

```
p   {font-family: Verdana; font-size: medium;}
```

El selector especifica los elementos de HTML sobre los que se define un estilo.

La propiedad es una característica del selector, en el ejemplo anterior serían la fuente y el tamaño del texto. El listado de propiedades y valores posibles, se encuentra en la especificación oficial del W3C (consultar http://www.w3.org/TR/CSS2/propidx.html).

Las reglas se pueden agrupar en caso de compartir la declaración o el selector:

```
h1 {font-family: Verdana; color: red;}
h2 {font-family: Verdana; color: red;}
```

Se pueden resumir en: `h1, h2 {font-family: Verdana; color: red;}`

Igualmente:

```
h1 {font-family: Verdana;}
h1 {color: red;}
```

Se pueden resumir en: `h1 {font-family: Verdana; color: red;}`

Para los comentarios utilizaremos los símbolos /*,..., */ procedentes del lenguaje C:

```
/* Comentarios */
```

3.3. Selectores

En CSS 2.1 existen los siguientes tipos de selectores:

Selector universal, Selector de tipo, Selector descendiente, Selector hijo, Selector adyacente, Selector de atributos, Selector de clase, Selector de id, Pseudo-clase y Pseudo-elemento.

3.3.1. Selector universal

Sirve para seleccionar todos los elementos de la página.

Se indica mediante el símbolo asterisco, como por ejemplo:

```
* { margin: 0px }
```

3.3.2. Selector de tipo

Sirve para seleccionar todos los elementos de la página cuya etiqueta coincida con el selector, por ejemplo:

```
p {font-family: Verdana; color: red;}
```

3.3.3. Selector descendiente

En primer lugar decimos que D es un elemento descendiente del elemento E, si D está contenido entre la apertura y cierre de E.

El selector descendiente se puede construir con 2 o más selectores simples separados por un espacio en blanco. Un selector del tipo E M {declaración}, sirve para seleccionar todos los elementos M que son descendientes de E a cualquier nivel, por ejemplo:

```
p a {font-size: 50px}
```

Aplicada al código:

```
<p>Texto1</p>
<p><a>Texto2</a></p>
<p><span class="nieto"><a>Texto3</a></span></p>
```

Visualizaría el Texto2 y el Texto3 a tamaño 50.

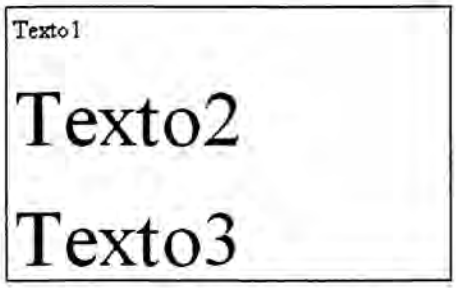


Figura 3.2: Selector descendiente

Si quisiéramos excluir de la selección a los hijos, y seleccionar a partir de los nietos en adelante, combinaríamos con el selector universal, por ejemplo,

```
p * a {font-size: 50px}
```

Visualizaría solamente el Texto3 a tamaño 50.

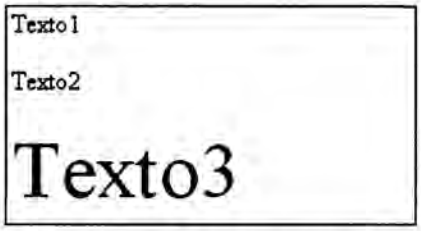
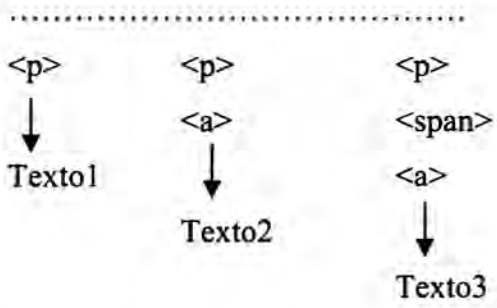


Figura 3.3: Selector descendiente de 2º nivel

El documento HTML genera una estructura en árbol, en ese sentido podemos hablar de elementos padre, hijo, nieto o descendiente en general:



3.3.4. Selector hijo

Sirve para seleccionar el primer descendiente de un elemento, es decir, el "hijo", excluyendo de la selección al resto de descendientes.

Un selector hijo se construye con el signo >, como por ejemplo:

```
p > a {font-size: 50px}
```

Visualizaría solamente el Texto2 a tamaño 50, ya que Texto3 sería descendiente de 2º nivel, pero no sería hijo (descendiente de 1º nivel).

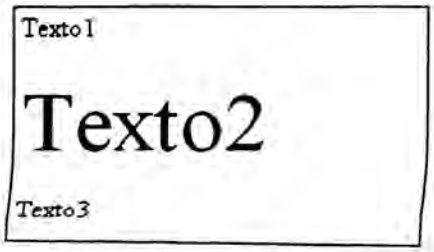


Figura 3.4: Selector hijo

3.3.5. Selector adyacente

Se emplea para seleccionar elementos que son "hermanos" (tienen el mismo padre) y son adyacentes (consecutivos) en el código.

Se construye con el signo +, de forma que una regla como E + M {declaración;}, selecciona a todos los elementos de tipo M, tales que E y M tienen el mismo padre y E precede inmediatamente a M en el código (salvo comentarios).

```
h1 + h2 {font-size: 50px}
```

3.3.6. Selector de atributos

El selector de atributos puede seleccionar de 4 formas distintas:

- [atributo]: selecciona los elementos que tengan ese atributo independientemente del valor que tome (puede tener otros atributos).
- [atributo = valor]: selecciona los elementos que tengan el atributo con el valor especificado (puede tener otros atributos).
- [atributo ~ = valor]: selecciona los elementos que tengan el atributo con al menos uno de los valores como el especificado (puede tener otros atributos).
- [atributo |= valor]: selecciona los elementos que tengan el atributo y que la palabra comience con el valor especificado (solo sirve para el atributo lang).

Ejemplos:

```
span[class] {font-family: Arial;}

span[class = "azulon"] {color: blue;}

span[class ~= "grandote"] {font-size: 50px}

span[lang |= "es"] {color: red;}
```

3.3.7. Selector de clase

Sirve para seleccionar una instancia en particular de un elemento.

Por ejemplo, en el siguiente código tenemos dos instancias del elemento **<p>**, para poder distinguir cada una utilizamos el atributo class.

```
<h3 class = "grandote">Texto1</h3>
<p class = "grandote">Texto2</p>
<p class = "peque">Texto3</p>
```

Se construye de la forma selector_simple.clase, como por ejemplo:

`p.grandote {font-size: 50px}` Visualizaría solamente el Texto2 en grande.

También se puede omitir el selector_simple, por ejemplo:

```
.grandote {font-size: 50px}
```

En este caso selecciona todos los elementos con la clase "grandote".

Visualizaría el Texto1 y el Texto2 en grande, ya que el elemento **<h3>** también tiene la clase "grandote".

Figura 3.5: Selector de clase

3.3.8. Selector ID

Sirve para seleccionar a una sola línea de toda la página.

Se puede utilizar el selector de clase, pero resulta menos eficiente porque está concebido para seleccionar un subconjunto de instancias, para una sola línea es preferible el selector id.

En el código anterior sería más adecuado escribir:

```
<p id="peque">Texto4</p>
```

La regla con el selector id, se construye usando el carácter almohadilla #:

```
#peque {font-size: 8px}
```

Visualizaría solamente el Texto3 en pequeño.

Actividad 3.1:

Prueba todos los selectores vistos hasta ahora, en un nuevo documento html.

Usa CSS interno, es decir, pon las reglas dentro de la etiqueta <style>.

3.3.9. Pseudo-clases

Se utilizan para seleccionar elementos en función, no del código como hasta ahora, sino del comportamiento posterior del usuario (eventos), por ejemplo, en función de los enlaces que visita, el movimiento del ratón, etc.

Algunas pseudo-clases son:

- :link (se aplica a los enlaces nunca visitados)
- :visited (se aplica a los enlaces visitados al menos una vez)
- :hover (se aplica a un elemento seleccionado por el usuario pero sin activarlo, como por ejemplo al situar el ratón encima de un objeto sin hacer clic)
- :active (se aplica a los elementos que están siendo activados, por ejemplo, mientras el usuario presiona el botón izquierdo del ratón)
- :focus (se aplica a un elemento que tiene el foco, por ejemplo, un cuadro de entrada en un formulario)

Algunos ejemplos de pseudo-clases serían los siguientes:

```
a:link {color: black;}                  Enlaces sin visitar -> negro
a:visited {color: blue;}                Enlaces visitados -> azul
a:hover {color: green;}                 Con el ratón encima -> verde
a:active {color: red;}                  Presionando el botón -> rojo
a:focus {background-color: yellow;}     Con el foco-> fondo amarillo
```

Actividad 3.2:

Crea una tabla con 4 enlaces y utiliza las tres primeras pseudo-clases, de modo que los enlaces aparezcan en negro al principio, en azul al ser visitados y en verde con el ratón encima.

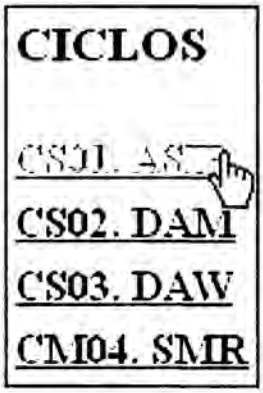


Figura 3.6: Selector pseudo-clase

3.3.10. Pseudo-elementos

Algunos pseudo-elementos son:

- **:first-line** (se aplica a la primera línea de un texto)
- **:first-letter** (se aplica a la primera letra de un texto)
- **:before** (sirve para generar texto antes del contenido de un elemento)
- **:after** (sirve para generar texto después del contenido de un elemento)

Ejemplo:

```
<html>
  <head>
    <style>
      h1:before {content: "Encabezado --- "; }
      h1:after {content: " --- Fin de encabezado"; }
      p:first-line {font-size: 20px; }
      p:first-letter {font-size: 3em; }
    </style>
  </head>
  <body>
    <h1>Texto1</h1>
    <p>
      Esta es la primera línea <br>
      Esta es la segunda línea <br>
      Esta es la tercera línea <br>
      Esta es la cuarta línea <br>
    </p>
  </body>
</html>
```

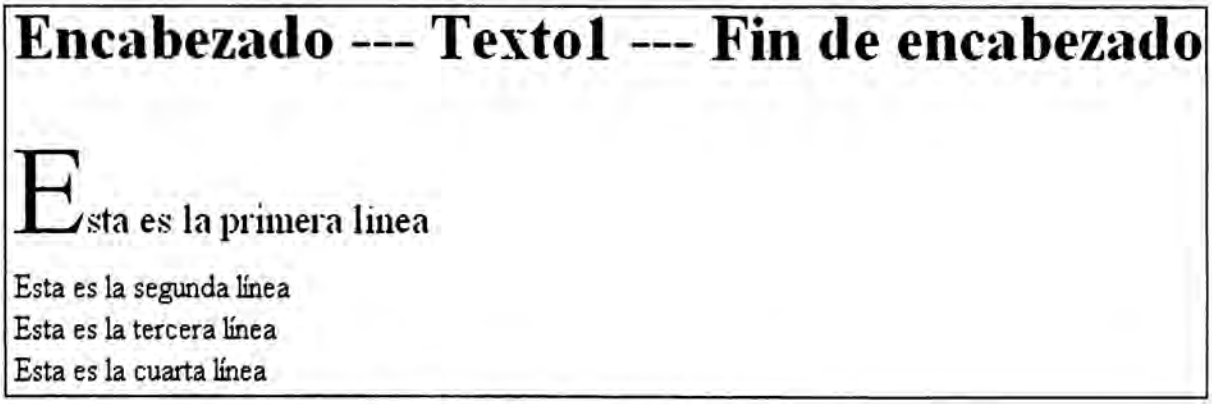


Figura 3.7: Selector pseudo-elemento

En CSS 3 se incorporan los siguientes selectores:

Selector general de hermanos, selectores de atributos (^=, $=, *=), :first-of-type, :last-of-type, only-of-type, only-child, :nth-child (n), :nth-last-child (n), :nth-of- type (n), :nth-last-of-type (n), :first-child :last-child, :root, :empty, :target, :enabled, :disabled, :checked, :not y ::selection.

A continuación, veremos el significado de los más relevantes.

- **Selector general de hermanos** (elemento1 ~ elemento2)

Selecciona los elemento2 precedidos por elemento1 y con el mismo padre (hermanos).

Ejemplo:

p ~ ul → Selecciona los elementos **<ul>** precedidos por **<p>**.

- **Selectores de atributos**:

· elemento[atributo^= "valor"] selecciona los elementos cuyo "atributo" empieza por "valor".

· elemento[atributo$="valor"] selecciona los elementos cuyo "atributo" termina por "valor".

· elemento[atributo*="valor"] selecciona los elementos cuyo "atributo" contiene "valor".

Ejemplo:

img[src^="android"] selecciona todas las imágenes cuyo atributo src empieza por "android".

- **Pseudoclases**:

· elemento:nth-child (n) selecciona los elementos que sea el hijo n-ésimo.

· elemento:nth-last-child (n) igual que el anterior pero se empieza a contar desde el último hijo.

· elemento:empty selecciona los elementos que no tenga ningún hijo ni contenido.

· elemento:first-child y elemento:last-child seleccionan los elementos que sean el primer hijo o el último hijo de su padre.

· elemento:nth-of-type (n) selecciona los elementos que sean el enésimo elemento de ese tipo.

· elemento:nth-last-of-type (n) idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

· :enabled , :disabled, :checked seleccionan campos de formulario según su estado.

· elemento:not (selector) selecciona elementos que no sean como "selector".

· ::selection selecciona el contenido marcado por el usuario.

Estos selectores son muy específicos por lo que se suelen utilizar con menos frecuencia que los anteriores de CSS 2.

3.4. Modelo de cajas

3.4.1. Fundamentos

El modelo de cajas es el elemento fundamental de CSS porque determina la composición de la página web. Todos los elementos que se insertan en el documento HTML se representan automáticamente mediante cajas rectangulares (excepto head).

En principio son cajas invisibles salvo que les asignemos bordes o color de fondo, para comprobar su apariencia podemos aplicarle bordes, como en la figura siguiente.

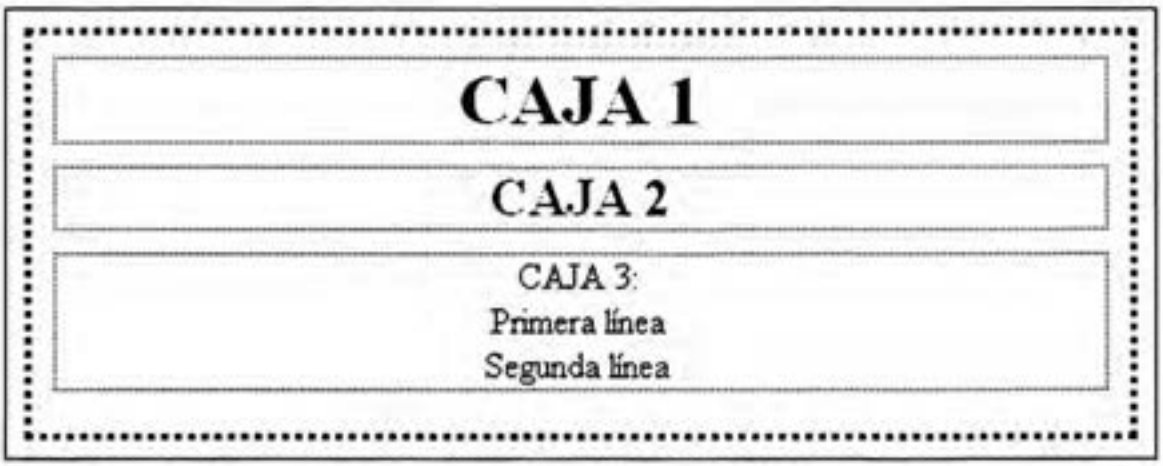


Figura 3.8: Modelo de cajas

La caja de puntos gruesos corresponde al elemento body, los puntos finos corresponden a los elementos h1, h2 y p. La caja exterior de borde sólido corresponde al elemento html.

```
<html>
 <head>
  <style>
   html {border: 1px solid;}
   body {border: 3px dotted; text-align: center;}
   p,h1,h2 {border: 1px dotted; text-align: center; margin: 10px;}
  </style>
 </head>
 <body>
  <h1>CAJA 1</h1>
  <h2>CAJA 2</h2>
  <p>
     CAJA 3: <br>
     Primera línea <br> Segunda línea <br> Tercera línea <br>
     Cuarta línea <br> Quinta línea
  </p>
 </body>
</html>
```

Las partes que componen cada caja y su orden de visualización son las siguientes:

1. **Contenido** (content): texto, imágenes, videos, listas, etc. del elemento.

2. **Relleno** (padding): espacio entre el contenido y el borde.

3. **Borde** (border): línea que encierra completamente el contenido y su relleno.

4. **Imagen de fondo** (background-image): imagen que se muestra por detrás del contenido y el espacio de relleno.

5. **Color de fondo** (background-color): color que se muestra por detrás del contenido y el espacio de relleno.

6. **Margen** (margin): espacio libre entre la caja y las cajas adyacentes.

Padding y margin son transparentes.

En el espacio ocupado por padding se muestra el color o imagen de fondo.

En el espacio ocupado por margin se muestra el color o imagen de fondo de su elemento padre.

Si ningún elemento padre tiene definido un color o imagen de fondo, se muestra el color o imagen de fondo de la página.

Si una caja define tanto un color como una imagen de fondo, la imagen tiene más prioridad (4 antes que 5) y se visualiza antes que el color.

Si la imagen de fondo no cubre totalmente la caja del elemento o si la imagen tiene zonas transparentes, entonces se visualiza el color de fondo en dichas zonas.

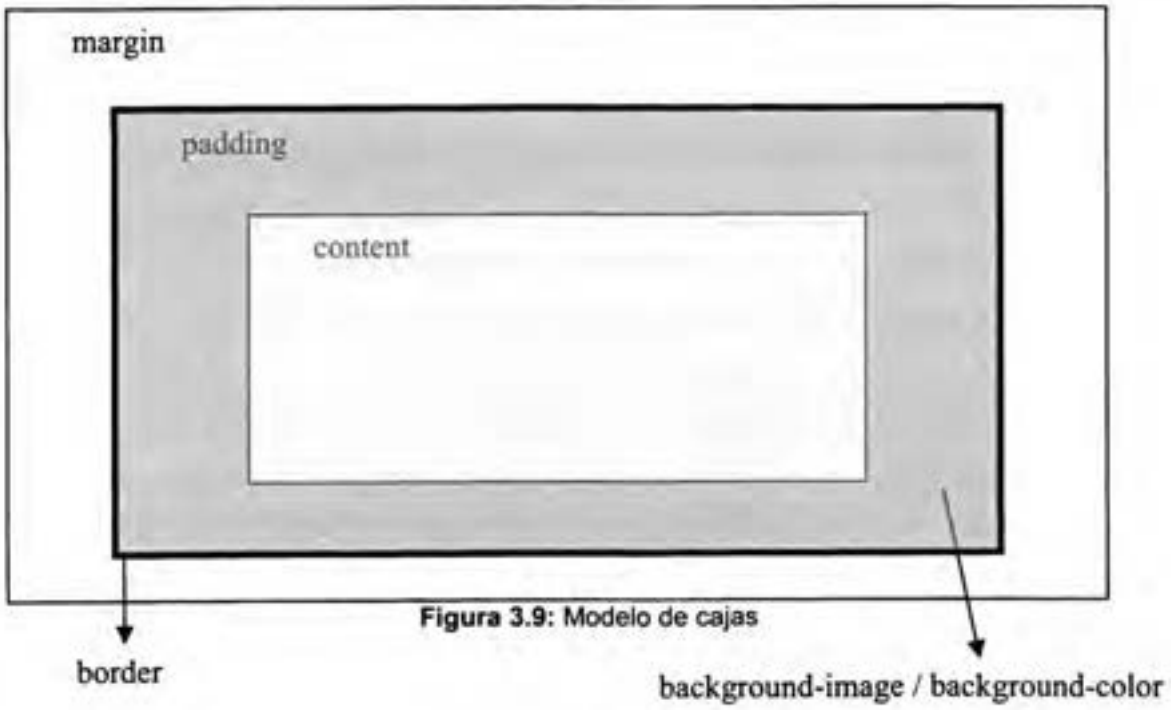


Figura 3.9: Modelo de cajas

3.4.2. Ancho, alto

Las propiedades que controlan el ancho y alto de una caja son width y height.

Propiedades: **width/height**

Significado: **ancho/alto de la caja**

Valores: **<numérico px> | <numérico %> | inherit | auto**

Aplicación: **elementos de bloque, imágenes y columnas de tabla**

<numérico px>: establece un ancho/alto fijo de la caja en píxeles.

<numérico %>: establece un ancho/alto fijo de la caja según un porcentaje del ancho/alto del elemento padre.

<inherit>: establece un ancho/alto fijo heredado del elemento padre.

<auto>: es el valor por defecto y determina que el navegador debe calcular el ancho/alto teniendo en cuenta el espacio disponible en la página.

Ejemplo:

```
#menu {width: 180px; height: 600px;}

<div id="menu">
  ...
</div>
```

3.4.3. Margin, Padding

Propiedades: **margin, margin-top, margin-right, margin-bottom, margin-left**

Significado: **margen, margen superior, derecho, inferior e izquierdo**

Valores: **<numérico px, em> | <numérico %> | inherit | auto**

Aplicación: **a todos los elementos** (excepto margin-top y margin-bottom)

Margin es una propiedad 'shorthand' que equivale a las otras cuatro a la vez.

El margen permite el centrado horizontal de una caja mediante → `margin: 0 auto;` que significa margen superior e inferior 0 y margen derecho e izquierdo automático.

Ejemplo:

```
#body {margin-top: 2em;}
#caja1 {margin: 10px;}
#centrado {margin: 0 auto;}
```

Propiedades: **padding, padding-top, padding-right, padding-bottom, padding-left**

Significado: **relleno, relleno superior, derecho, inferior e izquierdo**

Valores: **<numérico px, em> | <numérico %> | inherit | auto**

Aplicación: **a todos los elementos excepto cabeceras y pies de tablas**

Ejemplo:

```
#caja1 {padding: 5 10 5 10px;}
```

3.4.4. Bordes

Se pueden establecer tres tipos de propiedades a los bordes: el ancho, el color y el estilo.

· Ancho.

Propiedades: **border-width, border-top-width, border-right-width, border-bottom-width, border-left-width**

Significado: **ancho de borde, ancho de borde superior, ancho de borde derecho, ancho de borde inferior y ancho de borde izquierdo**

Valores: **<numérico px,em> | thin | medium | thick | inherit**

Aplicación: **a todos los elementos**

· Color.

Propiedades: **border-color, border-top-color, border-right-color, border-bottom-color, border-left-color**

Significado: color de borde, color de borde superior, color de borde derecho, color de borde inferior y color de borde izquierdo

Valores: **<color> | transparent | inherit**

Aplicación: **a todos los elementos**

· Estilo.

Propiedades: **border-style, border-top-style, border-right-style, border-bottom-style, border-left-style**

Significado: **estilo de borde, estilo de borde superior, estilo de borde derecho, estilo de borde inferior y estilo de borde izquierdo**

Valores: `none | hidden | dotted | dashed | solid | double |groove | ridge | inset | outset | inherit`

Aplicación: **a todos los elementos**

Si el estilo es none o hidden no aparece borde

3.4.5. Colores y fondos

Color

Básicamente hay dos formas de especificar los colores en CSS:

- Palabras reservadas y
- Números hexadecimales en el modelo RGB.

- Palabras reservadas:

Aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond. blue, blueviolet, brown, , burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue. cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkgrey. darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, ...

Por ejemplo:

```
p {color: yellow}
```

- Números en el modelo RGB:

Se trata de un modelo de colores aditivo, con el que se puede obtener cualquier color mediante la mezcla de los tres colores primarios (Red, Green, Blue) en la cantidad adecuada.

Existen varios formatos posibles para cada color: hexadecimal de 3 dígitos (rgb), hexadecimal de 6 dígitos (rrggbb), decimal con 3 valores, y porcentual con 3 valores.

Por ejemplo:

```
p {color: #ff0}
p {color: #ffff00}
p {color: rgb(255,255,0)}
p {color: rgb(100%,100%,0%)}
```

Los 4 formatos son equivalentes entre sí y equivalentes al color amarillo.

Para consultar todas las equivalencias:

http://www.w3.org/TR/2010/PR-css3-color-20101028/#svg-color

Fondo

El fondo solamente se visualiza en la zona ocupada por el contenido y su relleno (content – padding), ya que el borde tiene su propio color y los márgenes son transparentes.

Para establecer un color o imagen de fondo a la página entera, se debe aplicar fondo al elemento <body>. En este caso todos los elementos de la página se visualizan con el mismo fondo, a menos que algún elemento tenga el suyo propio.

CSS define cinco propiedades para establecer el fondo de cada elemento:

Propiedad: **background-color**

Significado: **color de fondo**

Valores: **<color> | transparent | inherit**

Aplicación: **a todos los elementos**

El valor inicial es transparent

Propiedad: **background-image**

Significado: **imagen de fondo**

Valores: **<url ("ruta a la imagen")> | none | inherit**

Aplicación: **a todos los elementos**

El valor inicial es none

Propiedad: **background-repeat**

Significado: **sirve para repetir la imagen de fondo hasta rellenar el elemento**

Valores: **repeat | repeat-x | repeat-y | no-repeat | inherit**

Aplicación: **a todos los elementos**

El valor inicial es repeat

Propiedad: **background-position**

Significado: **indica el desplazamiento de la imagen desde la esquina superior izquierda, mediante dos valores, el desplazamiento horizontal y el desplazamiento vertical**

Valores: **(<num%> | <medida> | left | center | right | inherit) y**

(<num%> | <medida> | top | center | bottom | inherit)

Aplicación: **a todos los elementos**

El valor inicial es 0% 0%. Si uno de los dos se omite, el valor por defecto es 50%.

Propiedad: **background-attachment**

Significado: **controla el comportamiento de la imagen de fondo respecto a la scroll-bar**

Valores: **fixed | scroll | inherit**

Aplicación: **a todos los elementos**

El valor inicial es scroll

Ejemplo de propiedades background:

Figura 3.10: Propiedades background.

Actividad 3.3:

Crea una página similar a la Figura 3.5 con la imagen que prefieras, utilizando las propiedades que aparecen en la figura sobre el elemento <body>.

Comprueba como permanece fija la imagen de fondo aunque utilicemos la barra de desplazamiento.

(Reduce la altura de la ventana si fuera necesario para que aparezca la barra)

Las propiedades background también pueden utilizarse en formularios, incluso dentro de cada campo del formulario, como veremos más adelante.

3.4.6. Posicionamiento

CSS tiene 5 formas de establecer la posición en pantalla de una caja:

Estático, Relativo, Absoluto, Fijo y Flotante

Para establecer la posición disponemos de las propiedades:

position, top, bottom, right, left y float

Propiedad: **position**

Significado: **establece la posición de la caja en pantalla**

Valores: **static | relative | absolute | fixed | inherit**

Aplicación: **a todos los elementos**

El valor por defecto es static

- static: la caja ignora las propiedades top, bottom, right y left

- relative: la caja se desplazará según los valores top, bottom, right y left

- absolute: la caja se desplazará según los valores top, bottom, right y left partiendo de su elemento contenedor

- fixed: como el absoluto pero la caja permanece inmóvil independientemente del resto de elementos y del movimiento de la página por parte del usuario

- inherit: heredado

Propiedades: **top, bottom, right, left**

Significado: **establece el desplazamiento vertical y horizontal de la caja.**

Valores: **<número px> | <número %> | auto | inherit**

Aplicación: **a todos los elementos**

El valor por defecto es auto

Propiedad: **float**

Significado: **establece la posición flotante de la caja**

Valores: **left | right | none | inherit**

Aplicación: **a todos los elementos**

El valor por defecto es none

· **Posicionamiento estático**. {position: static}

Se trata del posicionamiento por defecto que utilizan todos los navegadores.

Si los elementos a posicionar son de bloque, como un párrafo, las cajas se colocarán verticalmente una debajo de otra, dentro del elemento contenedor.

Si los elementos a posicionar son de línea, como un hipervínculo, las cajas se colocarán una junto a otra horizontalmente dentro del elemento de bloque.

Si las cajas no caben en una línea, entonces ocuparán la siguiente, etc. y si caben en una línea, se pueden alinear horizontalmente.

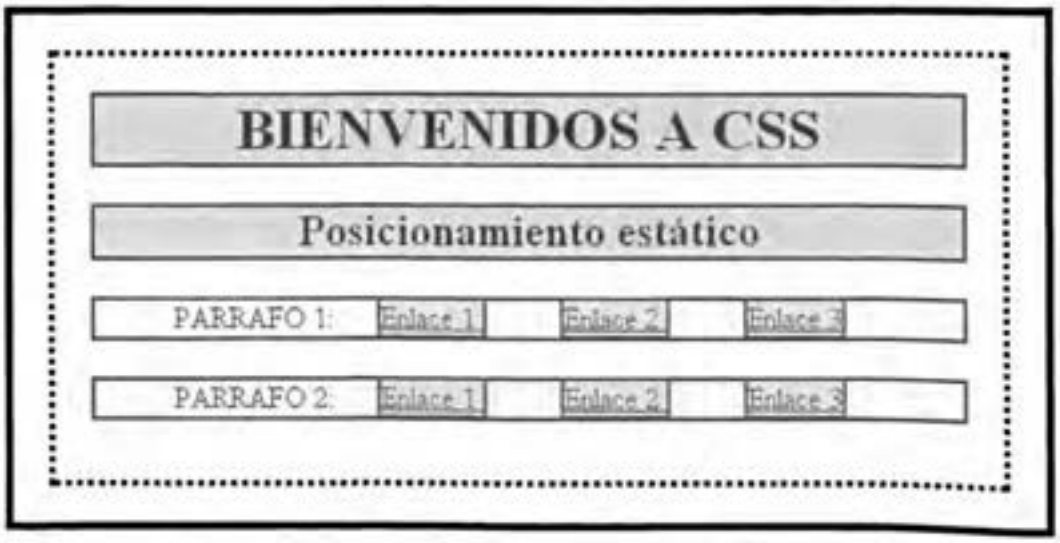


Figura 3.11: Posicionamiento estático

```
<style>
   * {border: 2px solid; text-align: center; margin: 20px;}
   html {border: 4px solid;}
   body {border: dotted;}
   h1,h2 {background-color: pink;}
   a {background-color: yellow;}
</style>

<body>
 <h1>BIENVENIDOS A CSS</h1>
 <h2>Posicionamiento Estático</h2>
 <p> PARRAFO 1:
     <a href="#"> Enlace 1 </a> <a href="#"> Enlace 2 </a>
     <a href="#"> Enlace 3 </a>
 </p>
 <p> PARRAFO 2:
     <a href="#"> Enlace 1 </a><a href="#"> Enlace 2 </a>
     <a href="#"> Enlace 3 </a>
 </p>
<body>
```

· **Posicionamiento relativo**. {position: relative}
Consiste en desplazar la caja una cantidad respecto del posicionamiento estático.

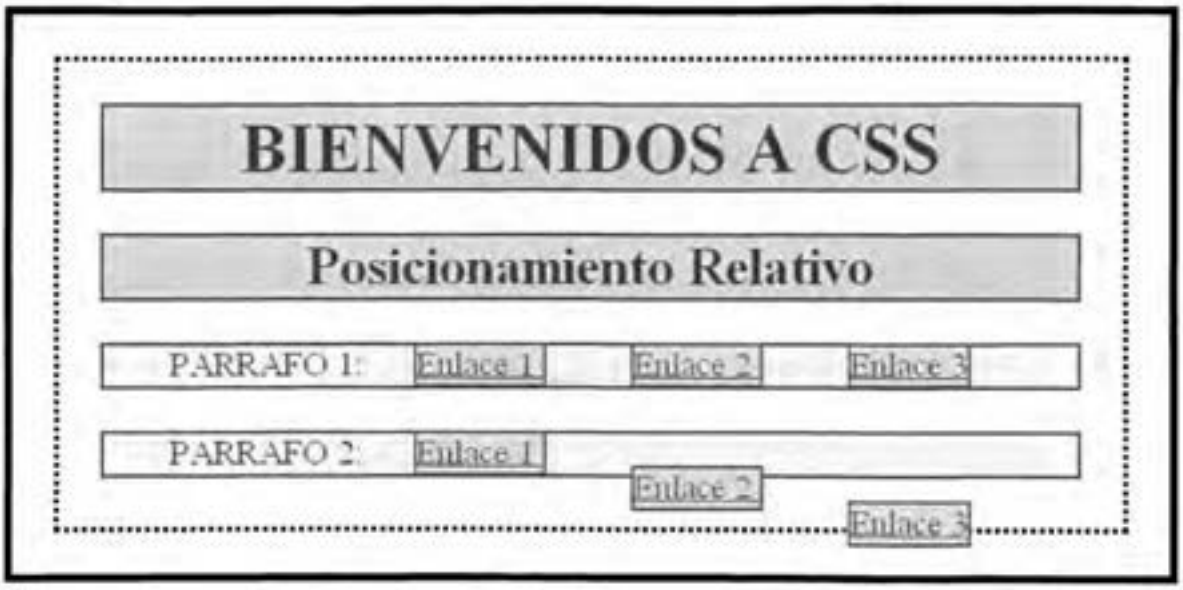


Figura 3.12: Posicionamiento relativo

```
<style>
  * {border: 2px solid; text-align: center; margin: 20px;}
  html {border: 4px solid;}
  body {border: dotted;}
  h1,h2 {background-color: pink;}
  a {background-color: yellow;}
  a.abajo {position: relative; top: 1em;}
  a.mas_abajo {position: relative; top: 2em;}
</style>

<body>
 <h1>BIENVENIDOS A CSS</h1>
 <h2>Posicionamiento Relativo</h2>
 <p>
    PARRAFO 1:
    <a href="#"> Enlace 1 </a>
    <a href="#"> Enlace 2 </a>
    <a href="#"> Enlace 3 </a>
 </p>
 <p>
    PARRAFO 2:
    <a href="#"> Enlace 1 </a>
    <a href="#" class="abajo"> Enlace 2 </a>
    <a href="#" class="mas_abajo"> Enlace 3 </a>
 </p>
</body>
```

Como se puede observar, la caja "Enlace 2" se ha desplazado 1em hacia abajo desde su posición normal (static), y la caja "Enlace 3" se ha desplazado 2em.

· **Posicionamiento absoluto**. {position: absolute}

Consiste en desplazar la caja una cantidad respecto de su elemento contenedor.

La caja sale del flujo normal de la página y el resto de cajas ocupan su lugar.

Para el desplazamiento se usan top, bottom, left right y se toma como origen de coordenadas la esquina superior izquierda del primer contenedor que no sea static.

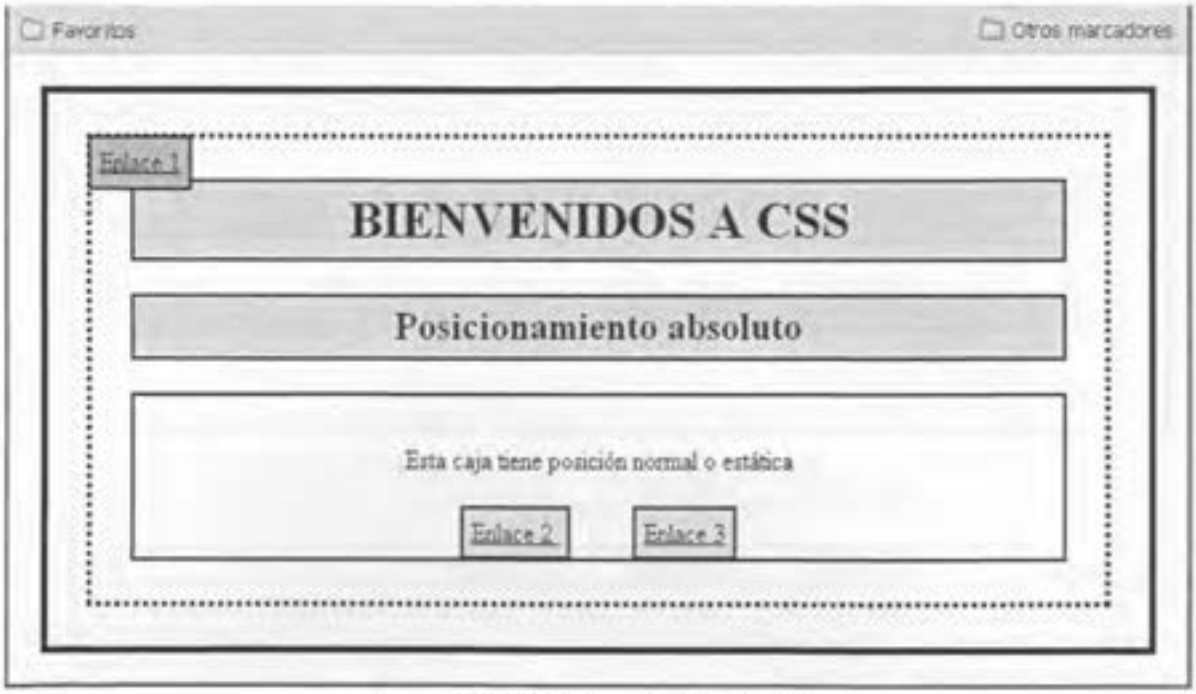


Figura 3.13: Posicionamiento absoluto

```
<style>
   * {text-align: center; margin: 20px; padding: 5px;}
   html {border: 4px solid;}
   body {border: dotted;}
   h1,h2 {background-color: pink; border: 2px solid;}
   div {border: 2px solid;}
   a {background-color: yellow; border: 2px solid;}
   a.absoluto {background-color: orange;
               position: absolute; top: 30px; left: 30px;}
</style>

<body>
  <h1>BIENVENIDOS A CSS</h1>
  <h2>Posicionamiento absoluto</h2>
  <div>
    <p>Esta caja tiene posición normal o estática</p>
    <a href="#" class="absoluto"> Enlace 1 </a>
    <a href="#"> Enlace 2 </a>
    <a href="#"> Enlace 3 </a>
 </div>
</body>
```

En este caso el "Enlace1" se desplaza 30px desde arriba y desde la izquierda, partiendo desde la esquina superior izquierda de la página, porque ** ** tiene posición normal y el siguiente es **<body>** (observar que además hay 20px de margen).

Las cajas "Enlace2" y "Enlace3" ocupan el hueco dejado por la caja "Enlace1".

Veamos otro ejemplo de posicionamiento absoluto.

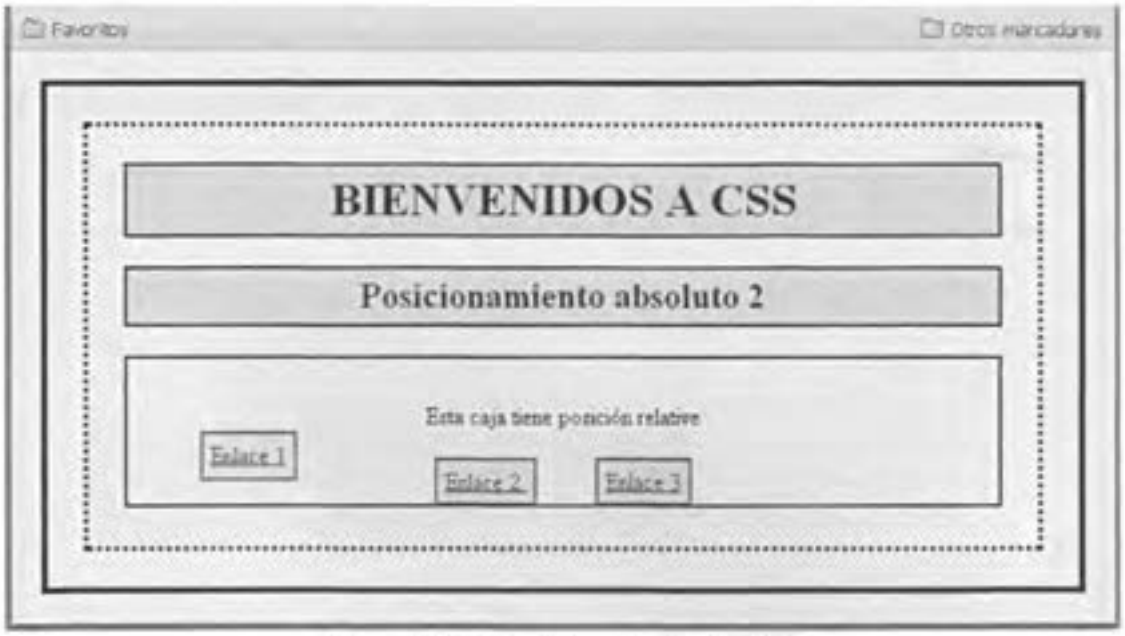


Figura 3.14: Posicionamiento absoluto

```
<style>
  * {text-align: center; margin: 20px; padding: 5px;}
  html {border: 4px solid;}
  body {border: dotted;}
  h1,h2 {background-color: pink; border: 2px solid;}
  div {position: relative; border: 2px solid;}
  a {background-color: yellow; border: 2px solid;}
  a.absoluto {background-color: orange;
              position: absolute; top: 30px; left: 30px;}
</style>
```

En este caso el "Enlace1" se desplaza 30px desde arriba y desde la izquierda, pero partiendo de su caja contenedora, que es la caja ** **, porque es la primera caja con posicionamiento no estático (tiene posición relative).

· **Posicionamiento fijo**. {position: fixed}

Similar al posicionamiento absoluto, pero cuando el usuario desplaza la página las cajas se mantienen fijas en el mismo lugar.

Esto resulta útil para encabezados o pies de página que se mantienen inalterables a lo largo de un documento extenso.

También es posible aplicar el posicionamiento fixed a un menú de navegación lateral, que necesitemos esté siempre presente, para los famosos widgets sociales, o un pequeño menú de ayuda al usuario.

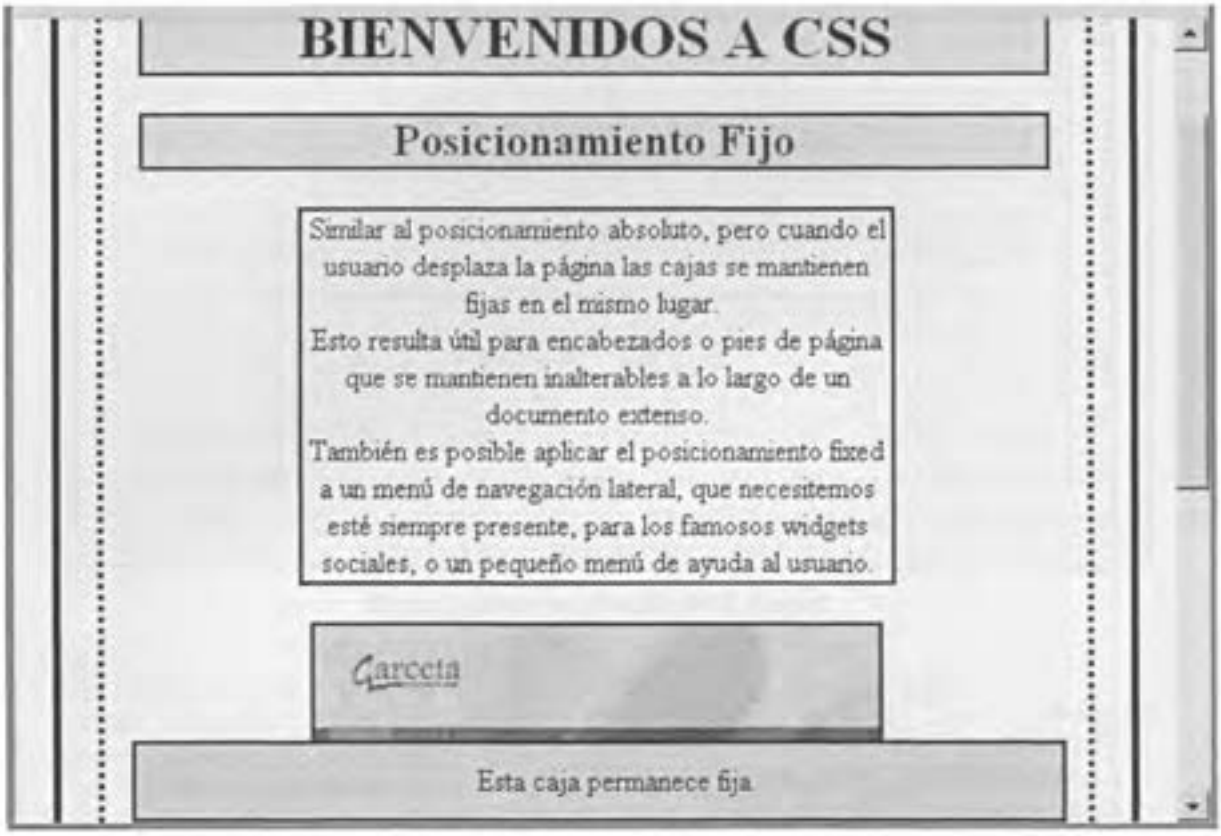


Figura 3.15: Posicionamiento fijo

```
<style>
  * {border: 2px solid; text-align: center; margin: 20px;}
  html {border: 4px solid;}
  body {border: dotted;}
  h1,h2 {background-color: pink;}
  p {width: 60%; position: relative; left: 20%; margin: 0;}
  .fijo {position: fixed; bottom:0; width: 80%; height: 40px;
         margin: 0; left: 10%; background-color:#CCC}
  .fijo p {border: 0; margin: 10px;}
</style>
```

Actividad 3.4:

Escribe el código html que falta para obtener la figura 3.10.

Comprueba como la caja inferior permanece fija aunque utilices la barra de desplazamiento.

· **Posicionamiento flotante**. {float: left | right | inherit | none}

- Desplaza las cajas todo lo posible hacia la izquierda o hacia la derecha de la línea en la que se encuentran.

- El resto de cajas ocupan el lugar dejado por la caja flotante.

- No se produce solapamiento de cajas porque la propiedad float tiene en cuenta el resto de cajas flotantes.

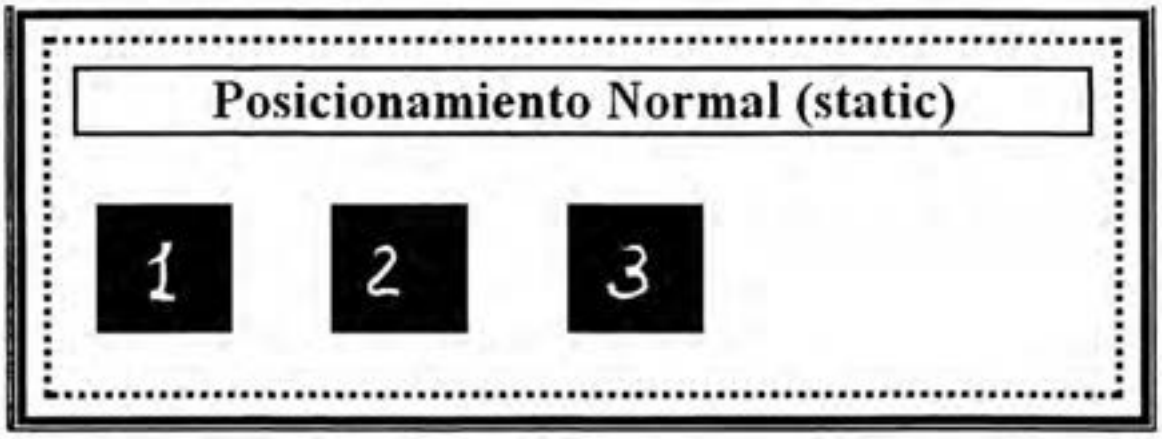


Figura 3.16: Posicionamiento normal

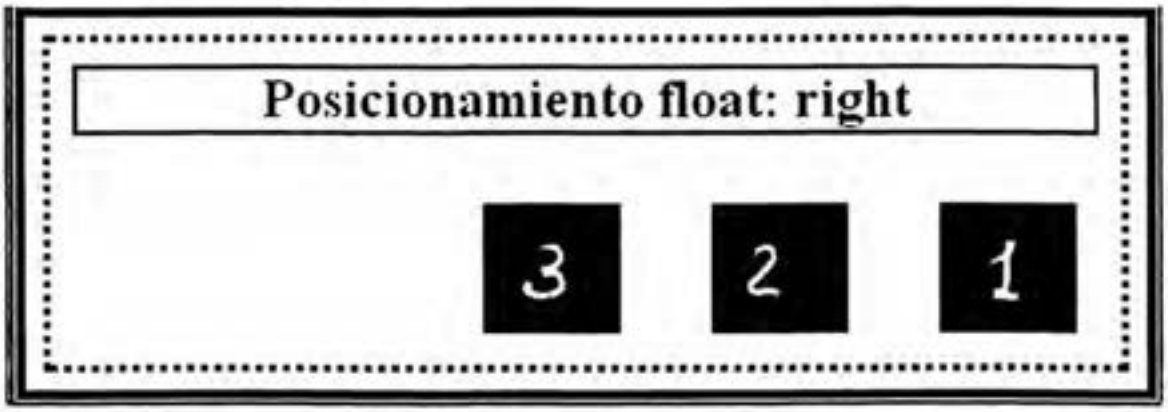


Figura 3.17: Posicionamiento float: right

Con `float: right` las imágenes se desplazan todo lo posible hacia la derecha de su posición original pero respetando al resto de cajas.

```
<style>
     html {border: 4px solid;}
     body {border: 2px dotted;}
        h2 {background-color: pink; border: 2px solid;
        margin: 10px;text-align: center;}
     img {width: 60px; margin: 20px; float: right;}
</style>
```

- Los elementos situados alrededor de la caja flotante se adaptan a la posición de la caja, lo que hace que sea idónea para la combinación de texto con imágenes.

Figura 3.18: Posicionamiento float: left

```
<html>
 <head>
   <title> Posicionamiento flotante</title>
   <style>
    #principal {border: 2px solid; width: 400px; font-size: large;}
    img {float: left;}
    p {margin: 5px;}
   </style>
 </head>
 <body>
  <h1>Posicionamiento flotante</h1>
  <div id="principal">
    <img src="salude.jpg">
    <p> En el posicionamiento flotante, el texto fluye alrededor de
       las imágenes hasta que se acabe el texto o aparezca la
       propiedad clear. En el posicionamiento flotante, el texto
       fluye alrededor de las imágenes hasta que se acabe el texto o
       aparezca la propiedad clear.
    </p>
  </div>
 </body>
</html>
```

- La propiedad clear permite evitar el comportamiento del posicionamiento flotante.

En el valor se establece el lado que no debe ser adyacente a ninguna caja flotante.

```
{ clear: none | left | right | both | inherit }
```

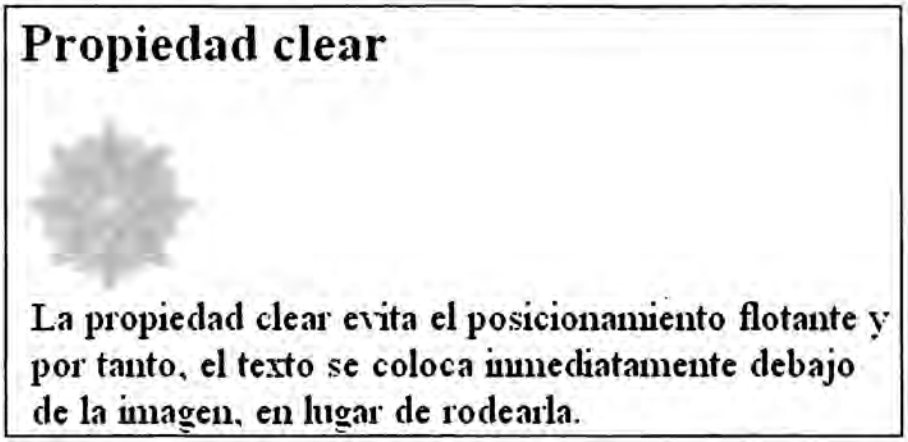


Figura 3.19: Propiedad clear

3.4.7. Visualización

En CSS hay cuatro propiedades que controlan la visualización de las cajas:

display, visibility, z-index y overflow.

Propiedad: **display**

Valores:

inline | block | none | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | inherit

Significado:

- El valor **inline** hace que cualquier elemento html se muestre como elemento en línea.

Se suele utilizar en las listas que se quieren mostrar horizontalmente.

- El valor **block** hace que cualquier elemento html se muestre como elemento de bloque, es decir ocupa todo el ancho de línea disponible.

Se suele utilizar para los enlaces que forman el menú de navegación.

Vamos a combinar las propiedades **background-color** y **display: block**, con el selector **a:hover** para crear un menú de navegación.

Figura 3.20: Menú de navegación

```
<style>
  body {background-color: #996;}
  a:link {color: #006699; padding: 2px 0px 2px 10px;
     border-top: 1px solid #cccccc; display: block;}
  a:hover {color: #006699; border-top: 1px solid #cccccc;
     background-color: #dddddd; padding: 2px 0px 2px 10px;}
  h3 {color: #334d55; padding: 10px 0px 0px 10px;}
  #navBar {width: 220px; height: auto; margin-top: 10px;
     padding: 0px; background-color: #eeeeee; font-family: arial;
     border-bottom: 1px solid #ccc; border-right: 1px solid #ccc;
     font-weight: bold;}
</style>
...
<body>
  <div id="navBar">
    <h3> MODULOS </h3>
    <a href="asir.html">CS01. ASIR</a>
    <a href="dam.html">CS02. DAM</a>
    <a href="daw.html">CS03. DAW</a>
    <a href="smr.html">CM04. SMR</a>
  </div>
</body>
```

La propiedad display: block nos permite hacer que un elemento en línea como **<a>** se comporte como un elemento de bloque y por tanto, ocupe toda la línea en la que se encuentra.

Mediante a:hover y background-color: #ddd conseguimos que la línea cambie de color cuando situamos el ratón encima, dando la seguridad al usuario de que ha seleccionado la opción deseada.

- El valor **none** permite ocultar completamente un elemento haciendo que desaparezca de la página, el resto de elementos de la página se mueven para ocupar su lugar.

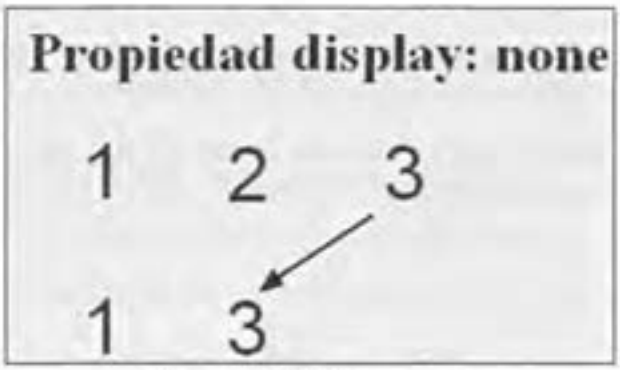


Figura 3.21: Display: none

```
<html>
 <head>
   <title>Propiedad display</title>
   <style>
    * {margin: 5px;}
    img {float: left;}
    #secundario {clear: left;}
    #dos {display: none;}
   </style>
 </head>
 <body>
  <h1>Propiedad display: none</h1>
  <div id="principal">
    <img src="uno.jpg">
    <img src="dos.jpg">
    <img src="tres.jpg">
  </div>
  <div id="secundario">
    <img src="uno.jpg">
    <img id="dos" src="dos.jpg">
    <img src="tres.jpg">
  </div>
 </body>
</html>
```

Podemos observar que la imagen "dos.jpg" no se muestra en la imagen, a pesar de que también está en el bloque **`<div id="secundario">`**, esto se debe a la última regla CSS: #dos {display: none;}.

Si **`display`** vale **none**, se ignoran las propiedades **`float`** y **`position`** y la caja no se muestra en la página.

Propiedad: **`visibility`**

Valores: **`visible | hidden | collapse | inherit`**

Significado:

- **`visible`**, muestra la caja, es la opción por defecto

- **`hidden`**, oculta la caja, pero deja el espacio de modo que el resto de cajas no cambian su posición

- **`collapse`**, se utiliza para filas y columnas de una tabla, y permite que se visualicen otros contenidos en la fila o columna indicada

- **`inherit`**, heredado

En la imagen se observa que el número 2 está oculto, pero el resto de cajas no cambian su posición.

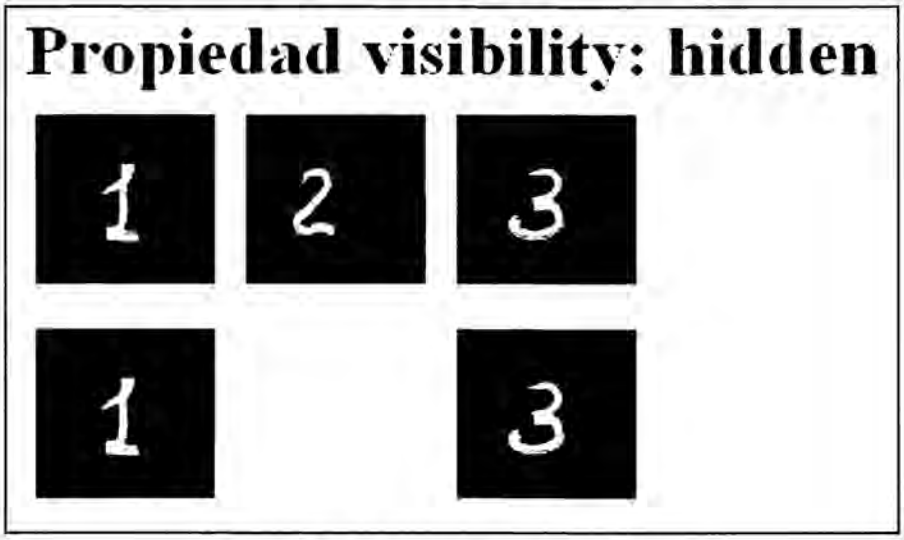


Figura 3.22: Propiedad visibility: hidden

Propiedad: **`overflow`**

Valores: **`visible | hidden | scroll | auto | inherit`**

Significado: **Sirve para controlar el desbordamiento del texto en una caja**

- **`visible`**, permite ver el contenido que se desborda, es la opción por defecto

- **`hidden`**, oculta el texto que se desborda

- **`scroll`**, mantiene el tamaño de la caja y además crea barras de desplazamiento horizontal y vertical

- **`auto`**, mantiene el tamaño de la caja y aparecen las barras solo si es necesario para ver todo el contenido

- **`inherit`**, heredado

Esta propiedad también sirve para solucionar un problema relacionado con las capas ** **.

Cuando construimos un ** ** podemos asignarle tamaño mediante width y height, si posteriormente añadimos contenido ya sea texto o imágenes que superan el tamaño establecido, en principio, suele ocurrir que la capa aumenta inmediatamente de tamaño, produciendo un efecto indeseado. Esto puede evitarse mediante el uso de `overflow: auto`.

Actividad 3.5:

Crea una caja y asigna ancho y alto, rellénala con texto e imagen.

Modifica el tamaño de la ventana y observa el comportamiento.

Mediante la propiedad overflow incluye barras para que el texto no se desborde.

Propiedad: **z-index**

Valores: **auto | <n° entero> | inherit**

Significado: **establece la posición de la caja respecto al eje z, es decir, la profundidad respecto al documento mediante un número entero.**

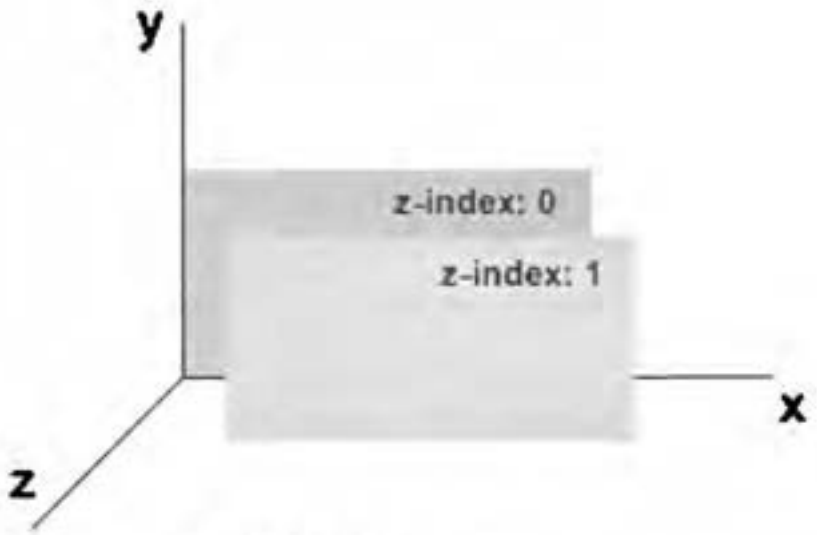


Figura 3.23: Propiedad z-index

Vamos a construir 4 cajas con diferente valor z-index y superpuestas en forma de escalera. Al pasar el ratón por encima de una caja debe convertirse en la caja más cercana al usuario.

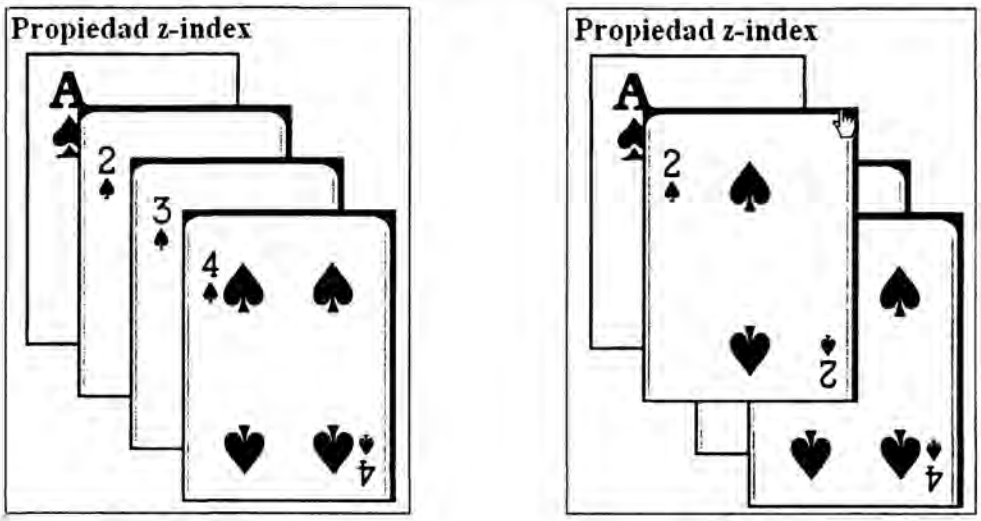


Figura 3.24: Propiedad z-index

```
<html>
<head>
 <title>Propiedad z-index</title>
 <style>
   #uno {z-index:1; position: absolute; top: 40px; left:20px;
         background-image: url("as_picas.jpg"); }
   #dos  {z-index:2; position: absolute; top: 80px; left:60px;
          background-image: url("dos_picas.jpg");}
   #tres  {z-index:3; position: absolute; top: 120px; left:100px;
           background-image: url("tres_picas.jpg");}
   #cuatro {z-index:4; position: absolute; top: 160px; left:140px;
            background-image: url("cuatro_picas.jpg");}

   div {width:160px; height:220px; border: 2px solid;}

   #uno:hover, #dos:hover, #tres:hover, #cuatro:hover {z-index:10;}
 </style>
</head>

<body>

   <h2>Propiedad z-index</h2>
   <div id="uno"> </div>
   <div id="dos"> </div>
   <div id="tres"> </div>
   <div id="cuatro"> </div>
 </body>

</html>
```

En este ejemplo, las cartas de la baraja tienen inicialmente un z-index bajo (1, 2, 3, 4 respectivamente), pero al pasar el ratón por encima le asignamos z-index: 10, por lo tanto, la carta seleccionada pasa a primer plano.

3.5. Texto

Las principales propiedades relacionadas con el texto se pueden ver en la siguiente tabla:

Propiedad: valor	Significado
color: `<color> | inherit`	Color del texto
font-family: `<fuente> | inherit`	Tipo de fuente de letra
font-size: `<absoluto> | <relativo> | <em> | <%> | inherit`	Tamaño de letra
font-weight: `normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit`	Grosor de letra
font-style: `normal | italic | oblique | inherit`	Estilo de letra
font-variant: `normal | small-caps | inherit`	Mayúsculas en pequeño
text-align: `left | right | center | justify | inherit`	Alineación horizontal
line-height: `normal | <numero> | <medida> | <porcentaje> | inherit`	Interlineado
text-decoration: `none | ( underline || overline || line-through || blink ) | inherit`	Decoración
text-transform: `capitalize | uppercase | lowercase | none | inherit`	Transformación
text-shadow: `none | h-shadow | v-shadow | blur | color`	Sombreado del texto
vertical-align: `baseline | sub | super | top | text-top | middle | bottom | text-bottom | <porcentaje> | <medida> | inherit`	Alineación vertical
text-indent: `<medida> | <porcentaje> | inherit`	Tabula las primeras líneas
letter-spacing: `normal | <medida> | inherit`	Espaciado entre letras
white-space: `normal | nowrap | pre | pre-line | pre-wrap | inherit`	Tratamiento de los espacios en blanco
word-spacing: `normal | <medida> | inherit`	Espaciado entre palabras

Tabla 3.2: Propiedades CSS para el texto

Actividad 3.6:

Modifica el menú anterior para que los enlaces no estén subrayados hasta que los seleccionamos con el ratón (propiedad text-decoration).

Figura 3.25: Enlaces sin subrayar

Para incluir tipos de fuente especiales, se puede agregar el archivo de fuentes en la carpeta apropiada del sistema (C:\Windows\fonts normalmente). Pero en el caso de contratar un hosting externo no tendremos acceso a dicha carpeta, ni falta que hace, puesto que con CSS podemos importar los archivos de fuentes deseados mediante la regla @font-face.

Formato de uso:

```
@font-face {font-family: mifuente; src: url ('archivo de fuentes');}
```

Ejemplo:

Figura 3.26: Tipos de Fuentes especiales

```
<html>
 <head>
  <style type="text/css">
    @font-face {font-family: chopin; src: url(ChopinScript.otf);}
    @font-face {font-family: allstar; src: url(AllStar.ttf);}
    .chopin {font-family:chopin;}
    .allstar {font-family:allstar;}
    p {font-size: 200%;}
  </style>
 </head>
 <body>
   <h1>Cambiar la tipografía con CSS3:</h1>
   <p class="chopin">Este texto utiliza la fuente Chopin</p>
   <p class="allstar">Este texto utiliza la fuente Allstar</p>
 </body>
</html>
```

En http://www.fonts2u.com/ dispones de fuentes bajo licencia Freeware.

3.6. Listas

Propiedad: valor	Significado
list-style-type: `disc | circle | square | decimal |decimal-leading-zero | lower-roman | upper-roman | lower-greek | lower-latin | upper-latin | armenian | georgian | lower-alpha | upper-alpha | none | inherit`	Viñeta usada para los elementos de la lista
list-style-position: `inside | outside | inherit`	Posición de la viñeta
list-style-image: `<url> | none | inherit`	Sustituye la viñeta por una imagen
List-style: (propiedad shorthand)	Combina las anteriores

Tabla 3.3: Propiedades CSS para listas

La figura siguiente es un ejemplo de lista con list-style-type: circle.

LISTA DE LA COMPRA SALUDABLE:

- Caja de leche desnatada 6L
- Menestra de verduras 1kg
- Dorada en filetes 450g

Figura 3.27: Listas con CSS

Vamos a construir un menú horizontal utilizando una lista, para lo cual, aplicaremos la propiedad display: inline, que visualiza los elementos en línea y sin viñeta.

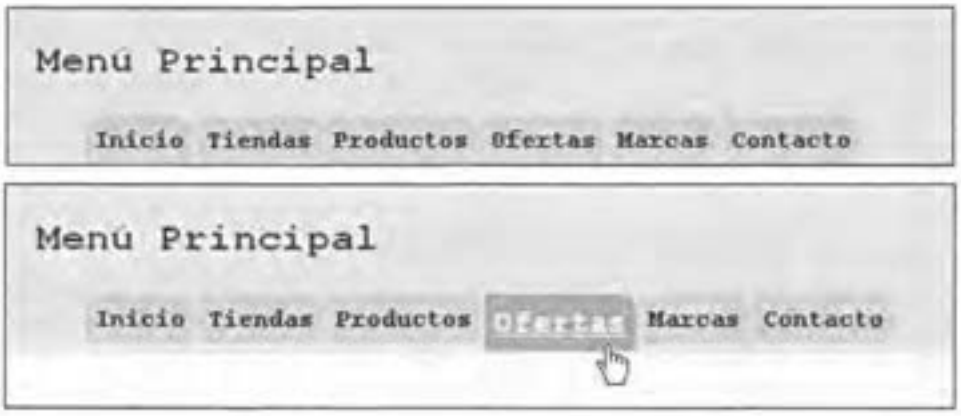


Figura 3.28: Menú horizontal

```
<style type="text/css">
    body {margin:5; padding:0; font: bold 14px Courier;}

    h2 {color: #000; padding: 0px 0px 0px 15px;}

    #menu_h {float:left; width:100%; background:#eee;}

    #menu_h li {display:inline; margin:0; padding:0;}

    #menu_h a  {float:left; display:block; margin:1px; padding:5px;
               text-decoration:none; color:#000;
                border-top-right-radius: 8px;
                background-color: #ccc;}
    #menu_h a:hover {color:#FFF; font-size: 1.2em;
                     background-color: #999;
                     text-decoration: underline;}
</style>
```

En principio los elementos de una lista se visualizan como elementos de bloque, es decir, saltando de línea como en la Figura 3.27. En este caso queremos un menú horizontal, por eso aplicamos display: inline a toda la lista, es decir, al selector #menu h li. Después redondeamos la esquina derecha de cada enlace para que tenga apariencia de pestaña, mediante la propiedad border-radius. Por último cambiamos tamaño, fondo y subrayamos al pasar el ratón por encima.

Actividad 3.7:

Genera el código html necesario para construir el menú horizontal de la figura anterior.

3.7. Tablas

Propiedad: valor	Significado
border-collapse: `collapse` | `separate` | `inherit`	Determina la fusión de bordes
border-spacing: `<medida> <medida>` | `inherit`	Separación de bordes horizontal y vertical
empty-cells: `show` | `hide` | `inherit`	Celdas vacías
caption-side: `top` | `bottom` | `inherit`	Posición del título

Tabla 3.4: Propiedades CSS para tablas

Vamos a construir una tabla con 3 colores, 1 color para la fila head y el resto de filas intercambiando el color blanco y verde o gris muy claro.

Empresa	Contacto	País
Mercedes Benz	Maria Naiditsch	Alemania
Volvo	Cristina Carlsen	Suecia
Seat	Francisco Valle	España
Rover	Elena Short	Reino Unido
Toyota	Hikaru Kamikaze	Japón
Ferrari	Danielle Rovelli	Italia
Cadillac	Simon Says	USA
Renault	Marie Curie	Francia

Figura 3.29: Tabla con CSS

```
<style type="text/css">
    #clientes {font-family:Arial, Helvetica; text-align: center;
               width:100%; border-collapse: collapse;
          }
    #clientes th {font-size:1.2em; border: 1px solid #98b;
                  padding-top:5px; padding-bottom:4px;
                  background-color:#AC4; color:#fff;
          }
    #clientes td {font-size:1em; border:1px solid #98b;
                  padding:4px 7px 2px 7px;
          }
    #clientes .alt {color:#000; background-color:#EFD;
          }
</style>
```

3.8. Formularios

Todas las propiedades vistas anteriormente se pueden aplicar a los elementos de un formulario para mejorar su aspecto y facilitar la entrada de datos por parte del usuario.

Vamos a crear un formulario con imagen de fondo y con ayudas visuales por ejemplo, fondo amarillo y borde más ancho para el cuadro de texto activo (propiedad :focus).

Figura 3.30: Formularios con CSS

```
<style type="text/css">
  form { width:390px; heigth:900px;  margin:10;  padding: 80px;
         background-image: url('imagenes/form-image.jpg');
         text-align: center;
         font-family: Lucida Calligraphy;}
  fieldset { margin: 20px; padding: 0 20px 0 20px;
             border:0px solid #000; text-align: center;}
 .titulo { font-size: 22; font-weight: bold;}
  p { margin: 0px; margin-bottom: 5px; }

  table { width: 370px; font-size: 11px;}
  label { margin-right: 7px; font-weight: bold; font-size: 12px;
          float: right;}
  input { border-radius: 8px; }

  #nombre {background-image: url('imagenes/nombre.jpg');
           background-repeat: no-repeat;}
  #ape {background-image: url('imagenes/nombre.jpg');
        background-repeat: no-repeat;}
  #tele {background-image: url('imagenes/telefono.jpg');
         background-repeat: no-repeat;}
  #pass {background-image: url('imagenes/pass.jpg');
         background-repeat:no-repeat;}
  #email {background-image: url('imagenes/email.jpg');
          background-repeat: no-repeat;}
  #acepto {float: right; margin-right: 15px;}
  #enviar {width: 130px; height: 35px; font-weight: bold;}
  :focus {background-color: yellow; border: 2px solid;}
</style>

<fieldset id="personales">
   <table>
      <tr><td><label for="nombre">Nombre</label></td>
          <td><input type="text" id="nombre" /></td>
      </tr>
      <tr><td><label for="ape">Apellidos</label></td>
          <td><input type="text" id="ape" />
      </tr>
      <tr><td><label for="tele">Teléfono</label></td>
          <td><input type="text" id="tele" /></td>
      </tr>
      <tr><td><label for="gen">Género</label></td>
          <td><input type="radio" id="gen"
                 value="hombre">Hombre</input>
              <input type="radio" id="gen"
                 value="mujer">Mujer</input></td>
      </tr>
      <tr><td><label for="email">E-mail</label></td>
          <td><input type="text" id="email" /></td>
      </tr>
      ...........................
   </table>
</fieldset>
```

3.9. Layout

El "layout" es el diseño o estructura general de la página creado mediante capas ** **. Podemos clasificar los diseños según el número de columnas, según donde coloquemos el menú de navegación, según el ancho sea fijo o variable, etc.

Layout líquido es aquel que tiene el ancho de las capas variable (usando porcentajes), para que se adapte a las medidas de la pantalla, de forma similar a como un líquido se adapta al contenedor donde se encuentra.

Vamos a poner como ejemplo un diseño de página líquido a 2 columnas y navegación izquierda. Las capas que vamos a utilizar para el diseño son las siguientes:

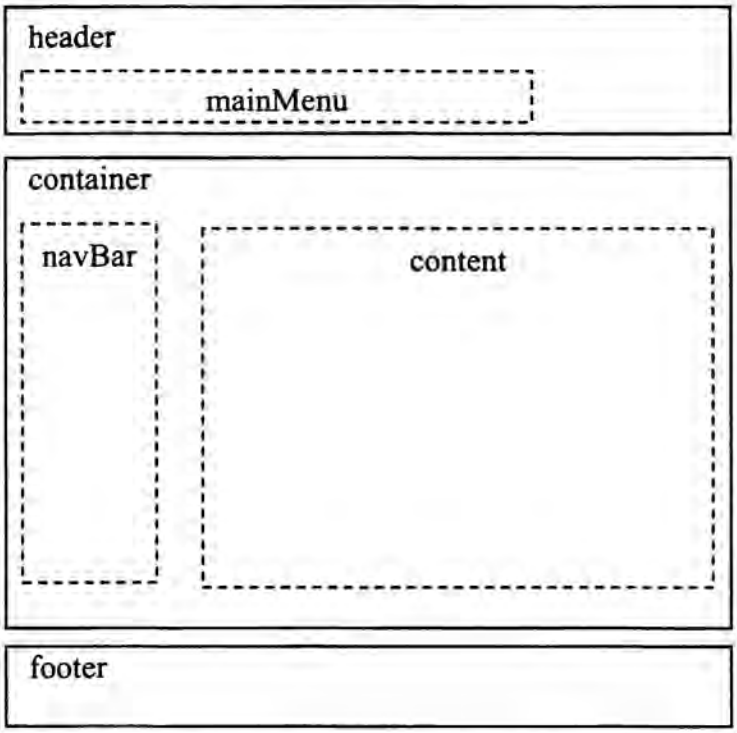


Figura 3.31: Capas usadas en un diseño a 2 columnas

Características principales:

- Las capas principales header, container y footer tienen las propiedades width: 100%. margin: 0, padding: 0 y overflow: auto.

Estas capas no deben tener margen ni relleno porque sobrepasarían el límite de 100% y entonces aparecería la barra de desplazamiento horizontal, que es preferible evitar.

- Las capas internas "navBAR" y "content" tienen las propiedades `float: left` y `margin: 1%`, (se podría añadir otra capa de la misma forma para un diseño a 3 columnas).

- El elemento **<body>** tiene la propiedad `min-width: 600px` para evitar que la estructura se descomponga cuando la ventana del navegador se reduce por debajo de esa medida.

- Se pueden añadir capas internas, tantas como queramos, para distribuir contenidos.

```
/* layout.css */

/* Estilos en HTML ------------------------  */

html{
      margin: 0px;
}

body{
      margin: 0px;
      padding: 0px;
      width: 100%;
      min-width: 650px;
      background-color: #fff ;
      border-style: none;
      font-family: Arial;
      color: #333;
}

h1, h2, h3{
      font-family: Arial;
}

a:link, a:visited {
      color: #069;
      text-decoration: none;
}

a:hover {
      color: #069;
      text-decoration: underline;
}

/* Estilos de Layout Divs ------------- */

#header{
      width: 100%;
      margin: 0px;
      padding: 0;
      overflow: auto;
}

#container{
      width: 100%;
      margin: 0px;
      padding: 0;
      overflow: auto;
}

#footer{
      width: 100%;
      height: 100px;
      margin: 0px;
      padding: 0;
```

```
        background-color: #eee;
        color: #aaa;
        font-size: 70%;
        font-weight: bold;
        border-top: 1px solid #000;
        overflow: auto;
}

#navBar{
        float: left;
        width: 18%;
        margin-top: 20px;
        margin: 1%;
        background-color: #ddd;
}

#content{
        float:left;
        width: 78%;
        margin: 1%;
        overflow: hidden;
}

/* Estilos de #header ------------------------- */

#header h1{
        padding-left: 1%;
}

/* Estilos de #main_menu ---------------------- */

#main_menu{
        padding-bottom: 5px;
        padding-left: 1%;
        border-bottom: 1px solid #ccc;
        overflow: auto;
}

#main_menu ul {
        margin: 0px;
        padding: 0px;
}

#main_menu li {
        display:inline;
        padding:0;
        margin: 0px;
}

#main_menu a {
        float: left; display: block; text-align: center;
        margin: 1px; padding: 5px; width: 80px;
        text-decoration: none; color:#069; font-weight: bold;
        background-color: #eee;
}
```

```
#main_menu a:hover {
      color: white; font-weight: bold;
      background-color: #069;
      text-decoration: underline;
}

/* Estilos de #navBar ------------- */

#navBar h1, #navBar h2, #navBar h3{
      padding-left: 10px;
}

#navBar ul a:link, #navBar ul a:visited {
      display: block;
      color: #069;
}
#navBar ul {
      list-style: none;
      margin: 0;
      padding: 0;
}

/* Estilos de #sectionLinks --------------- */

#sectionLinks{
      position: relative;
      margin-top: 20px;
      padding: 0px;
      border-bottom: 1px solid #ccc;
      font-size: 90%;
}

#sectionLinks a:link{
      padding: 2px 0px 2px 10px;
      border-top: 1px solid #ccc;
}

#sectionLinks a:visited{
      border-top: 1px solid #ccc;
      padding: 2px 0px 2px 10px;
}

#sectionLinks a:hover{
      border-top: 1px solid #ccc;
      background-color: #ccc;
      padding: 2px 0px 2px 10px;
      font-size: 120%;
      font-weight: bold;
}

/* Estilos de #content -------------------- */

#content h1, #content h2, #content h3{
      padding-left: 15px;}
```

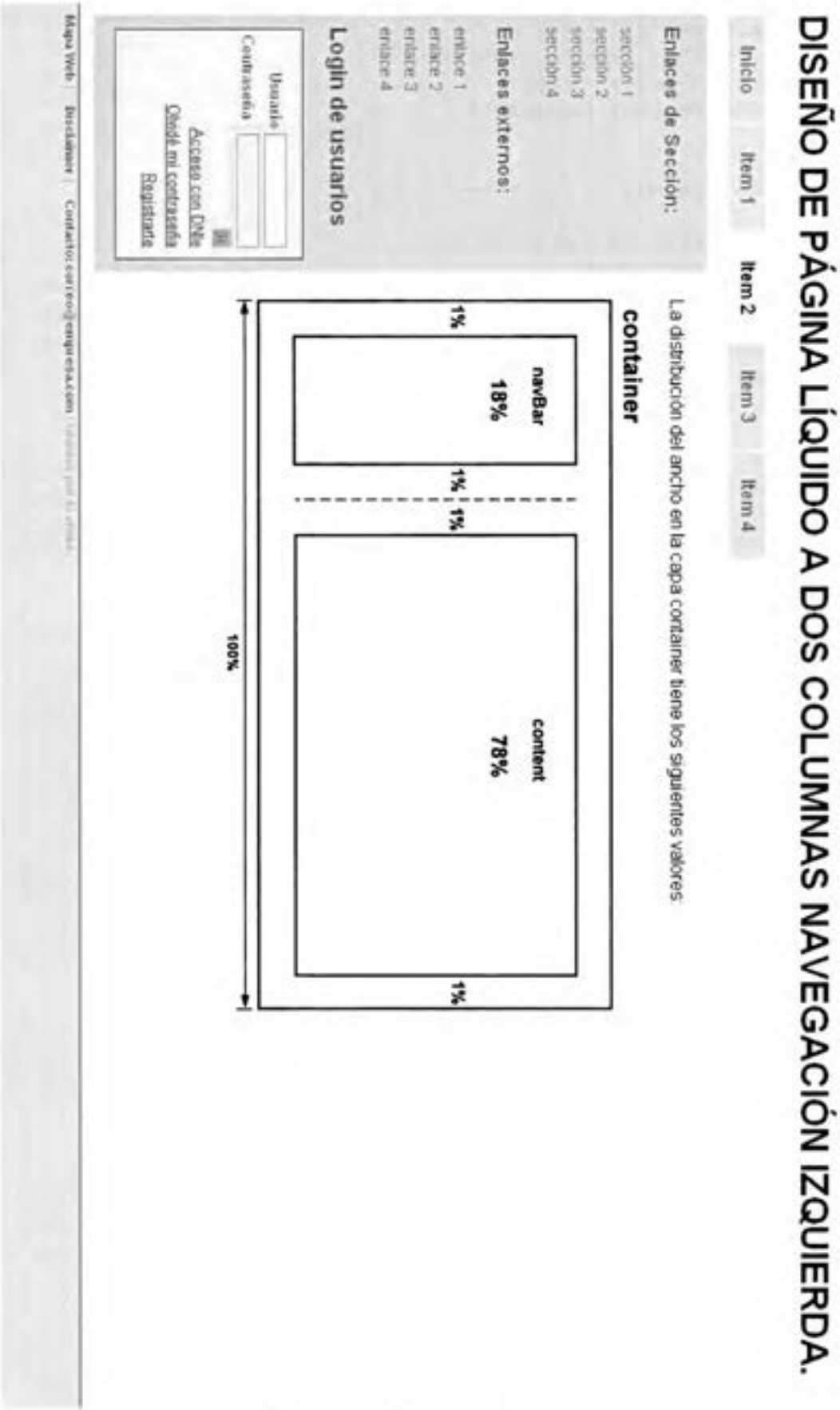


Figura 3.32: Layout a 2 columnas

3.10. Prioridad

A parte de las hojas de estilo que hemos visto hasta ahora, es decir, las definidas por los diseñadores, existen otras dos: la del navegador y la del usuario.

La hoja de estilos del navegador es la primera que se aplica y se utiliza para establecer el estilo inicial por defecto de todos los elementos. Podemos comprobar sus valores mediante las herramientas "Developer tools" de Chrome, son los valores en color gris de la derecha.

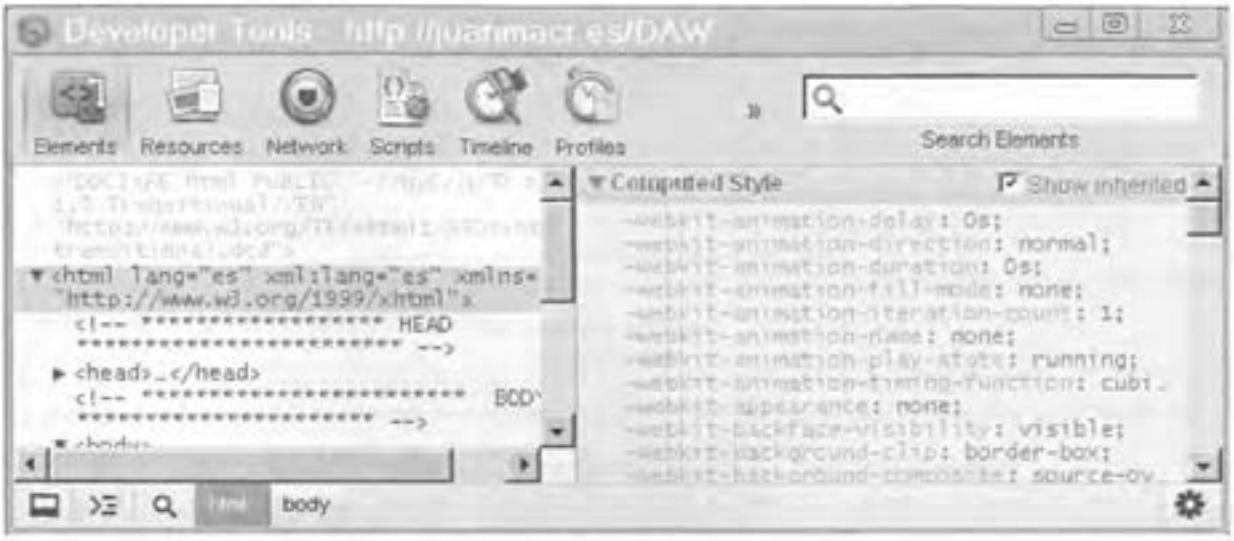


Figura 3.33: Hoja de estilos del navegador

La hoja de estilos del usuario se configura mediante alguna opción avanzada del navegador. Se trata de una opción muy útil para personas mayores o con deficiencias visuales, ya que pueden aumentar el tamaño del texto o el zoom de página completa para facilitar su lectura.

En la siguiente figura se muestra como configurar algunos estilos de usuario en Chrome mediante la ruta: Opciones, configuración avanzada, contenido web.

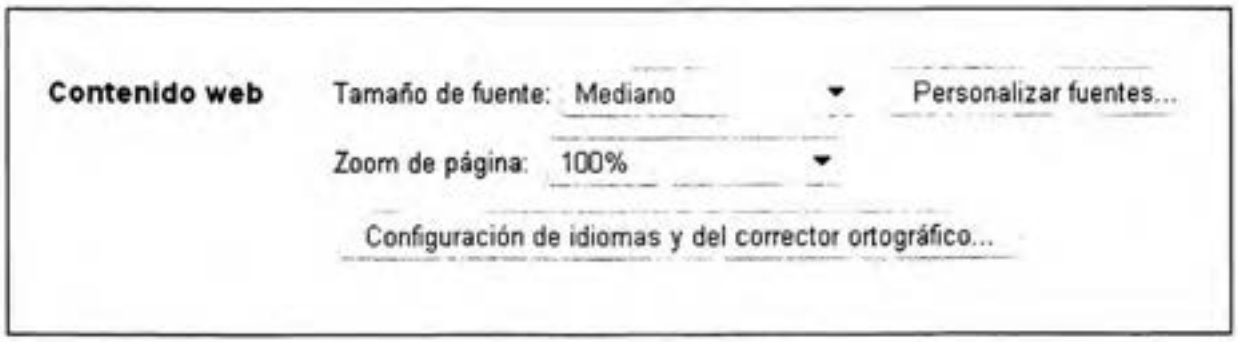


Figura 3.34: Estilos de usuario

El orden en el que se aplican las hojas de estilo es el siguiente:

Figura 3.35: Orden en el que se aplican las diferentes hojas de estilos

Las reglas que se aplican en primer lugar son las de los navegadores, después se aplican las reglas definidas por los usuarios y por último se aplican las reglas definidas por el diseñador, que por tanto son las que más prioridad tienen.

Debemos tener en cuenta que los valores por defecto que asignan los navegadores a ciertas propiedades no son siempre exactamente iguales, esto también provoca una diferente visualización de la misma página. Si dejamos alguna propiedad importante sin definir, esta tomará el valor por defecto de la hoja de estilos del navegador, que podría resultar un valor no deseado, por eso, lo aconsejable es definir todas las propiedades aunque sea a valor 0. Así por ejemplo, el margen por defecto asignado a **<body>** suele ser de 8px como podemos observar en la siguiente figura.

Figura 3.36: Estilos del navegador

3.11. Miscelánea

3.11.1. Estructuración del código CSS

Lo correcto a la hora de escribir código CSS, es distribuirlo en hojas de estilos independientes. Dentro de cada hoja de estilos, las reglas se agrupan a su vez en base a la función que realizan. Por ejemplo, se pueden organizar del siguiente modo:

- Estilos globales (para el contenido general de <html> y <body>)
- Estilos del layout (ancho, alto, posición, etc. de las capas principales)
- Estilos de cada una de las capas

3.11.2. Aplicaciones Web

Disponemos de algunas herramientas para visualizar nuestro sitio en diferentes navegadores sin necesidad de instalarlos en el equipo.

Por ejemplo http://browsershots.org/ nos visualiza una misma página web en cientos de navegadores incluyendo diferentes sistemas operativos. El resultado que nos proporciona son capturas de pantalla. El servicio es gratuito y de código abierto.

Figura 3.37: Browsershots.

3.11.3. Sitios Web orientados al diseño

- http://www.csszengarden.com/: es un sitio dedicado enteramente al diseño mediante CSS, su objetivo es inspirar y motivar a diseñadores gráficos. Dispone de ejemplos útiles.

- http://www.openwebdesign.org/: es un sitio que ofrece plantillas CSS e imágenes gratuitas bajo licencia Creative Commons, que permite utilizarlas y modificarlas libremente.

- http://css.maxdesign.com.au/ es un sitio que ofrece tutoriales en línea, muy interesantes para el aprendizaje de propiedades CSS.

Ejercicios propuestos

1. Artículo periodístico.

Crea el archivo articulo.html o bien descárgalo de http://juanmacr.es/recursos (user: Karpov. sin contraseña). Crea el archivo articulo.css para obtener el aspecto de la siguiente figura:

Sistema Operativo Android™

#comment

Ana Mª Suárez León 15/06/2012

Se trata de un sistema operativo basado en **Linux** de código abierto y pensado para dispositivos móviles. Fué desarrollado por la OHA (Open Handset Alliance) formada por un conjunto de empresas como **Google**, Ebay, Intel, Nvidia, Mottorola, Samsung o Telefónica entre otras. Sale a luz en Noviembre de 2007. Desde entonces es líder en ventas con cada vez más diferencia sobre sus competidores.

El sistema es ligero pues solo contiene 12 Millones de LCD (líneas de código) escritas en diferentes lenguajes como **XML**, C, Java y C++. Al tratarse de código abierto, la comunidad de desarrolladores es muy grande, por ello no es de extrañar el número tan elevado de aplicaciones del que dispone, en torno al medio millón. Algunas de las cuales son:

- Audio: ***Google Music*** para escuchar música, ***TuneIn*** para radio, ***Speaktoit*** para reconocimiento de voz
- Libros: Lectores de archivos ***Kindle, Aldiko***
- Comunicación: ***Skype, Viber, WhatsApp, eBudy***
- Juegos: ***Angry Birds***
- Redes sociales: ***FaceBook, Messenger, LinkedIn, Twitter***
- Imagen: ***PicsArt*** para edición de imagen, ***Instagram*** para compartirlas
- Televisión: ***TDT Android*** para ver tv online
- Widgets: ***Widgetsoid*** permite crear widgets

Like 72 Send +1 47 **Tweet** 194

Figura 3.38: Artículo

2. Currículum Vitae.

Crea los archivos cv.html y cv.css para obtener el resultado de la figura.

Crea una capa por cada zona del curriculum, dentro de la capa, utiliza listas sin ordenar para secuenciar el contenido. Sitúalo en el centro mediante `margin: 0 auto;`

CURRICULUM VITAE

Datos Personales

Nombre: José Andrés Sánchez García
Fecha de Nacimiento: 1979-02-07
Email: jasg@gmail.com
Teléfono: 689 689 689

Formación

- Título 1: Técnico superior en radiotelecomunicación
  Lugar: Escuela Superior de Electrónica
  Período: Fecha Inicio=1995-10-01 Fecha Fin=1997-06-30
- Título 2: Master en dirección de empresas
  Lugar: Escuela de Negocios Metropolitana
  Período: Fecha Inicio=2000-01-10 Fecha Fin=2001-06-30

Experiencia Laboral

- Cargo: Inspector
  Lugar: Policía
  Período: Fecha Inicio=2004-03-14
- Cargo: Radiotelegrafista
  Lugar: Aeropuerto
  Período: Fecha Inicio=2001-09-12 Fecha Fin=2004-03-07

Idiomas

- Idioma: Castellano (Lengua Materna)
- Idioma: Inglés
  Nivel (1-5):
  - Expresión Oral: 3
  - Comprensión Oral: 2
  - Expresión Escrita: 3
  - Comprensión Escrita: 4
- Idioma: Francés
  Nivel (1-5):
  - Expresión Oral: 2
  - Comprensión Oral: 2
  - Expresión Escrita: 3
  - Comprensión Escrita: 3

Otras Informaciones

- Licencia de Conducción: B1
- Vehículo Propio
- Descripción Personal: Tímido, perfeccionista, polifacético

Figura 3.39: Curriculum Vitae

3. Factura.

Genera los archivos factura.html y factura.css para obtener el resultado de la siguiente figura. Crea las capas: header, container y footer.

Dentro de header va el logo, los datos de empresa y los datos de la factura.

Dentro de container van los datos del cliente y las líneas de factura mediante tablas.

En el footer insertamos un formulario con campos checkbox y textbox para la forma de pago y datos bancarios.

Para los datos con recuadro puedes crear una clase específica llamada "recuadro".

industrias starfuerte

Industrias StarFuerte
CIF: 12345678-G
C/Forjadores s/n
Isla del Hierro
Tenerife
España

FACTURA

Número: 1
Fecha: 30/04/2008

Datos del Cliente

Cliente: Tío Sam CIF: 94-6666666

Dirección: Pennsylvania Avenue, 1600 Provincia: Washington

País: Estados Unidos

Líneas de factura

Cantidad	Artículo	Precio	Subtotal
1	AR001 - Armadura de oro reluciente	350.000$	350.000$
3	ES002 - Escudo antiproyectiles	50.000$	150.000$

Base Imponible	500.000$
Descuento	0
IVA 20%	100.000$
Total Neto	600.000$

Forma de pago y Datos bancarios

Pago al contado ☐
Pago domiciliado ☑
Pago con tarjeta ☐

Entidad:
Nº de cuenta:

Figura 3.40: Factura.html

4. Menú desplegable.

Genera los archivos desplegable.html y desplegable.css para obtener el resultado de la siguiente figura. El menú está formado por una lista anidada, es decir, ul li ul li.

Para las propiedades css tener en cuenta el siguiente posicionamiento:

#menu ul li {float: left}

#menu ul li a {position: relative}

#menu ul li ul {position: absolute}

#menu ul li ul li {position: relative}

#menu ul li ul li a {display : block}

Figura 3.41: Desplegable.html

CAPÍTULO 4

XML. Almacenamiento de datos

CONTENIDO

Introducción
Documentos XML
Estructura jerárquica de un documento XML
Modelo de datos de un documento XML. Nodos
Corrección sintáctica: documento XML bien formado
Documentos XML válidos
Validación de documentos XML con DTD
Validación de documentos XML con esquemas XML
Otros mecanismos para validar XML
Otros lenguajes basados en XML
Otras formas de almacenar información
Ejercicios propuestos

4.1. Introducción

En este tema se van a tratar, de manera casi exclusiva, aspectos relativos a una tecnología aparecida hace ya algunos años: el **XML**. Se trata de un formato de almacenamiento de información a base de etiquetas o marcas definidas por el usuario.

Por el hecho de ser un lenguaje de marcas, deberá cumplir una serie de reglas que harán que un **documento XML** esté **bien formado**: la forma en la que se abren y cierran las etiquetas, la forma en la que se escriben sus atributos, la existencia de un elemento que los contenga a todos, la aparición de comentarios y su formato, etc.

Adicionalmente, se aprenderán mecanismos para **validar un documento XML**. Esto significa "crear" un lenguaje de marcas para un uso específico, ya que se indica qué elementos y atributos pueden aparecer (vocabulario), el orden en el que aparecen, qué elemento contiene a cual, qué atributos tiene un elemento, qué elementos o atributos son optativos y cuáles son obligatorios, etc.

Aquí se revisarán fundamentalmente dos técnicas: los **DTD** y los **esquemas XML**. La primera es una técnica algo obsoleta pero muy extendida hasta principios de los 2000. La segunda, que es una técnica más sofisticada que permite unos niveles de definición mucho más precisos, se revisará en profundidad.

Se describirán brevemente otras técnicas mucho menos usadas para validar documentos XML: Relax NG y Schematron.

Se hará un repaso ligero de **algunos lenguajes basados en XML** para usos específicos, como SVG, WML, RSS, FO...

Se citarán dos formatos de almacenamiento de información alternativos al XML: **JSON** y **YAML**.

Este es un **capítulo prioritario** del libro. Los **contenidos más importantes** del capítulo son la comprensión del formato XML, qué representa que un documento esté bien formado, las técnicas de validación de documentos XML, sobre todo los esquemas.

4.2. Documentos XML

El XML (eXtensible Markup Language – Lenguaje de Marcado eXtensible) es un estándar (o norma), no una implementación concreta.

Es un metalenguaje de marcas, lo que significa que no dispone de un conjunto fijo de etiquetas (como sucedía con HTML) que todo el mundo debe conocer. Por el contrario, XML permite definir a los desarrolladores los elementos que necesiten y con la estructura que mejor les convenga.

Define una sintaxis general para maquetar datos con etiquetas sencillas y comprensibles al ojo humano (a cualquier humano). Provee, asímismo, un formato estándar para documentos informáticos. Es un formato flexible, de manera que puede ser adaptado al campo de aplicación que se desee.

Ejemplo:

En el campo de la química, tendría sentido la existencia de etiquetas como `<átomo>`, `<molécula>` o `<enlace>`.

En el campo de la composición musical, tendría sentido que hubiera etiquetas como `<entera>`, `<negra>` o `<semicorchea>`.

De ahí viene el extensible del nombre XML: se pueden crear nuevas etiquetas en función de las necesidades.

Lo que no es XML

- XML no es un lenguaje de programación, de manera que no existen compiladores de XML que generen ejecutables a partir de un documento XML.
- XML no es un protocolo de comunicación, de manera que no enviará datos por nosotros a través de internet (como tampoco lo hace HTML). Protocolos de comunicación son HTTP (**H**yper-**T**ext **T**ransfer **P**rotocol – Protocolo de Transferencia de Hipertexto), FTP (**F**ile **T**ransfer **P**rotocol – Protocolo de Transferencia de Ficheros), etc. Estos y otros protocolos de comunicación pueden enviar documentos con formato XML.
- XML no es un sistema gestor de bases de datos. Una base de datos relacional puede contener campos del tipo XML. Existen, incluso, bases de datos XML nativas, que todo lo que almacenan son documentos con formato XML. Pero XML en sí mismo no es una base de datos.
- No es propietario, es decir, no pertenece ninguna compañía, como sucede con otros formatos.

Formato adecuado para el almacenamiento y la transmisión

Puesto que es un formato de *texto plano* (no se trata de archivos binarios) es adecuado para almacenar información y trasmitirla. Con el más simple editor de textos se puede editar un documento XML. Asímismo, los documentos XML son relativamente ligeros para ser almacenados y enviados, puesto que ocupan lo que los datos que contienen más las etiquetas que los delimitan. Suelen tener extensión `.xml`, aunque no es imprescindible.

Estas etiquetas (y sus atributos) son meta-información, es decir, información sobre la información (sobre los datos). Nos permiten estructurar el documento y facilitan su procesamiento, pero no son información en sí mismas.

4.3. Estructura jerárquica de un documento XML

En un documento XML la información se organiza de forma jerárquica, de manera que los elementos del documento se relacionan entre sí mediante relaciones de padres, hijos, hermanos, ascendentes, descendentes, etc.

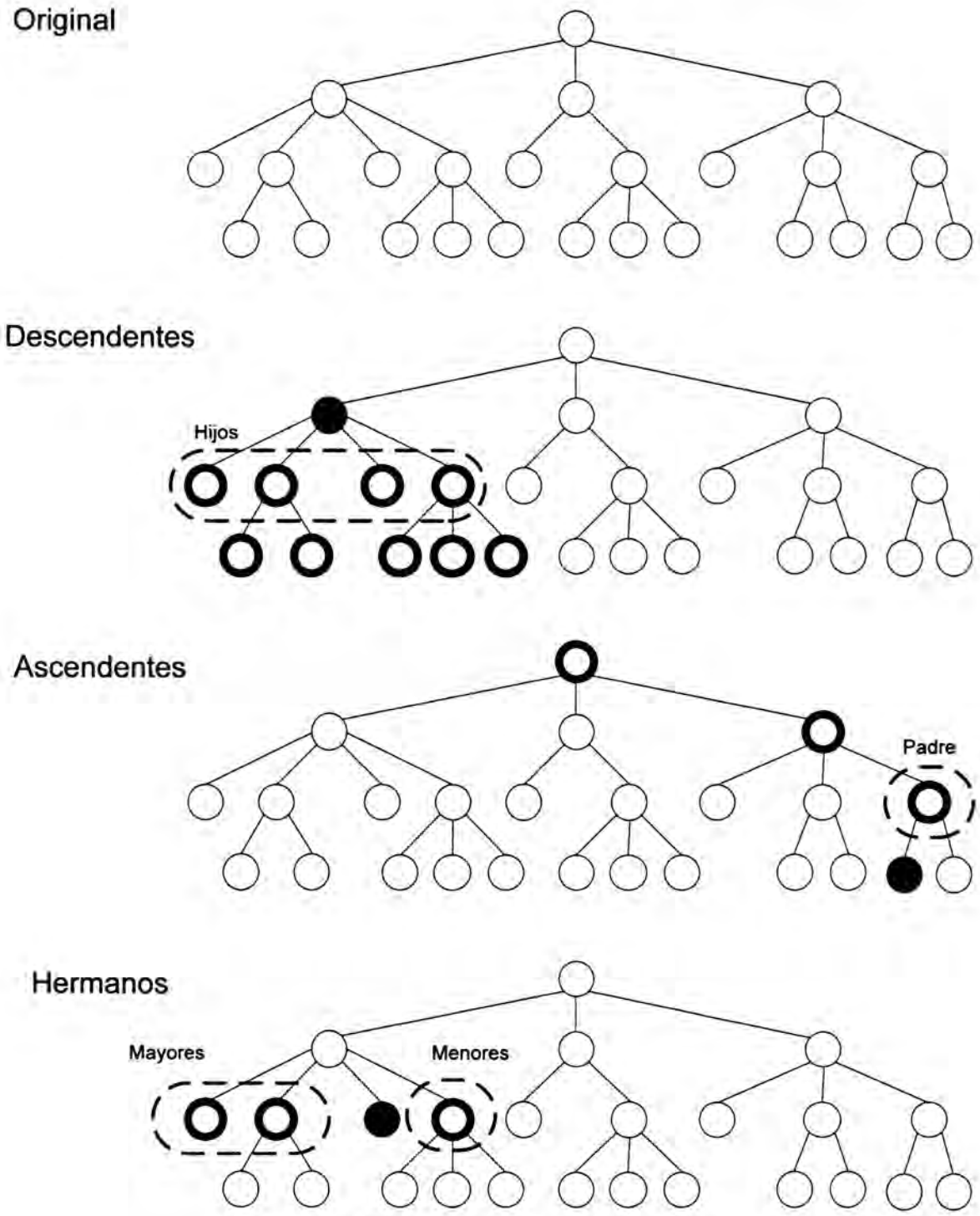


Figura 4.1: Relaciones entre nodos

A esta estructura jerárquica se la denomina árbol del documento XML. A las partes del árbol que tienen hijos se las denomina nodos intermedios o ramas, mientras que a las que no tienen se conocen como nodos finales u hojas.

Ejemplo:

El elemento <persona> es "padre" (contiene) a los elementos <nombre> y <apellido>, que son "hermanos" entre sí.

```
<persona>
      <nombre>María</nombre>
      <apellido>González</apellido>
</persona>
```

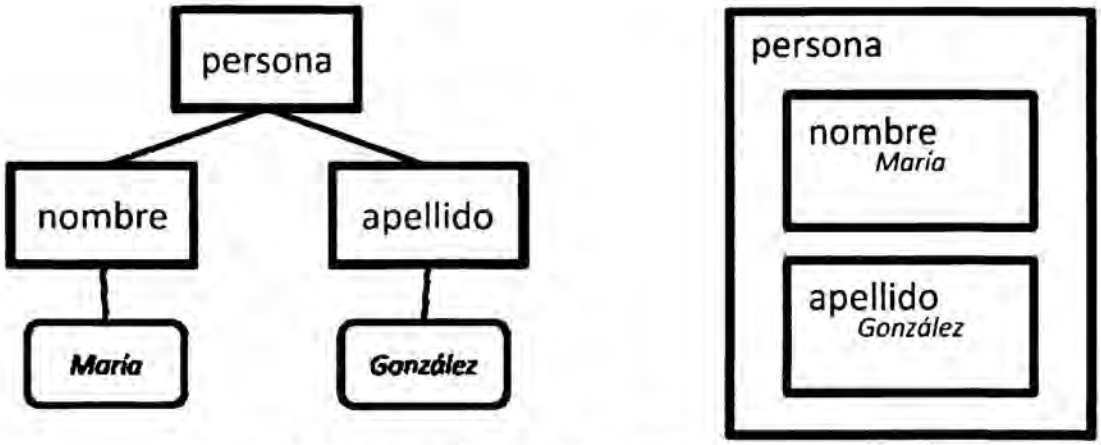


Figura 4.2: Modelo de datos: diagrama de árbol y diagrama de cajas

Ejemplo:

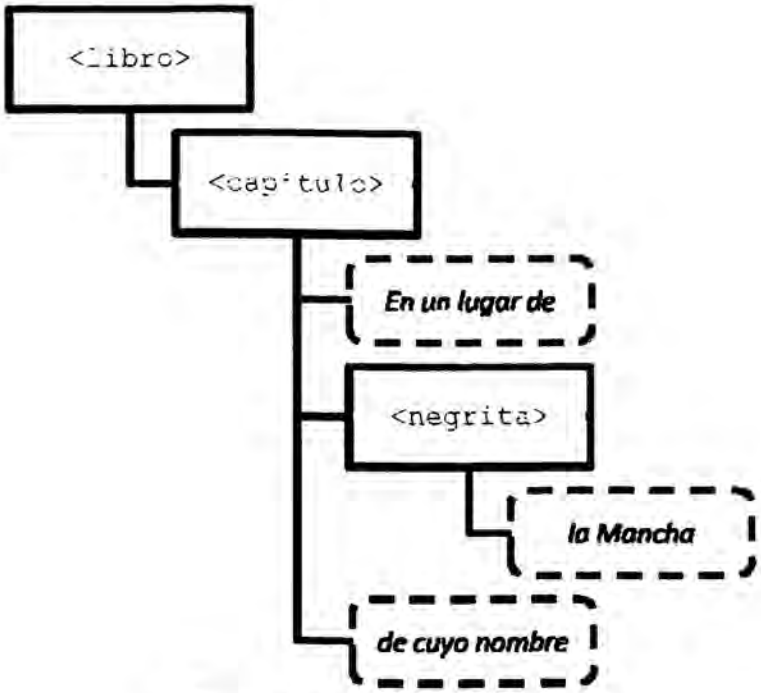


Figura 4.3: Estructura de nodos

```
<libro>
  <capitulo>
    En un lugar de <negrita>la Mancha</negrita> de cuyo nombre...
  </capitulo>
</libro>
```

El elemento `<capitulo>` tiene tres hijos:

- El nodo hoja (contenido textual) *En un lugar de*
- El nodo `<negrita>`
- El nodo *de cuyo nombre...*

Visualización de un documento XML

No dispone de una visualización concreta en un navegador puesto que el documento no refleja una apariencia, sino unos datos.

Existen varias maneras para representar visualmente los datos de un documento XML:

- Una forma es mediante una hoja de estilo CSS que indique al navegador cómo convertir cada elemento del documento XML en un elemento visual. Se lograría con la siguiente instrucción de procesamiento que se verá más adelante:

  ```
  <?xml-stylesheet type="text/css" href="estilos.css"?>
  ```

- Otra manera es mediante el uso de una hoja de transformaciones XSLT. La instrucción equivalente para asociar a un documento XML una hoja de transformaciones XSLT es:

  ```
  <?xml-stylesheet type="text/xsl" href="transforma.xsl"?>
  ```

- También se podría hacer mediante el uso de un lenguaje de programación, como Java o JavaScript, que procese el documento XML.

4.4. Modelo de datos de un documento XML. Nodos

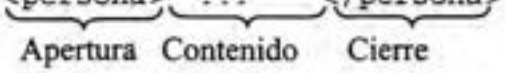
Como se ha comentado, un documento XML consta de una determinada estructura, formada por los siguientes tipos de componentes o nodos:

- Raíz:

Por encima de cualquier elemento se ubica el nodo raíz, que se designa como "/". No es un componente que tenga representación dentro del documento XML, pero se utilizará más adelante como punto de partida para recorrer el árbol XML y ubicar el resto de nodos.

- Elementos:

Es la unidad básica de un documento XML. Son delimitadores/contenedores de información. Al igual que los elementos de HTML, se identifican por una etiqueta de apertura (como `<persona>`) y una de cierre (como `</persona>`). Lo que se ubica entre ambas es el contenido de ese elemento, que puede ser textual, otros elementos o vacío.



Algunos tipos de elementos especiales son:

- Elemento raíz: todo documento XML bien formado debe contener un único elemento raíz que contiene a todos los demás (no tiene ascendentes ni hermanos). También se le llama elemento documento.

- Elementos sin contenido: aunque puede tener atributos, se abre y se cierra con una sola etiqueta.

Ejemplo:

Elemento sin contenido y sin atributos. `<separador />`

Elemento sin contenido pero con atributos. `<separador cantidad="7" />`

- Atributos:

Son como los atributos de HTML. Son pares nombre-valor que permiten especificar datos adicionales de un elemento. Se ubican en la etiqueta de apertura del elemento. Para asignar un valor a un atributo se utiliza el signo igual. Todos los atributos, independientemente del tipo de datos que representen, se tratarán como texto y aparecerán entre comillas simples o dobles.

Se usan fundamentalmente para almacenar información sobre la información (meta-información) contendida en el documento.

Ejemplo:

```
<distancia unidades="km">70</distancia>
```

También se emplean para recoger información identificativa del elemento que permita distinguirlo de otro elemento.

Ejemplo:

```
<persona nif="12345678Z">...</persona>
```

- Texto:

El texto, como tipo de nodo que representa los datos del documento XML, puede aparecer, bien como contenido de un elemento, bien como valor de un atributo. No puede aparecer en ningún otro lugar.

Un tratamiento especial tienen los espacios en blanco. Hay cuatro tipos de caracteres de espaciado en XML:

- Tabulador:	\t		TAB
- Nueva línea:	\n
LF
- Retorno de carro:	\r
CR
- Espacio:	\s		NO-BREAK SPACE

Dentro del contenido textual de un elemento se mantendrán como están y así serán tratados por el procesador. Como valor de un atributo, los espacios en blanco adyacentes se condensarán en uno solo. Los espacios en blanco entre elementos serán ignorados.

Ejemplo:

<A> Dato </A>	≠	<A>Dato</A>
<A b="otro dato" />	=	<A b="otro dato" />
<A>Más datos</A> <B>Y más</B>	=	<A>Más datos</A><B>Y más</B>

- Comentarios:

Son iguales que los de HTML. Empiezan por los caracteres <!-- y se cierran con los caracteres -->. Dentro de ellos se puede escribir cualquier signo (sin necesidad de caracteres de escape) menos el doble guión (--) que confundiría al analizador con un posible cierre del comentario.

Pueden ubicarse en cualquier lugar de un documento, menos dentro de una etiqueta de apertura, ni dentro de una etiqueta de cierre.

- Espacio de nombres:

Es un mecanismo para distinguir etiquetas cuando se mezclan distintos vocabularios. Se verán en detalle más adelante.

- Instrucciones de procesamiento:

Las instrucciones de procesamiento empiezan por <? y terminan por ?>. Son instrucciones para el procesador XML, de manera que son dependientes de él. No forman parte del contenido del documento XML. Se utilizan para dar información a las aplicaciones que procesan el documento XML.

Ejemplo:

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

- Entidades predefinidas:

Representan caracteres especiales de marcado, pero son interpretados como texto por parte del procesador XML.

Entidad	Carácter
&	&
<	<
>	>
'	'
"	"

Ejemplo:

```
<libreria>Barnes &amp; Noble</libreria>
```

El contenido del elemento <bebida> será interpretado por el analizador como Barnes & Noble.

- Secciones CDATA:

Son conjuntos de caracteres que el procesador no debe analizar (parecidos a un comentario). La definición de estas secciones permite agilizar el análisis del documento y deja libertad al autor del documento para introducir libremente en ellas caracteres como < y &.

No pueden aparecer antes del elemento raíz ni después de su cierre. No pueden contener el propio signo delimitador de final de sección CDATA, es decir, la combinación de caracteres]]>

Ejemplo:

Se quiere introducir como contenido de un elemento <codigo> un trozo de código HTML y se desea que el analizador XML no lo tome como etiquetas, sino que lo deje sin procesar.

```
<codigo>
  <![CDATA[
    <html><body><h3>Título de la página</h3></body></html>
  ]]>
</codigo>
```

- Definición de Tipo de Documento (DTD):

Permite definir reglas que fuercen ciertas restricciones sobre la estructura de un documento XML aunque su existencia no es obligatoria. Un documento XML bien formado que tiene asociado un documento de declaración de tipos y cumple con las restricciones allí declaradas se dice que es **válido**. Más adelante se verán en profundidad éste y otros mecanismos de validación de documentos XML.

Debe aparecer en la segunda línea del documento XML, entre la instrucción de procesamiento inicial y el elemento raíz.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> ❶
<!DOCTYPE document system "elementos.dtd"> ❷
<!-- Aquí viene un comentario --> ❸
<?xml-stylesheet type="text/css" href="elementos.css"?> ❹
<elementos> ❺
  <elemento> ❻
    <nombre>Agua</nombre>
    <color>Azul</color>
  </elemento>
  <elemento> ❻
    <nombre>Fuego</nombre>
    <color>Rojo</color>
  </elemento>
</elementos> ❼
```

❶ Instrucción de procesamiento que declara que el documento está en formato XML.

❷ Inclusión de un DTD externo, ubicado en el archivo elementos.dtd.

❸ Un comentario.

❹ Instrucción de procesamiento que vincula al documento XML una hoja de estilos ubicada en el archivo elementos.css.

❺ Elemento raíz (también llamado elemento documento). En este punto se abre.

❻ Diversos elementos descendientes del elemento raíz y los descendientes de éstos.

❼ Cierre del elemento raíz.

Nombres XML

En XML se utilizan una serie de reglas para definir nombres correctos de elementos. Son las mismas que para los atributos.

Un nombre XML, que así se denomina:

- Puede empezar con una letra (con o sin tilde), que podría ser de un alfabeto no latino, subrayado o dos puntos (este último carácter es desaconsejado ya que se reserva su uso para los espacios de nombres).
- Los siguientes caracteres pueden ser letras, dígitos, subrayados, guiones bajos, comas y dos puntos.
- Los nombres que empiezan por las letras XML, en cualquier combinación de mayúsculas y minúsculas, se reservan para estandarización.
- No pueden contener:
    - Ningún carácter de espaciado.
    - Ningún otro carácter de puntuación que los ya citados como válidos. Esto incluye: comillas simples o dobles, signo de dólar, acento circunflejo, signo de porcentaje, punto y coma.

Ejemplo:

Nombres correctos:

```
<Número_Seguridad_Social>50-12345678</Número_Seguridad_Social>
<primerApellido>Rodríguez</primerApellido>
<_cuenta_Tweeter>@follower</_cuenta_Tweeter>
<persona>Галина"Иванов</persona>
```

Nombres incorrectos:

```
<O'Donnell>General</O'Donnell>  ❶
<día/mes/año></día/mes/año>  ❷
<fecha nacimiento>2011-02-02 </fecha nacimiento>  ❸
```

❶ No es válido por el carácter apóstrofe

❷ No es válido por el signo de dividir

❸ No es válido por el espacio

Uso de elementos frente a uso de atributos

Los elementos:

- se emplean para representar jerarquías o contenido de unos dentro de otros.
- se pueden extender con otros elementos en su interior.
- el orden en el que aparecen es representativo.
- pueden tener atributos.
- puede haber múltiples ocurrencias de un elemento.

Los atributos:

- van asociados a los elementos.
- son modificadores de la información.
- se suelen usar para registrar metadatos.
- el orden en que aparecen dentro del elemento al que van asociados no es representativo.
- no se pueden extender con otros elementos contenidos en su interior.
- no puede haber múltiples ocurrencias de un atributo dentro de un mismo elemento.

Espacios de nombres

Es un mecanismo para evitar conflictos de nombres, de manera que se puedan diferenciar elementos o atributos dentro de un mismo documento XML que tengan idénticos nombres pero diferentes definiciones. No aparecieron en la primera especificación de XML, pero sí pronto después.

Se declaran como atributo de elementos de la forma:

```
<nombre_elemento xmlns:prefijo="URI_del_espacio_de_nombres">
```

Y se usan anteponiendo a elementos y atributos con el prefijo asociado al espacio de nombres además del carácter dos puntos ":".

Ejemplo:

```
<info:pedido xmlns:info="empresa:espacios:info">
  <info:item info:id="i_13">Afeitadora eléctrica</info:item>
  ... ...
</info:pedido>
```

Cualquier cadena de texto puede ser usada como prefijo del espacio de nombres. El URI del espacio de nombres sí debe ser único, aunque realmente no se comprueba mediante conexión alguna. El URI no es más que un nombre lógico del espacio de nombres. A veces se usa como URI `http://-`.

El **ámbito** de declaración o uso **de un espacio de nombres**, incluye los lugares donde se puede referenciar este espacio de nombres. Cubre el elemento donde se ha declarado y sus elementos descendientes. A todos estos elementos se les puede anteponer el prefijo del espacio de nombres.

Asimismo, se puede declarar un espacio de nombres diferente para un elemento descendiente de otro, en el cual ya se declarado otro espacio de nombres.

Ejemplo:

En un documento XML se declaran dos espacios de nombres, `empresa:espacios:dept` y `empresa:espacios:emp`, de prefijos `dept` y `emp` respectivamente.

```
<dept:departamentos xmlns:dept="empresa:espacios:dept"
                    xmlns:emp="empresa:espacios:emp">
  <dept:departamento dept:deptno="10">
    <emp:empleado emp:empno="7654">
      <emp:nombre>Roberto</sec:nombre>
      <emp:apellido>Mate</emp:apellido>
    </emp:empleado>
    <emp:empleado emp:empno="7891">
      <emp:nombre>Lucia</emp:nombre>
      <emp:apellido>Palmito</emp:apellido>
    </emp:empleado>
  </dept:departamento>
</dept:departamentos>
```

Los atributos pueden pertenecer al mismo espacio de nombres al que pertenece el elemento en el que están asociados o a otro diferente. La definición es la misma que con los elementos, anteponiendo el prefijo del espacio de nombres y los dos puntos al nombre el atributo. Si no aparece ningún prefijo de espacio de nombres, el atributo no pertenecerá a ningún espacio de nombres.

Ejemplo:

El atributo `trabajo:empno` del elemento `<emp:datosPersonales>` pertenece a un espacio de nombres diferente que éste. El prefijo del espacio de nombres del atributo es `trabajo`. El atributo `deptno` del elemento `<emp:departamento>` no pertenece a ningún espacio de nombres.

```
<emp:empleado xmlns:emp="empresa:espacios:emp">
  <emp:datosPersonales trabajo:empno="7583"
          xmlns:trabajo="empresa:espacios:trabajo">
    <emp:departamento deptno="10">Ventas</emp:departamento>
    <emp:nombre>Roberto</emp:nombre>
    <emp:apellido>López</emp:apellido>
  </emp:datosPersonales>
</emp:empleado>
```

Actividad 4.1:

Indica a qué espacio de nombres pertenece el elemento `<info:venta>` del siguiente documento XML y justifica tu respuesta.

```
<?xml version="1.0" ?>
<info:venta xmlns:info="empresa:espacios:info"
            xmlns:prod="empresa:espacios:prod">
  <prod:producto prod:id="p_987">Mesa de despacho</prod:producto>
  <prod:precio prod:mondeda="Euro">300</prod:precio>
  <prod:unidades>200</prod:unidades>
</info:venta>
```

a. A `empresa:espacios:info`

b. A `empresa:espacios:prod`

c. A `empresa:espacios:info` y `empresa:espacios:prod`

d. A ningún espacio de nombres

Dos atributos con el mismo nombre pero con distinto prefijo de espacio de nombres son diferentes, lo que significa que pueden estar asociados al mismo elemento.

Ejemplo:

Los atributos `trabajo:empno` y `emp:empno` son diferentes, luego pueden aparecer asociados al mismo elemento.

```
<emp:empleado xmlns:emp="empresa:espacios:emp"
              xmlns:trabajo="empresa:espacios:trabajo">
  <emp:datosPersonales trabajo:empno="7583" emp:empno="e_7583">
    <emp:departamento deptno="10">Ventas</emp:departamento>
    <emp:nombre>Mercedes</emp:nombre>
    <emp:apellido>López</emp:apellido>
  </emp:datosPersonales>
</emp:empleado>
```

Espacio de nombres por defecto

Un espacio de nombres por defecto es aquel en el que no se define un prefijo. El ámbito de aplicación de ese espacio de nombres es el del elemento en el que se ha declarado y sus elementos descendientes, pero no a sus atributos.

Si se tuviera que añadir un espacio de nombres con prefijo a un documento XML grande ya creado, habría que ir añadiendo el prefijo a los elementos y atributos, algo tedioso y tendente a producir errores. De esta manera, se puede evitar tener que escribir el prefijo.

El mayor inconveniente de esta medida, citado ya antes, es que el espacio de nombres por defecto sólo afecta al elemento en el que está declarado y a sus descendientes, no a los atributos.

Ejemplo:

El espacio de nombres por defecto, `empresa:espacios:emp`, no afectará al atributo `empno` del elemento `<datosPersonales>` ni al atributo `deptno` del elemento `<departamento>`.

```
<empleado xmlns="empresa:espacios:emp"
                xmlns:trabajo="empresa:espacios:trabajo">
  <datosPersonales trabajo:empno="7583" empno="e_7583">
    <departamento deptno="10">Ventas</departamento>
    <nombre>Mercedes</nombre>
    <apellido>López</apellido>
  </datosPersonales>
</empleado>
```

Se puede desasignar a un elemento de un espacio de nombres por defecto con la orden:

```
<nombre_elemento xmlns="">
```

Actividad 4.2:

Indica a qué espacio de nombres pertenece el atributo `moneda` del elemento `<precio>` del siguiente documento XML y justifica tu respuesta.

```
<?xml version="1.0" ?>
<venta xmlns="empresa:espacios:info"
         xmlns:venta="empresa:espacios:venta">
    <producto id="p_987">Mesa de despacho</producto>
    <precio moneda="Euro">300</precio>
    <unidades>200</unidades>
</venta>
```

a. A empresa:espacios:info

b. A `empresa:espacios:venta`

c. A `empresa:espacios:info` y `empresa:espacios:venta`

d. A ningún espacio de nombres

Atributos especiales

Hay algunos atributos especiales en XML:

- `xml:space`: le indica a la aplicación que usa el XML si los espacios en blanco del contenido textual de un elemento son significativos.

  Ejemplo:

  Se le indica al XML que los espacios se mantengan (*preserve*) o se eliminen (*default*):

```
<!ATTLIST quitaEspacios xml:space #FIXED "default">
<!ATTLIST dejaEspacios xml:space #FIXED "preserve">
```

- `xml:lang`: permite especificar el idioma en el que está escrito el contenido textual de todo un documento o de sus elementos individuales. Los posibles valores son:
    - un código de dos letras (ISO 639).

      Ejemplo: *en* para English (existen subcódigos, como *en-GB* y *en-US* para inglés británico y americano respectivamente).
    - un identificador de idioma registrado por el IANA (Internet Assigned Numbers Authority – Autoridad de Números Asignados en Internet). Estos identificadores empiezan por el prefijo i- o I-.
    - un identificador definido por el usuario. Estos identificadores empiezan por el prefijo x- o X—
- `xml:base`: permite definir una URI distinta a la del documento.

Parser XML

Un analizador XML (llamado en inglés *parser*), es un procesador que lee un documento XML y determina la estructura y propiedades de los datos en él contenidos. Un analizador estándar lee el documento XML y genera el árbol jerárquico asociado, lo que permite ver los datos en un navegador o ser tratados por cualquier aplicación.

Si el analizador comprueba las reglas de buena formación y además valida el documento contra un DTD o esquema, se dice que se trata de un **analizador validador**. Estos analizadores también comprueban la semántica del documento e informan de los errores existentes.

Existen validadores XML en línea, como XML Validation (http://www.xmlvalidation.com).

Editores XML

Existen multitud de herramientas relacionadas con el manejo de XML y sus tecnologías asociadas (DTD, XSD, XPATH, XSLT, XSL-FO...). Algunas son muy sofisticadas y, en general con licencias propietarias. Hay otras más sencillas, también de código abierto. Por citar algunas:

- **XMLSpy**, de Altova (http://www.altova.com/es). Es un editor de XML, pero también de XSLT, XPath, XQuery... Forma parte de una suite de herramientas (MissionKit) interrelacionadas entre sí, entre las que cabe destacar también StyleVision, cuyo uso principal es el diseño visual de hojas de transformaciones XSLT o XSL-FO y la generación de documentos de salida en formatos como PDF, HTML, RTF, PostScript...
  La fortaleza más importante de estos programas es la existencia de representaciones gráficas de los distintos documentos, que pueden ser directamente editadas.
  En el momento de la impresión de este libro, la versión más moderna es la Enterprise 2012 y la licencia es propietaria.
- **<oXygen/>**, de Syncro Soft (http://www.oxygenxml.com). Al igual que el producto de Altova, además de XML, es un editor de diferentes tecnologías como XSD, XSLT o XQuery. También las interfaces de desarrollo gráfico están muy cuidadas.

Existen versiones para diversas plataformas (Windows, Linux, Mac...). Está compuesto de dos módulos principales, XML Developer (el editor de XML en sí) y XML Author (generador de documentos en distintos formatos), que se pueden instalar también por separado. En el momento de la impresión de este libro, la versión más moderna es la 13.2 y la licencia es propietaria.

- **XML Copy Editor** (http://xml-copy-editor.sourceforge.net). Es software libre bajo licencia GNU GPL, siendo Gerald Schmidt el desarrollador principal. La última versión existente es la 1.2.0.6.
  Es un editor de XML, XSD, XSLT...
  Esta es la herramienta que se usará para el seguimiento de este libro. Si bien las otras dos, citadas previamente son de una excelente calidad e incluyen funcionalidades más avanzadas que XML Copy Editor, esta última es suficiente.
- **XMLPad Pro Edition**, de WMHelp (http://www.wmhelp.com/download.htm). Es una herramienta sencilla cuyo desarrollo parece interrumpido en el momento de la publicación de este libro, pero que incluye funcionalidades como comprobación de buen formado de documentos XML, validación frente a DTD o esquemas XML e inferencia de DTD o esquema XML a partir de un XML.
- **Otras aplicaciones:**
  Exchanger XML Editor (http://www.exchangerxml.com)
  Liquid XML Editor (http://www.liquid-technologies.com/Xml-Studio.aspx)

Figura 4.4: Interfaz del programa XML Copy Editor

4.5. Corrección sintáctica: documento XML bien formado

La especificación de XML define la sintaxis que el lenguaje debe seguir:

- Cómo se delimitan los elementos con etiquetas.
- Qué formato puede tener una etiqueta.
- Qué nombres son aceptables para los elementos.
- Dónde se colocan los atributos.

Un documento XML se dice que está **bien formado** si cumple las reglas establecidas por el W3C en las especificaciones para XML. Estas reglas son muy numerosas y, en ocasiones, complejas de entender, así que se citarán las más significativas

- El documento puede (el W3C lo recomienda) empezar por una instrucción de procesamiento `xml`, que indica la versión del XML y, opcionalmente, la codificación de caracteres (`encoding`), que por defecto es *UTF-8*, y si está listo para procesarse independientemente o requiere de otros archivos externos para dicha tarea (`standalone`), que por defecto vale *no*.

  La instrucción de procesamiento correcta más simplificada es:
  ```
  <?xml version = "1.0"?>
  ```
  Otra más extensa e igualmente correcta:
  ```
  <?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
  ```
  Unos apuntes sobre juegos de caracteres y codificaciones:

  . Un juego de caracteres es una colección de caracteres asociados cada uno a un número, llamado *punto de código*. Por ejemplo, en ASCII el punto de código de la A es 65.
  . Un ejemplo de juego de caracteres es Unicode (actualmente en su versión 6). Contiene todos los caracteres usados en todos los alfabetos existentes en el mundo.
  . Otro ejemplo de juego de caracteres es el ISO Latin 1. Una codificación de este juego de caracteres es la ISO-8859-1. Son 256 caracteres que coinciden en codificación con los 256 primeros de Unicode.
  . Una codificación (*encoding*) de caracteres determina cómo se representan en bytes los puntos de código. Por ejemplo, el punto de código de la A, el 65, se puede codificar como un byte con signo, dos bytes sin signo en formato little-endian, 4 bytes...
  . UTF-8 (**U**nicode **T**ransformation **F**ormat de longitud variable de 8 bits) es un ejemplo de codificación de caracteres para el juego de caracteres Unicode.
  . El punto de código de la letra "A" en lenguas latinas es U+0041.
  . ASCII es un juego de caracteres y una codificación de caracteres. Unicode es un superconjunto de ASCII (lo contiene).

- Debe existir un único elemento raíz, que "cuelga" del nodo raíz (/). Este elemento tendrá como descendientes a todos los demás elementos. El documento XML bien formado más simple contendría sólo un elemento raíz, sin ningún descendiente.

- Los elementos que no sean vacíos deben tener una etiqueta de apertura y otra de cierre.

- Los elementos vacíos deber cerrarse con />. Ejemplo: `<br />`

- Los elementos deben aparecer correctamente anidados en cuanto a su apertura y su cierre, no solaparse. Es decir, los elementos se cierran en orden inverso al que se abren.

  Ejemplo:

  Se abre `<alfa>`, se abre `<beta>` y se abra `<gamma>`. Se cerrarán en orden inverso al de apertura. Primero se cierra `</gamma>`, luego `</beta>` y, por último, `</alfa>`.

  `<alfa> <beta> <gamma>...</gamma> </beta> </alfa>` ✓

  En este caso se ha cerrado </beta> antes que </gamma>.

  `<alfa> <beta> <gamma>...</beta> </gamma> </alfa>` ✗

- Los nombres de elementos y atributos son sensibles a mayúsculas/minúsculas.
- Los valores de los atributos deben aparecer entre comillas simples o dobles, pero del mismo tipo.

  Ejemplo:

  `<libro signatura="SIL001" />` ✓

  `<libro signatura='SIL001" />` ✗

- No puede haber dos atributos con el mismo nombre asociados al mismo elemento.
- No se pueden introducir ni instrucciones de procesamiento ni comentarios en ningún lugar del interior de las etiquetas de apertura y cierre de los elementos.

  Ejemplo:

  Un comentario incorrecto:

  `<persona <!-- comentario incorrecto --> >...</persona>` ✗

- No puede haber nada antes de la instrucción de procesamiento `<?xml ... ?>`.
- No puede haber texto antes ni después del elemento documento.
- No pueden aparecer los signos < ni & en el contenido textual de elementos ni atributos.

Una manera de verificar que un documento XML está bien formado es abriéndolo en un navegador como Mozilla Firefox o Internet Explorer. Si se muestra el árbol de nodos, significa que está bien formado.

```
- <correo>
    <de>José Ramón</de>
    <para>JuanMa</para>
    <asunto>Partido de rugby en TV</asunto>
  - <mensaje>
      <saludo>Hola Juanma:</saludo>
    - <cuerpo>
        El próximo sábado ponen en la tele el Francia-Inglaterra del seis naciones, ¿te apetece que lo veamos juntos? Espero tus noticias.
      </cuerpo>
      <despedida>Un abrazo.</despedida>
    </mensaje>
  </correo>
```

Figura 4.5: Visualización en un navegador de un documento XML

Actividad 4.3:

Identifica cuáles de los siguientes documentos están bien formados y cuáles no. Cuando no lo estén, explica el (los) motivo(s). Para comprobar el resultado se puede usar un editor XML, como el XML Copy Editor, para que valide por nosotros los documentos.

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

```
<documento>Texto de prueba</documento>
```

```
<?xml?>
<documento>Texto de prueba</documento>
```

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

```
<?XML version="1.0"?>
<Documento codigo="135">
  <nombre>Artículo</nombre>
  <amplitud>Media</amplitud>
</Documento>
```

```
<?xml version="1.0"?>
<El documento>
  <nombre>Artículo</nombre>
  <amplitud>Media</amplitud>
</El documento>
```

4.6. Documentos XML válidos

Hasta ahora se ha visto qué componentes forman un documento XML (elementos, atributos, comentarios...) y qué reglas deben de cumplir para que el documento sea bien formado, es decir, sea sintácticamente correcto.

Nada se ha dicho sobre qué elementos o atributos concretos pueden aparecer, en qué orden, qué valores pueden tomar, etc.

Si se hiciera esto, se estaría definiendo un lenguaje XML concreto, con su vocabulario (elementos y atributos permitidos), las relaciones entre elementos y atributos, los valores permitidos para ellos, etc.

Se dice que un documento XML es **válido** si existen unas reglas de validación (por ejemplo en forma de definición de tipo de documento (DTD)) asociadas a él, de manera que el documento deba cumplir con dichas reglas. Estas reglas especifican la estructura gramatical y sintaxis que debe tener el documento XML.

Todo documento XML válido está bien formado, pero no a la inversa.

Documento XML válido $\underset{\not\Leftarrow}{\Rightarrow}$ Documento XML bien formado

Ejemplo:

Para entender la diferencia entre un documento bien formado y un documento válido, veremos el equivalente con una frase en castellano.

montaña La nevada está ←→ La montaña está nevada

Ambas frases están **bien formadas** en castellano desde el punto de vista léxico, es decir, las palabras que en ellas aparecen son correctas en castellano. Ahora bien, desde el punto de vista gramatical, la primera se trata de una frase incorrecta (no se cumple la clásica regla de sujeto + predicado). Así, sólo la segunda frase es **válida** con respecto a la gramática castellana.

4.7. Validación de documentos XML con DTD

Un DTD (**D**ocument **T**ype **D**efinition – Definición de Tipo de Documento) es una descripción de la estructura de un documento XML. Se especifica qué elementos tienen que aparecer, en qué orden, cuáles son optativos, cuáles obligatorios, qué atributos tienen los elementos, etc.

Es un mecanismo de validación de documentos que existía antes de la aparición de XML. Se usaba para validar documentos SGML (**S**tandard **G**eneralizad **M**arkup **L**anguage – Lenguaje de Marcas eStándar Generalizado), que es el precursor de todos los lenguajes de marcas actuales. Cuando apareció XML, se integró en su especificación como modelo de validación gramatical.

En el momento de la publicación de este libro, la técnica de validación con DTDs ha quedado algo obsoleta, si bien se usó de manera intensiva en los años 1990 y principios de los 2000.

Actualmente, se usan más otras técnicas como los esquemas XML, también conocidos por las siglas XSD (**X**ML **S**chema **D**ocument – Documento de Esquema XML). Sin embargo, muchos documentos han quedado validados únicamente mediante DTDs.

Ejemplo:

DTD estricto y DTD transicional para HTML 4.01

- http://www.w3.org/TR/html4/sgml/dtd.html
- http://www.w3.org/TR/html4/sgml/loosedtd.html

Generación automática de DTDs

Cuando los documentos XML contienen muchos datos, los DTD que los validan pueden ser muy extensos y su escritura tediosa. Para facilitar este proceso, existen herramientas de generación automática de DTDs a partir de un documento XML, proceso conocido como inferencia. Se detallará su uso más adelante.

Estas herramientas realizan deducciones "gruesas" de las reglas gramaticales, que el desarrollador debe después afinar para que se ajusten a los requisitos semánticos.

- Generadores on-line:

http://www.hitsw.com/xml_utilites/default.html

- Generadores instalables:
  - Trang (http://www.thaiopensource.com/download): es un programa desarrollado en Java, ejecutable desde la línea de comandos, que genera un DTD a partir de un XML dado.
- Editores XML que generan un DTD a partir de un documento XML. Prácticamente todos los citados anteriormente:
  - XMLSpy de Altova
  - <oXygen/>, de Syncro Soft
  - XMLPad Pro Edition, de WMHelp

4.7.1. Estructura de un DTD. Elementos

Al igual que ya sucedía con las hojas de estilo en cascada (CSS), el DTD puede aparecer integrado en el propio documento XML, en un archivo independiente (habitualmente con extensión `.dtd`) e incluso puede tener una parte interna y otra externa. La opción del documento externo permitirá reutilizar el mismo DTD para distintos documentos XML, facilitando las posibles modificaciones de una manera centralizada.

Siempre que se quiera declarar un DTD, se hará al comienzo del documento XML. Las reglas que lo constituyen son las que podrán aparecer a continuación de la declaración (dentro del propio documento XML) o en un archivo independiente.

Declaración del DTD:

<!DOCTYPE>

Es la instrucción donde se indica qué DTD validará el XML. Aparece al comienzo del documento XML. El primer dato que aparece es el nombre del elemento raíz del documento XML.

En función del tipo de DTD la sintaxis varía. Las características que definen el tipo son:

- Ubicación: dónde se localizan las reglas del DTD.
  - Interno: las reglas aparecen en el propio documento XML.
  - Externo: las reglas aparecen en un archivo independiente.
  - Mixto: mezcla de los anteriores, las reglas aparecen en ambos lugares. Las reglas internas tienen prioridad sobre las externas.
- Carácter: si es un DTD para uso privado o público.
  - Para uso privado: se identifica por la palabra SYSTEM.
  - Para uso público: se identifica por la palabra PUBLIC. Debe ir acompañado del **FPI** (**F**ormal **P**ublic **I**dentifier – Identificador Público Formal), una etiqueta que identifica al DTD de manera "universal".

Las distintas combinaciones son:

Sintaxis	Tipo de DTD
`<!DOCTYPE elemento_raíz [reglas]>`	Interno (luego privado)
`<!DOCTYPE elemento_raíz SYSTEM URL>`	Externo y privado
`<!DOCTYPE elemento_raíz SYSTEM URL [reglas]>`	Mixto y privado
`<!DOCTYPE elemento_raíz PUBLIC FPI URL>`	Externo y público
`<!DOCTYPE elemento_raíz PUBLIC FPI URL [reglas]>`	Mixto y público

Tabla 4.1: Diferentes sintaxis para DOCTYPE

La combinación interno y público no tiene sentido, ya que el hecho de ser público fuerza que el DTD esté en un documento independiente.

Atributos:

- Nombre del elemento raíz del documento XML: aparece justo a continuación de la palabra DOCTYPE.
- Declaración de privacidad/publicidad: aparece a continuación del nombre del elemento raíz e indica si el DTD es de uso público (PUBLIC) o de uso interno de la organización que lo desarrolla (SYSTEM).
- Identificador: sólo existe cuando el DTD es PUBLIC e indica el FPI por el que se conoce el DTD.

Estructura del FPI:

Está compuesto de 4 campos separados por el carácter //.

· Primer campo: norma formal o no formal

- Si el DTD no ha sido aprobado por una norma formal, como por ejemplo un DTD escrito por uno mismo, se escribe un signo menos (-).
- Si ha sido aprobado por un organismo no oficial, se escribe un signo más (+).
- Si ha sido aprobado por un organismo oficial, se escribe directamente una referencia al estándar.

· Segundo campo: nombre del organismo responsable del estándar.

· Tercer campo: tipo del documento que se describe, suele incluir un número de versión.

· Cuarto campo: idioma del DTD.

Ejemplo:

Se va a declarar un DTD respecto a un documento XML cuyo elemento raíz se llama `html` y que es de carácter público.

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

❶ Al aparecer un signo menos (-) significa que el DTD no ha sido aprobado por una norma formal.

❷ El nombre del organismo responsable del DTD es el W3C.

❸ El tipo de documento es XHTML Transicional, en su versión 1.0.

❹ El idioma del DTD es el inglés (EN).

- url: sólo existe cuando el DTD (en parte o en su totalidad) se encuentra declarado en un archivo externo, del que da su ubicación. Como ya se ha comentado, puede darse el caso que un DTD esté definido en parte en un archivo externo y en parte en el documento XML.

Ejemplo:

Se declara el tipo de un documento XML cuyo elemento raíz se llama `<planetas>`. Se trata de un DTD de uso privado (SYSTEM) y la ubicación de las reglas del DTD es externa, en un archivo llamado `planetas.dtd` que se encuentra en el mismo directorio que el archivo XML.

```
<!DOCTYPE planetas SYSTEM "planetas.dtd">
```

Ejemplo:

Un DTD mixto y privado tendrá sus reglas repartidas entre la cabecera del documento XML y un archivo externo.

El documento XML, *charlas.xml*, será:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE charlas SYSTEM "charlas.dtd" [
  <!ELEMENT charlas (charla)+>
  <!ELEMENT charla (nombre, lugar, despedida)>
  <!ENTITY despedidaInglesa "Thank you, good bye!">
  <!ENTITY despedidaFrancesa "Merci, au revoir!">
]>
<charlas>
  ...
</charlas>
```

El archivo de reglas externo, *charlas.dtd*, será:

```
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT lugar (#PCDATA)>
<!ELEMENT despedida (#PCDATA)>
```

Nótese que el archivo externo no incluye la declaración del tipo de documento (DOCTYPE), puesto que sólo aparece en la cabecera del documento XML.

Conviene resaltar también que el atributo `standalone` de la instrucción de procesamiento `<?xml ... ?>` tiene el valor *no*, lo que significa que para el correcto procesado del documento necesitamos del uso de otros documentos externos (en este caso, de `charlas.dtd`).

Componentes del DTD:

Hay cuatro tipos posibles de componentes que se pueden declarar en un DTD:

- Elemento
- Atributo
- Entidad
- Notación

<!ELEMENT>

Es una declaración de tipo de elemento. Indica la existencia de un elemento en el documento XML.

Sintaxis general: `<!ELEMENT nombre_elemento modelo_contenido>`

El nombre del elemento tiene que ser el mismo que el correspondiente del documento XML, sin los signos de menor (<) y mayor (>) propios de las etiquetas.

El modelo de contenido puede indicar:

- Una regla, en cuyo caso será:

  · `ANY` (cualquier cosa): se puede utilizar al construir el DTD para dejar la descripción de un elemento como válida en cualquier caso, eliminando cualquier comprobación sintáctica. Es un comodín que no debe aparecer en el DTD definitivo.

- EMPTY (elemento vacío): describe un elemento sin descendientes.

Ejemplo:

La descripción del elemento <br /> de HTML será:

```
<!ELEMENT br EMPTY>
```

- Datos (caracteres), sean textuales, numéricos o cualquier otro formato que no contenga marcas (etiquetas). Se describe como #PCDATA (Parsed Character **Data** – Datos de Caracteres Procesados) y debe aparecer entre paréntesis.

Ejemplo:

Una regla de un DTD puede ser:

```
<!ELEMENT titulo (#PCDATA)>
```

Que se corresponde con un elemento llamado <titulo> con contenido textual.

```
<titulo>Lenguajes de marcas</titulo>
```

- Elementos descendientes. Su descripción debe aparecer entre paréntesis y se basa en las siguientes reglas:

Cardinalidad de los elementos:

Para indicar el número de veces que puede aparecer un elemento o una secuencia de elementos existen ciertos símbolos:

Símbolo	Significado
?	El elemento (o secuencia de elementos) puede aparecer 0 o 1 vez
*	El elemento (o secuencia de elementos) puede aparecer de 0 a N veces
+	El elemento (o secuencia de elementos) puede aparecer de 1 a N veces

Secuencias de elementos:

En las secuencias de elementos se utilizan símbolos para indicar el orden en que un elemento debe aparecer, o bien si aparece como alternativa a otro:

Símbolo	Significado
A, B	El elemento B aparecerá a continuación del elemento A
A | B	Aparecerá el elemento A o el B, pero no ambos

Se pueden combinar el uso de los símbolos de cardinalidad de los elementos con los de la secuencias de elementos.

Ejemplo:

En un correo electrónico, se podría describir el elemento raíz <email> como una secuencia de elementos <para>, <cc> (optativo), <cco> (optativo), <asunto> y <cuerpo>. Las reglas que representan esto serán:

```
<!ELEMENT email (para, cc?, cco?, asunto, cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT cc (#PCDATA)>
...
```

Ejemplo:

Un contrato tiene una lista de cláusulas. Cada una de las cláusulas está compuesta de varios epígrafes y sus desarrollos asociados, y concluyen por un único epílogo:

```
<!ELEMENT clausulas (clausula)+>
<!ELEMENT clausula ((epígrafe, desarrollo)+, epilogo)>
...
```

- Contenido mixto, mezcla de texto más elementos descendientes.

Ejemplo:

Se quiere describir un elemento `<párrafo>` que simule el párrafo de un editor de textos, de forma que pueda contener texto sin formato, texto en `<negrita>` (rodeado de esta etiqueta) o texto en `<cursiva>` (rodeado de esta etiqueta). Estas dos últimas etiquetas podrán a su vez contener, bien texto sin formato, bien la otra etiqueta. Por tanto, `<párrafo>`, `<negrita>` y `<cursiva>` son elementos de contenido mixto. Las reglas que describen esto serán:

```
<!ELEMENT parrafo (negrita | cursiva | #PCDATA)*>
<!ELEMENT negrita (cursiva | #PCDATA)*>
<!ELEMENT cursiva (negrita | #PCDATA)*>
```

❗ NOTA: No se puede usar el cuantificador + con un contenido mixto, por eso se ha usado el *.

Un documento XML válido con respecto a las reglas del anterior DTD:

```
<parrafo>
  Aquí un <cursiva>tema <negrita>importante</negrita></cursiva>
</parrafo>
```

Actividad 4.4:

Se quiere que el elemento `<grupoSanguineo>` tenga como único elemento descendiente a uno solo de los cuatro siguientes A, B, AB ó O. Indica cuál de las siguientes es una declaración correcta del citado elemento.

a. `<!ELEMENT grupoSanguineo (A ? B ? AB ? O) >`

b. `<!ELEMENT grupoSanguineo (A , B , AB , O) >`

c. `<!ELEMENT grupoSanguineo (A | B | AB | O) >`

d. `<!ELEMENT grupoSanguineo (A + B + AB + O) >`

Actividad 4.5:

¿Cuál de las siguientes afirmaciones es correcta respecto a la declaración del elemento?

a. `<!ELEMENT contenido (alfa | beta*) >`

Tanto el elemento alfa como el elemento beta pueden aparecer 0 o más veces como descendientes del elemento contenido.

b. `<!ELEMENT contenido (alfa , beta) >`

Tanto el elemento alfa como el elemento beta pueden aparecer una vez como descendientes del elemento contenido, sea cual sea el orden.

c. `<!ELEMENT contenido (alfa | beta) >`

El elemento alfa y el elemento beta deben aparecer una vez cada uno como descendientes del elemento contenido.

d. `<!ELEMENT contenido (alfa , beta*) >`

El elemento alfa debe aparecer una vez y a continuación el elemento beta debe aparecer 0 o más veces, ambos como descendientes del elemento contenido.

NOTA: Los documentos HTML no necesitan estar bien formados para poderse visualizar en los navegadores. Por eso, en los DTD que validan estos documentos se usa una determinada nomenclatura para indicar que un elemento pueda ser opcionalmente abierto o cerrado. Así, si aparecen dos guiones después del nombre del elemento tanto la etiqueta inicial como la final son obligatorias. Un guion seguido por la letra "O" indica que puede omitirse la etiqueta final. Un par de letras "O" indican que tanto la etiqueta inicial como la final pueden omitirse.

<!ATTLIST>

Es una declaración de tipo de atributo. Indica la existencia de atributos de un elemento en el documento XML. En general se utiliza un solo `ATTLIST` para declarar todos los atributos de un elemento (aunque se podría usar un `ATTLIST` para cada atributo).

Sintaxis general:
```
<!ATTLIST nombre_elemento
                    nombre_atributo tipo_atributo caracter
                    nombre_atributo tipo_atributo caracter

          >
```

El **nombre** del atributo tiene que ser un nombre XML válido.

El **carácter** del atributo puede ser:

- un valor textual entre comillas, que representa un valor por defecto para el atributo.
- `#IMPLIED`, el atributo es de carácter opcional y no se le asigna ningún valor por defecto.

- `#REQUIRED`, el atributo es de carácter obligatorio, pero no se le asigna un valor por defecto.
- `#FIXED`, el atributo es de carácter obligatorio y se le asigna un valor por defecto que además es el único valor que puede tener el atributo.

Los posibles **tipos de atributo** son:

- `CDATA`: caracteres que no contienen etiquetas
- `ENTITY`: el nombre de una entidad (que debe declararse en el DTD)
- `ENTITIES`: una lista de nombres de entidades (que deben declararse en el DTD), separadas por espacios
- Enumerado: una lista de valores de entre los cuales, el atributo debe tomar uno

  Ejemplo:

  Se quiere definir una regla que valide la existencia de un elemento `<semaforo>`, de contenido vacío, con un único atributo `color` cuyos posibles valores sean rojo, naranja y verde. El valor por defecto será `verde`.

```
<!ELEMENT semaforo EMPTY>
<!ATTLIST semaforo
          color (rojo | naranja | amarillo) "verde">
```

- `ID`: un identificador único. Se usa para identificar elementos, es decir, caracterizarlos de manera única. Por ello, dos elementos no pueden tener el mismo valor en atributos de tipo ID. Además, un elemento puede tener a lo sumo un atributo de tipo ID. El valor asignado a un atributo de este tipo debe ser un nombre XML válido.
- `IDREF`: representa el valor de un atributo ID de otro elemento, es decir, para que sea válido, debe existir otro elemento en el documento XML que tenga un atributo de tipo ID y cuyo valor sea el mismo que el del atributo de tipo IDREF del primer elemento.

  Ejemplo:

  Se quiere representar un elemento `<empleado>` que tenga dos atributos, `idEmpleado` e `idEmpleadoJefe`. El primero será de tipo `ID` y carácter obligatorio, y el segundo de tipo `IDREF` y carácter optativo.

```
<!ELEMENT semaforo (nombre, apellido)>
<!ATTLIST empleado
            idEmpleado ID #REQUIRED
            idEmpleadoJefe IDREF #IMPLIED>
...
```

Aquí tenemos un fragmento de XML válido con respecto a las reglas recién definidas. Cada elemento `<empleado>` tiene un atributo `idEmpleado`, con un valor válido (nombre XML válido) y el segundo elemento `<empleado>` tiene también un atributo `idEmpleJefe`, cuyo valor ha de ser el mismo que el del atributo de tipo ID de otro

elemento existente en el documento. En este caso, este atributo `idEmpleJefe` del segundo `<empleado>` vale igual que el atributo `idEmpleado` del primer `<empleado>`:

```
<empleados>
  <empleado idEmpleado="e_111">...</empleado>
  <empleado idEmpleado="e_222" idEmpleadoJefe="e_111">      ✓
    ...
  </empleado>
</empleados>
```

Un fragmento de XML no válido con respecto a las reglas recién definidas sería aquél en el cual el atributo `idEmpleadoJefe` del primer `<empleado>` tenga un valor que no exista para ningún atributo de tipo ID de otro elemento del documento:

```
<empleados>
  <empleado idEmpleado="e_111" idEmpleadoJefe="e_333">      ✗
    ...
  </empleado>
  <empleado idEmpleado="e_222" idEmpleadoJefe="e_111">
    ...
  </empleado>
</empleados>
```

- `IDREFS`: representa múltiples IDs de otros elementos, separados por espacios.
- `NMTOKEN`: cualquier nombre sin espacios en blanco en su interior. Los espacios en blanco anteriores o posteriores se ignorarán.

Ejemplo:

Se quiere declarar un atributo de tipo `NMTOKEN` de carácter obligatorio.

En el siguiente DTD, la declaración del elemento `<rio>` y su atributo `pais` es:

```
<!ELEMENT rio (nombre)>
<!ATTLIST rio pais NMTOKEN #REQUIRED>
```

En el siguiente fragmento XML, el valor del atributo `pais` es válido con respecto a la regla anteriormente definida:

```
<rio pais="EEUU">                                           ✓
  <nombre>Misisipi</nombre>
</rio>
```

En el siguiente documento XML, el valor del atributo `pais` no es válido con respecto a la regla anteriormente definida por contener espacios en su interior:

```
<rio pais="Estados Unidos">                                 ✗
  <nombre>Misisipi</nombre>
</rio>
```

- NMTOKENS: una lista de nombres, sin espacios en blanco en su interior (los espacios en blanco anteriores o posteriores se ignorarán), separados por espacios.
- NOTATION: un nombre de notación, que debe estar declarada en el DTD.

<!ENTITY>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración de tipo de entidad. Hay diferentes tipos de entidades y, en función del tipo, la sintaxis varía:

- Referencia a entidades generales (internas o externas).
- Referencia a entidades parámetro (internas o externas).
- Entidades no procesadas (*unparsed*).

- Referencia a entidades generales, se utilizarán dentro del documento XML.

Sintaxis general: `<!ENTITY nombre_entidad definición_entidad>`

Ejemplo:

En primer lugar, se declara una entidad en el DTD

```
<!ENTITY rsa "República Sudafricana">
```

A continuación, se usa en el XML anteponiendo al nombre de la entidad el carácter ampersand (&) y a continuación un carácter punto y coma (;). El programa analizador del documento realizará la sustitución.

```
<país>
  <nombre>&rsa;</nombre>
  ...
</país>
```

Ejemplo:

En el DTD de HTML se declaran diversas entidades para referenciar caracteres con acentos diversos, como é para referenciar el carácter é.

- Referencia entidades generales externas, ubicadas en otros archivos:

Sintaxis general: `<!ENTITY nombre_entidad tipo_uso url_archivo>`

Siendo el tipo de uso privado (SYSTEM) o público (PUBLIC).

Ejemplo:

Se dispone de un archivo de texto, *autores.txt*, que contiene el siguiente texto plano "Juan Manuel y José Ramón".

Se crea un documento XML que hace referencia a ese archivo de texto, en forma de entidad externa. Al visualizar el documento, la referencia a la entidad general externa se sustituirá por el texto contenido en el archivo.

```
<?xml version="1.0"?>
<!DOCTYPE escritores [
  <!ELEMENT escritores (#PCDATA)>
  <!ENTITY autores SYSTEM "autores.txt">
]>
<escritores>&autores;</escritores>
```

- Referencia a entidades parámetro, que se usarán en el propio DTD y funcionan cuando la definición de las reglas del DTD se realiza en un archivo externo.

Sintaxis general: `<!ENTITY % nombre_entidad definición_entidad>`

Ejemplo:

Se declara una entidad parámetro `dimensiones` y se referencia dentro del propio DTD.

```
<!ENTITY % dimensiones "alto CDATA #IMPLIED ancho CDATA #IMPLIED
profundo CDATA #IMPLIED">
<!ELEMENT objeto (nombre)>
<!ATTLIST objeto
            codigo ID #REQUIRED
            %dimensiones;>
...
```

Este código es equivalente a haber escrito:

```
<!ELEMENT objeto (nombre)>
<!ATTLIST objeto
            nombre CDATA #REQUIRED
            alto CDATA #IMPLIED
            ancho CDATA #IMPLIED
            profundo CDATA #IMPLIED>
...
```

- Referencia entidades parámetro externas, ubicadas en otros archivos:

Sintaxis general: `<!ENTITY % nombre_entidad tipo_uso fpi url_archivo>`

Siendo el tipo de uso privado (SYSTEM) o público (PUBLIC). Si es PUBLIC, es necesario definir el FPI que, recordemos, es el nombre por el cual se identifica públicamente un determinado elemento (sea un DOCTYPE, un ELEMENT o una ENTITY), en este caso la entidad.

- Entidades no procesadas, referencian a datos que no deben ser procesados por el analizador XML, sino por la aplicación que lo use (como una imagen).

Sintaxis general: `<!ENTITY % nombre_entidad tipo_uso fpi valor_entidad NDATA tipo>`

Ejemplo:

Se declara una notación (se comentará más adelante) de nombre JPG para el tipo **MIME** (**M**ultipurpose **I**nternet **M**ail **E**xtensions - Extensiones Multipropósito de Correo de Internet). Se declara una entidad no procesada de nombre `mediterraneo`, asociada al archivo de

imagen *mediterraneo.jpg*. Por último, se declara un elemento `<mar>`, que cuenta con atributo `imagen` que es del tipo `ENTITY` recién declarado.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg" NDATA JPG>
<!ELEMENT mar ... >
<!ATTLIST mar
  imagen ENTITY #IMPLIED>
```

Y en el XML, el valor del atributo `imagen` del elemento `<mar>` es la entidad no procesada declarada en el DTD:

```
<mares>
  <mar imagen="mediterraneo">
    <nombre>Mediterráneo</nombre>
  </mar>
      ...
</mares>
```

NOTA: En ocasiones se utiliza otra tecnología, llamada XLink, en sustitución de las entidades no procesadas.

Ejemplo:

Una extensión del ejemplo anterior sería el permitir incluir múltiples imágenes como valor del atributo `imagen` del elemento `<mar>`.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo1 SYSTEM "mediterraneo1.jpg" NDATA JPG>
<!ENTITY mediterraneo2 SYSTEM "mediterraneo2.jpg" NDATA JPG>
<!ELEMENT mar ... >
<!ATTLIST mar
            imagen ENTITIES #IMPLIED>
```

Y en el documento XML, el valor del atributo `imagen` del elemento `<mar>` son las dos entidades no procesadas declaradas en el DTD:

```
<mares>
  <mar imagen="mediterraneo1 mediterraneo2">
    <nombre>Mediterráneo</nombre>
  </mar>
      ...
</mares>
```

<!NOTATION>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración del tipo de atributo `NOTATION`. Una notación se usa para especificar un formato de datos que no sea XML. Se usa con frecuencia para describir tipos MIME, como *image/gif* o *image/jpg*.

Se utiliza para indicar un tipo de atributo al que se le permite usar un valor que haya sido declarado como notación en el DTD.

Sintaxis general de la notación:

```
<!NOTATION  nombre_notación SYSTEM "identificador_externo">
```

Sintaxis general del atributo que la usa:

```
<!ATTLIST  nombre_elemento
           nombre_atributo  NOTATION valor_defecto>
```

Ejemplo:

Se declaran tres notaciones que corresponden a otros tantos tipos MIME de imágenes (gif, jpg y png). También se declara un elemento `<mar>` y sus atributos `imagen` y `formato_imagen`, este último referenciando a las notaciones recién creadas. Por último, se declara una entidad no procesada que se asocia a un archivo de imagen.

```
<!NOTATION GIF SYSTEM "image/gif">
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION PNG SYSTEM "image/png">
<!ELEMENT mar ... >
<!ATTLIST mar
          imagen ENTITY #IMPLIED
          formato_imagen NOTATION (GIF | JPG | PNG) #IMPLIED>
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg">
```

Y en el documento XML, el valor del atributo `imagen` del elemento `<mar>` es la entidad no procesada declarada en el DTD, y el valor del atributo `formato_imagen` el de una de sus alternativas válidas, la notación JPG:

```
<mares>
  <mar imagen="mediterraneo" formato_imagen="JPG">
    <nombre>Mediterráneo</nombre>
  </mar>
    ...
</mares>
```

Secciones condicionales

Permiten incluir o excluir reglas en un DTD en función de condiciones. Sólo se pueden ubicar en DTDs externos. Su uso tiene sentido al combinarlas con referencias a entidades parámetro. Las secciones condicionales son `IGNORE` e `INCLUDE`, teniendo la primera precedencia sobre la segunda.

Ejemplo:

Supónganse dos estructuras diferentes para un mensaje:

- una sencilla que incluye emisor, receptor y contenido
- otra extendida que incluye los datos de la estructura sencilla más el título y el número de palabras.

Se quiere diseñar un DTD que, en función del valor de una entidad parámetro, incluya una estructura de mensaje o la otra. Por defecto se incluirá la larga.

El DTD, que ha de ser externo, tendrá la siguiente estructura:

```
<!-- Mensaje corto -->
<![IGNORE[
   <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo -->
<![INCLUDE[
   <!ELEMENT mensaje (titulo, emisor, receptor, contenido, palabras)>
   <!ELEMENT titulo (#PCDATA)>
   <!ELEMENT palabras (#PCDATA)>
]]>

<!-- Declaración de elementos y atributos comunes -->
<!ELEMENT emisor (#PCDATA)>
<!ELEMENT receptor (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>
```

Al añadir las referencias a entidad parámetro, el DTD quedaría:

```
<!ENTITY %corto "IGNORE">
<!ENTITY %largo "INCLUDE">

<!-- Mensaje corto -->
<![%corto[
   <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo -->
<![%largo[
   <!ELEMENT mensaje (titulo, emisor, receptor, contenido, palabras)>
   <!ELEMENT titulo (#PCDATA)>
   <!ELEMENT palabras (#PCDATA)>
]]>
<!-- Declaración de elementos y atributos comunes -->
<!ELEMENT emisor (#PCDATA)>
<!ELEMENT receptor (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>
```

Para cambiar el tipo de mensaje que se va a incluir, basta con modificar la asociación de valores de `%corto` a `INCLUDE` y de `%largo` a `IGNORE`, de la forma:

```
<!ENTITY %corto "INCLUDE">
<!ENTITY %largo "IGNORE">
```

Se podrían dejar la declaración de las entidades en la DTD interna al documento XML, donde se determinaría qué tipo de mensaje se quiere incluir de manera específica para ese documento:

```
<?xml verion="1.0"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd" [
  <!ENTITY %corto "INCLUDE">
  <!ENTITY %largo "IGNORE">
]>
```

```
<mensaje>
  ...
</mensaje>
```

4.7.2. Poniendo todo junto

Se quiere construir un DTD que valide el siguiente documento XML, *persona.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE persona SYSTEM "persona.dtd"]>
<persona dni="12345678-L" estadoCivil="Casado">
  <nombre>María Pilar</nombre>
  <apellido>Sánchez</apellido>
  <edad>60</edad>
  <enActivo/>
</persona>
```

El esquema XML asociado se quiere que cumpla ciertas restricciones semánticas:

- El atributo `dni` es un identificador obligatorio.
- El estado civil puede ser: *Soltero*, *Casado* o *Divorciado*. Por defecto es *Soltero*.
- El elemento `<enActivo>` es optativo.

El DTD que cumple con esto, ubicado en el archivo *persona.dtd*, es:

```
<!ELEMENT persona (nombre, apellido, edad, enActivo?)>
<!ATTLIST persona dni ID #REQUIRED
          estadoCivil (Soltero | Casado | Divorciado) "Soltero">
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT edad (#PCDATA)>
<!ELEMENT enActivo (#PCDATA) EMPTY>
```

Herramientas de generación automática de DTDs

Va a haber muchos documentos XML que puedan ser válidos por un mismo DTD. Imagínese simplemente un DTD que valide documentos XML que simulen un correo electrónico básico, es decir, que contengan elementos `<para>`, `<cc>` (optativo), `<cco>` (optativo), `<asunto>` (optativo) y `<cuerpo>`.

Cualquier correo que cumpla con estas reglas será un documento válido con respecto a este DTD. Por ejemplo, igualmente válido sería un documento XML que contenga el elemento `<cc>` que el que no lo contenga, puesto que es optativo.

La cuestión aquí es que se va a tratar de inferir, a partir de un documento XML (que es un ejemplo concreto válido respecto a un DTD), el DTD genérico que lo valide.

Por ello, lo que se generará es un DTD que valide exactamente el XML que lo generó, pero si las reglas de negocio imponen restricciones adicionales, la herramienta de generación automática no será capaz de deducirlas, lo que nos obligará a añadirlas manualmente al DTD generado.

Entiéndase por **regla de negocio** a una restricción o característica propia del ámbito en el que se trabaja. Por ejemplo, una regla de negocio podría ser que un pago se haga en euros o dólares.

Ejemplo:

Se quiere generar un DTD a partir de un XML en el que hay un elemento `<distancia>`, el cual posee un atributo `unidad`, y se quiere que el valor de dicho atributo pueda ser *centímetro*, *metro* o *kilómetro*.

No hay manera de indicárselo al programa de generación automática. Éste sólo podrá suponer que existe un atributo `unidad`, perteneciente al elemento `distancia`, que contiene texto. Nada más. La enumeración de los posibles valores y, si se diera el caso, la existencia de uno de ellos por defecto, tendrá que indicarse "a mano" sobre el DTD generado.

A pesar de todo, son herramientas prácticas desde el punto de vista que realizan el trabajo mecánico inicial y, sobre el DTD que generan, resulta más cómodo introducir los ajustes necesarios para cumplir las restricciones semánticas (reglas de negocio).

NOTA: Para lograr una inferencia lo más precisa posible, se recomienda escribir un documento XML lo más completo posible, sin omitir elementos opcionales, ni atributos, etc.

Limitaciones de los DTD

Algunas limitaciones de los DTD son:

1. Un DTD no es un documento XML, luego no se puede verificar si está bien formado.
2. No se pueden fijar restricciones sobre los valores de elementos y atributos, como su tipo de datos, su tamaño, etc.
3. No soporta espacios de nombres.
4. Sólo se puede enumerar los valores de atributos, no de elementos.
5. Sólo se puede dar un valor por defecto para atributos, no para elementos.
6. Existe un control limitado sobre las cardinalidades de los elementos, es decir, concretar el número de veces que pueden aparecer.

4.8. Validación de documentos XML con esquemas XML

Un esquema XML es un mecanismo para comprobar la validez de un documento XML. Se trata de una forma alternativa a los DTD, pero con ciertas ventajas sobre éstos:

- Es un documento XML, por lo que se puede comprobar si está bien formado.
- Existe un extenso catálogo de tipos de datos predefinidos para elementos y atributos que pueden ser ampliados o restringidos para crear nuevos tipos.
- Permiten concretar con precisión la cardinalidad de un elemento, es decir, las veces que puede aparecer en un documento XML.
- Permite mezclar distintos vocabularios (juegos de etiquetas) gracias a los espacios de nombres.

Como característica negativa, decir que los esquemas XML son más difíciles de interpretar por el ojo humano que los DTD.

En un esquema XML se describe lo que un documento XML puede contener: qué elementos, con qué atributos, de qué tipos de datos son tanto elementos como atributos, en qué orden aparecen los elementos, cuántas ocurrencias puede haber de cada elemento, etc.

Los documentos XML que se validan contra un esquema se llaman **instancias del esquema**. De hecho, se puede ver el esquema como un molde, y los documentos XML como objetos que tratan de ajustarse a ese molde. También se puede hacer una analogía con la programación orientada a objeto, los esquemas serían como las clases y los documentos XML como los objetos.

NOTA: Se pueden definir esquemas muy restrictivos o poco restrictivos. Los primeros fuerzan a los documentos XML validados a ser más precisos.

Herramientas para validar esquemas

Todas las herramientas ya vistas que comprueban si un documento XML está bien formado son capaces de validar el documento frente a un esquema XML.

Por citar una herramienta de validación on-line:

- http://www.corefiling.com/opensource/schemaValidate.html

Herramientas para generar esquemas automáticamente a partir de documentos XML

Sucede aquí lo mismo que con la generación automática de DTDs a partir de un documento XML: el generador no es capar de conocer las reglas de negocio, sólo de crear un armazón básico a partir del documento XML.

- Herramientas de línea de comandos:
    - Trang (http://www.thaiopensource.com/relaxng/trang.html): una herramienta de software libre instalable que permite inferir un esquema XML a partir de un documento XML. Igualmente, permite convertir esquemas XML en DTD y viceversa (así como entre otros mecanismos de validación de documentos XML, como RELAX NG).
- Editores de XML con esta funcionalidad:
    - XMLSpy de Altova.
    - <oXygen/> de
- Herramientas on-line:
    - http://www.flame-ware.com/products/xml-2-xsd
    - http://www.freeformatter.com/xsd-generator.html

4.8.1. Estructura de un esquema XML. Componentes

Un esquema XML es un documento XML que ha de cumplir una serie de reglas:

- Su elemento raíz se llama `<schema>`.
- Su espacio de nombres debe ser `http://www.w3.org/2001/XMLSchema`. Se podría no definir un prefijo o usar uno de los más comunes: `xs` o `xsd`. En este libro se usará `xs`.

  Así, el esquema XML más sencillo será:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

- Puesto que un documento XML bien formado contiene al menos un elemento raíz, el esquema XML dispondrá de, al menos, un componente de declaración de elemento que defina el elemento raíz del documento XML. Este componente de declaración de elemento se identifica con el elemento `<xs:element>`, que tendrá un atributo `name` cuyo valor será el nombre del elemento raíz del documento XML.

  Ejemplo:

  Un documento XML muy simple:

```
<?xml version="1.0"?>
<simple>Es difícil escribir un documento más simple</simple>
```

  Un posible esquema que lo valida:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="simple" />
</xs:schema>
```

  NOTA: Este esquema analizará como válido cualquier documento XML cuyo elemento raíz sea `<simple>`. De esta manera se ha obtenido un esquema XML completamente laxo en la validación de documentos XML. Cuantas más reglas y restricciones le añadamos, más restrictivo (y por tanto preciso) será.

- Puede haber un esquema con múltiples elementos raíz.

  Ejemplo:

  Un esquema con dos elementos raíz declarados, `<simple>` y `<complejo>`. Cualquier documento XML cuyo elemento raíz sea bien simple, bien complejo, se considerará conforme con este esquema.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="simple" />
  <xs:element name="complejo" />
</xs:schema>
```

- Un esquema es un conjunto de declaraciones de elementos y atributos y definición de tipos que sirve para fijar la estructura que deben tener sus documentos instancia XML. El orden en que se declaran los elementos, llamados **componentes**, en un esquema no es significativo ni afecta al funcionamiento del mismo.
- La vinculación de un esquema al documento XML se realiza en el documento XML, insertando `xmlns:xsi` y `xsi:noNamespaceSchemaLocation` como atributos del elemento raíz, en el caso de esquemas no asociados a espacios de nombres. Si el esquema estuviera asociado a un espacio de nombres se usaría el atributo `xsi:schemaLocation`. En este libro se usará sobre todo el primer caso.

  Ejemplo:

  En el documento XML se declara un espacio de nombres de prefijo `xsi`, propio de las instancias de esquemas XML, y se indica la ubicación del documento del esquema, en este caso de nombre *simple.xsd*.

```
<?xml version="1.0"?>
<simple xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="simple.xsd">
  ...
</simple>
```

Atributos de los documentos instancia del esquema o documento XML

En la terminología de los esquemas, los documentos XML son instancias de los mismos, al fin y al cabo, son objetos salidos del molde que es el esquema.

Por eso, en los documentos XML aparece un espacio de nombres que tiene sentido cuando se trabaja con esquemas asociados. Se ha visto en el elemento raíz del documento XML del ejemplo anterior, la declaración del espacio de nombres **`http://www.w3.org/2001/XMLSchema-instance`** asociada al prefijo **`xsi`**.

A continuación, y en el mismo elemento raíz, se declara un atributo de ese espacio de nombres, **`xsi:noNamespaceSchemaLocation="simple.xsd"`**, que permite vincular el documento XML con un esquema de nombre *simple.xsd*.

Existen otros atributos de este espacio de nombres que se usan ocasionalmente en documentos XML (o instancias de esquema). Son:

- `xsi:nil`: asociado a un elemento indica que debe ser vacío de contenido. Puede valer *false* o *true*. Por defecto vale ***false***, y para poderlo poner a *true*, se debe activar el atributo `nillable` en la definición del elemento en el esquema asociado. Se verá más adelante.

  Ejemplo:

  En el esquema se declararía un elemento de la forma:

```
<xs:element name="fechaCompra" type="xs:date" nillable="true"/>
```

  En el documento instancia XML se explicitaría que el contenido de `<fechaCompra>` debe ser vacío:

```
<fechaCompra xsi:nil="true"></fechaCompra>
```

- `xsi:type`: puede ser la definición de un tipo de un elemento, como `xs:string` o existente en un esquema asociado.
- `xsi:schemaLocation`: localización de un esquema con un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.
- `xsi:noNamespaceSchemaLocation`: localización de un esquema sin un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.

4.8.2. Componentes básicos de un esquema

Se van a introducir los componentes que van a permitir construir un esquema XML que valide documentos XML. Como se ha comentado, a los documentos XML se les llama instancias del esquema. Los componentes imprescindibles son:

- `xs:schema`
- `xs:element`
- `xs:attribute`

AVISO: Se van a describir diferentes componentes que tienen atributos. Cuando el valor de un atributo pertenezca a una lista cerrada de posibles opciones, se indicarán explícitamente. Asímismo, aparecerá en negrita en valor por defecto.

xs:schema

Es el componente de declaración de esquema y elemento raíz de todo esquema XML.

Atributos optativos principales:

- `xmlns`: una referencia URI que indica uno o más espacios de nombres a usar en el esquema. Si no se indica ningún prefijo, los componentes del esquema del espacio de nombres pueden usarse de manera no cualificada.

Otros atributos optativos:

- `id`: identificador único para el elemento.
- `targetNamespace`: una referencia URI al espacio de nombres del esquema.
- `elementFormDefault`: formato de los nombres de los elementos en el espacio de nombres del esquema. Puede ser *unqualified*, que indica que lo elementos no llevan prefijo, o *qualified*, indica que los elementos deben llevar el prefijo.

  Posibles valores: ***unqualified***, *qualified*
- `attributeFormDefault`: formato de los nombres de los atributos en el espacio de nombres del esquema. Funciona igual elementFormDefault .

  Posibles valores: ***unqualified***, *qualified*
- `version`: la versión del esquema.

Ejemplo:

Estructura de cualquier esquema:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

xs:element

Es el componente de declaración de elemento y representa la existencia de un elemento en el documento XML.

Atributos optativos principales:

- `name`: indica el nombre del elemento. Es un atributo obligatorio si el elemento padre es `<xs:schema>`.
- `ref`: indica que la descripción del elemento se encuentra en otro lugar del esquema, es decir, referencia a la descripción del elemento. Este atributo no se puede usar si el elemento padre es `<xs:schema>`.
- `type`: indica el tipo del elemento.
- `default`: es el valor que tomará el elemento al ser procesado por alguna aplicación (habitualmente un analizador de XML o un navegador) cuando en el documento instancia XML no había recibido ningún valor. Sólo se pude usar si el contenido del elemento es únicamente textual.
- `fixed`: indica el único valor que puede contener el elemento en el documento instancia XML. Sólo se pude usar si el contenido del elemento es únicamente textual.
- `minOccurs`: indica el número mínimo de ocurrencias que del elemento puede haber en el documento XML. No se puede usar este atributo si el componente padre es `<xs:schema>`. Va desde 0 hasta ilimitado (*unbounded*).

  Posibles valores: *0, 1, 2, ..., unbounded.*
- `maxOccurs`: indica el número máximo de ocurrencias que del elemento puede haber en el documento XML. No se puede usar este atributo si el componente padre es `<xs:schema>`. Va desde 0 hasta ilimitado (*unbounded*).

  Posibles valores: *0, 1, 2, ..., unbounded.*

Otros atributos optativos:

- `id`: identificador único para el elemento.
- `form`: especifica el formato del nombre del atributo. Puede ser cualificado, es decir, con el espacio de nombres como prefijo del nombre, o no cualificado, sin el espacio de nombres.

  Posibles valores: *unqualified* y *qualified.* El valor por defecto es el del atributo `elementFormDefault` del componente `<xs:schema>`.

- `substitutionGroup`: indica el nombre de otro elemento que puede ser sustituido por este elemento. Sólo se puede usar este atributo si el componente padre es `<xs:schema>`.
- `nillable`: especifica si puede aparecer el atributo de instancia `xsi:nil`, asociado al elemento en el documento instancia XML. Por defecto es falso, lo que significa que no se puede asignar a un elemento el atributo de instancia `xsi:nil`. Más adelante se comentarán los **atributos de instancia**.

  Posibles valores: ***false*** y *true*.
- `abstarct`: indica si el elemento puede ser usado en un documento XML instancia. Si vale cierto, indica que el elemento no puede aparecer en una instancia.

  Posibles valores: ***false*** y *true*.
- `final`: indica si el elemento se puede derivar de alguna manera: por extensión o por restricción o ambas. Posibles valores: *extension, restriction y #all*.

Ejemplo:

Se quiere indicar que en el documento instancia XML aparecerá un elemento de nombre `<autor>`. La manera más genérica es:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="autor"/>
</xs:schema>
```

El problema de esta descripción es que es demasiado general y no es útil, ya que no detalla si el elemento `<autor>` es de un tipo determinado, si tiene descendientes, contenido textual...

Para precisar más, indicaremos que el elemento `<autor>` es del tipo de datos predefinido `xs:string` (cadena de texto), deberá aparecer un mínimo de una vez y un máximo de tres.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="autor" type="xs:string" maxOccurs="3"/>
</xs:schema>
```

AVISO: Todos los ejemplos que se muestran deben aparecer como contenido del componente de declaración de esquema `<xs:schema>`, sin embargo, y por evitar repetición de código, se omitirá a partir de ahora. De igual forma, se omitirá la instrucción de procesamiento `<?xml ... ?>`.

Ejemplo:

Se le asigna un **valor por defecto**, *verde*, al elemento `<semaforo>`.

```
<xs:element name="semaforo" type="xs:string" default="verde"/>
```

Lo que representa esto es:

a. Si no se le da ningún valor al elemento, la aplicación que procese el documento XML le asignará de manera automática el valor por defecto.

   `<semaforo></semaforo>` ➔ `<semaforo>verde</semaforo>`

b. Si se le da un valor al elemento, que puede ser distinto al valor por defecto, se quedará con ese valor:

```
<semaforo>amarillo</semaforo>
```

Ejemplo:

Se le asigna un **valor fijo**, *rojo*, al elemento `<semaforo>` y no se le podrá asignar ningún otro valor.

```
<xs:element name="semaforo" type="xs:string" fixed="rojo"/>
```

Lo que representa esto es:

a. Si no se le da ningún valor al elemento, la aplicación que procese el documento XML le asignará de manera automática el valor fijo.

```
<semaforo></semaforo> ➔ <semaforo>rojo</semaforo>
```

b. Se le da un valor al elemento semáforo, que sólamente puede ser el valor fijo. Si se le asignara otro valor distinto, produciría un error de documento no válido:

```
<semaforo>rojo</semaforo>        ✓

<semaforo>amarillo</semaforo>    ✗
```

xs:attribute

Es el componente de declaración de atributo y representa la existencia de un atributo de un elemento en el documento XML.

Atributos optativos principales:

- `name`: nombre del atributo. Este atributo no puede aparecer simultáneamente que `ref`.
- `ref`: referencia a la descripción del atributo que se encuentra en otro lugar del esquema. Si aparece este atributo no aparecerán los atributos `type`, ni `form`, ni podrá contener un componente `<xs:simpleType>`.
- `type`: tipo del elemento.
- `use`: indica si la existencia del atributo es opcional, obligatoria o prohibida.

  Posibles valores: ***optional***, *required, prohibited.*
- `default`: valor que tomará el atributo al ser procesado por alguna aplicación (habitualmente un analizador de XML o un navegador) cuando en el documento XML no había recibido ningún valor. No puede aparecer simultáneamente con `fixed`.
- `fixed`: único valor que puede contener el atributo en el documento XML. No puede aparecer simultáneamente con `default`.

Otros atributos optativos:

- `id`: indica un identificador único para el atributo.

- `form`: especifica el formato del nombre del atributo. Puede ser cualificado, es decir, con el espacio de nombres como prefijo del nombre, o no cualificado, sin el espacio de nombres.

  Posibles valores: *unqualified* y *qualified*. El valor por defecto es el del atributo `attributeFormDefault` del componente `<xs:schema>`.

Ejemplo:

Atributo de nombre `moneda`, tipo de datos textual y con **valor por defecto** *Euro*. Este valor por defecto se le asigna al atributo si no se le ha asignado otro o no aparece.

```
<xs:attribute name="moneda" type="xs:string" default="Euro"/>
```

Ejemplo:

Atributo de nombre `unidad`, tipo de datos textual y **valor fijo** *Minutos*.

```
<xs:attribute name="unidad" type="xs:string" fixed="Minutos"/>
```

Ejemplo:

Atributo de nombre `idEmple`, tipo de datos entero positivo y **existencia** obligatoria.

```
<xs:attribute name="idEmple" type="xs:positiveInteger"
    use="required"/>
```

4.8.3. Tipos de datos

Se han visto ya en algunos ejemplos que en las declaraciones de elementos y atributos, a estos se les puede asignar un tipo de datos como `xs:string`, `xs:positiveInteger`...

En los esquemas XML, en la declaración de elementos y atributos se concreta el valor que puedan tomar mediante un tipo de datos. Los tipos de datos se organizan según diversos criterios. Una de las divisiones es en predefinidos y construidos:

- Los **predefinidos** son los que vienen integrados en la especificación de los esquemas XML.
- Los **construidos** son tipos generados por el usuario basándose en un tipo predefinido o en un tipo previamente construidos.

Tipos de datos predefinidos

Existen 44 tipos predefinidos (del inglés *built-in types*, podría traducirse como "tipos que vienen de serie"). Se organizan en forma de relación jerárquica, a la manera de una jerarquía de clases en programación orientada a objeto, de forma que cada tipo será igual que su tipo padre más alguna particularidad que lo distinga.

Existe **un tipo predefinido especial**, `xs:anyType`, que se encuentra en la raíz de la jerarquía de tipos, y del que se derivan todos los demás tipos, tanto predefinidos como complejos. Este tipo es el más genérico de todos y de cualquier elemento se podría decir que es de ese tipo. Este tipo, en sí mismo, es un tipo complejo, por lo que no se podrían declarar atributos del mismo. A un elemento del cual no se indica tipo se le asociará el tipo `xs:anyType`.

Cuanto más se desciende en la jerarquía de tipos, más restrictivos (y por tanto precisos) van siendo los tipos. La recomendación al declarar elementos y atributos es asignarles el tipo que más se ajuste a lo que se quiere definir. Por ejemplo, si queremos determinar el tipo para un elemento edad, podríamos optar por `xs:anyType`, pero es una declaración demasiado genérica. En este caso sería más adecuado usar `xs:nonNegativeInteger`, que representa a todos los números enteros más el 0.

Existe **otro tipo predefinido especial**, `xs:anySimpleType`, que representa a cualquier tipo simple sin particularizar. Se puede usar para cualquier atributo, y también para elementos que sean de tipo simple pero, de nuevo, es demasiado genérico.

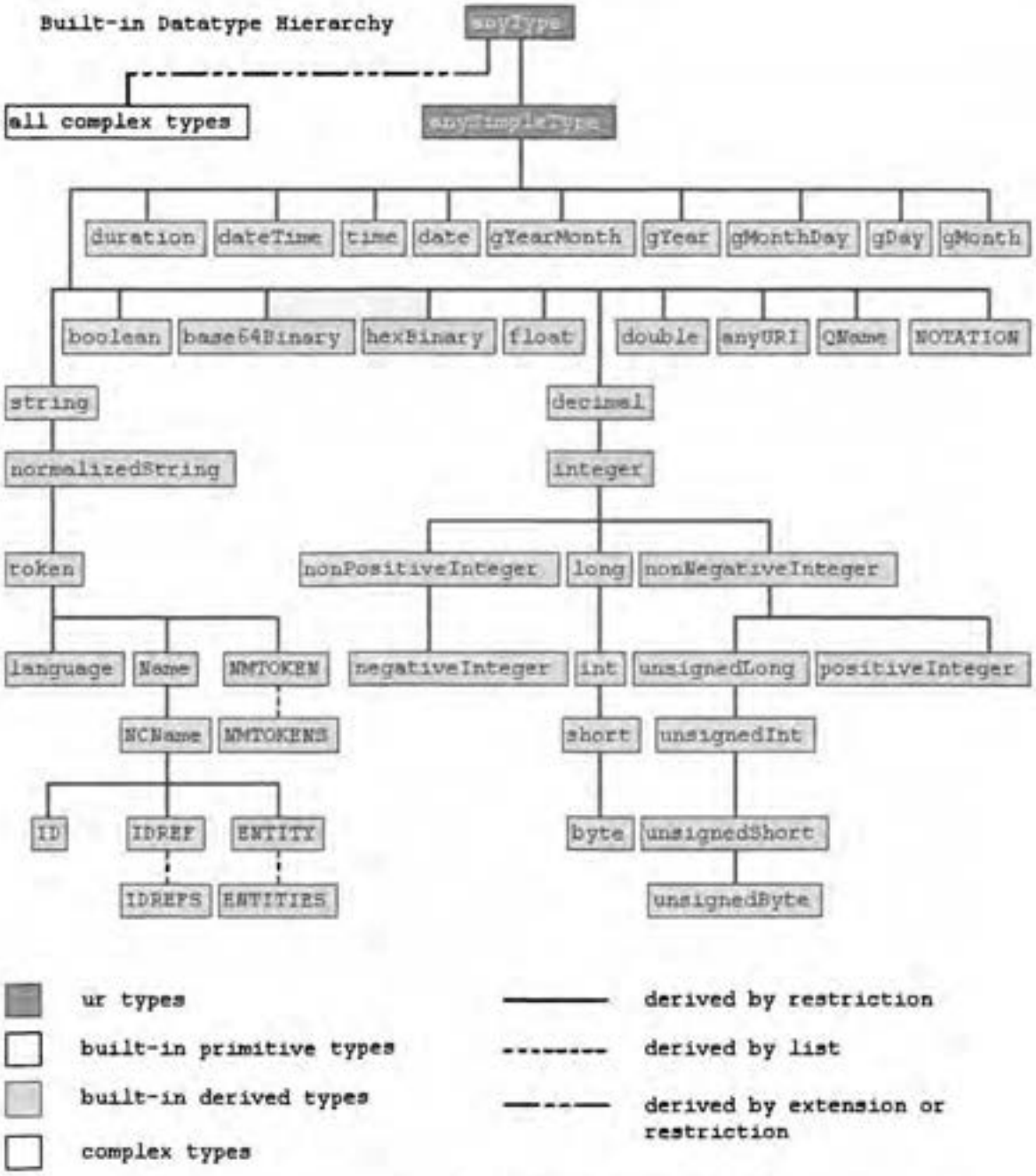


Figura 4.6: Tipos de datos predefinidos (primitivos, derivados y especiales)

Los tipos predefinidos se dividen también en **primitivos** y **no primitivos** (derivados de los primitivos).

- Los primitivos descienden directamente de `xs:anySimpleType`. Hay 19. Véase la *Figura 4.6.*
- Los no primitivos derivan de alguno de los primitivos. Hay 25. Véase la *Figura 4.6.*

Los tipos de datos predefinidos se agrupan en cinco categorías, en función de su contenido:

- Numéricos (hay 16)
- De fecha y hora (hay 9)
- De texto (hay 16)
- Binarios (hay 2)
- Booleanos (hay 1)

Numéricos

Tipo de datos	Descripción
float	Número en punto flotante de precisión simple (32 bits)
double	Número en punto flotante de precisión doble (64 bits)
decimal	Números reales que pueden ser representados como $i \cdot 10^{-n}$
integer	Números enteros, de -∞ a +∞ (-∞ … -3, -2, -1, 0, 1, 2, 3 … +∞)
nonPositiveInteger	Números enteros negativos más el 0
negativeInteger	Números enteros menores que 0
nonNegativeInteger	Números enteros positivos más el 0
positiveInteger	Números enteros mayores que 0
unsignedLong	Números enteros positivos desde 0 hasta 18.446.744.073.709.551.615 (64 bits de representación ➔ 2^{64} combinaciones)
unsignedInt	Números enteros positivos desde 0 hasta 4.294.967.295 (32 bits de representación ➔ 2^{32} combinaciones)
unsignedShort	Números enteros positivos desde 0 hasta 65.535 (16 bits de representación ➔ 2^{16} combinaciones)
unsignedByte	Números enteros positivos desde 0 hasta 255 (8 bits de representación ➔ 2^{8} combinaciones)
long	Números enteros representados con 64 bits ➔ 2^{64} combinaciones, desde -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807
Int	Números enteros representados con 32 bits ➔ 2^{32} combinaciones, desde -2.147.483.648 hasta 2.147.483.647
short	Números enteros representados con 16 bits ➔ 2^{16} combinaciones, desde -32.768 hasta 32.767

byte	Números enteros representados con 8 bits → 2^8 combinaciones, desde -128 hasta 127

Tabla 4.2: Tipos de datos numéricos

De fecha y hora

Tipo de datos	Descripción
duration	Duración en años + meses + días + horas + minutos + segundos
dateTime	Fecha y hora, en formato aaaa-mm-dd T hh:mm:ss
date	Sólamente fecha en formato aaaa-mm-dd
time	Sólamente la hora, en formato hh:mm:ss
gDay	Sólo el día, en formato –dd (la g viene de calendario Gregoriano)
gMonth	Sólo el mes (g → Gregoriano)
gYear	Sólamente el año, en formato yyyy (g → Gregoriano)
gYearMonth	Sólo el año y el mes, en formato yyyy-mm (g → Gregoriano)
gMonthDay	Sólo el mes y el año, en formato -mm-dd (g → Gregoriano)

Tabla 4.3: Tipos de datos de fecha y hora

El tipo de datos dateTime representa un valor de fecha y hora según las especificaciones de la norma ISO 8601. Es, en sí mismo, la combinación de los otros dos tipos predefinidos time y date.

El patrón de este valor es;

$$\underbrace{-?\backslash d\{4\}-\backslash d\{2\}\backslash d\{2\}}_{\text{date}}\underbrace{T\backslash d\{2\}:\backslash d\{2\}:\backslash d\{2\}\ (\backslash .\backslash d+)?}_{\text{time}}\underbrace{((\backslash [+-\backslash]\backslash d\{2\}:\backslash d\{2\})|Z)?}_{\text{común a date y time}}$$

Patrón	Significado
-?	Un signo negativo (opcional). Para representar fechas anteriores al año 0
\d{4}-\d{2}\d{2}	La parte de la fecha (obligatoria). Equivale a yyyy-mm-dd
\d{2}:\d{2}:\d{2}	La parte de la hora (obligatoria). Equivale a hh:mm:ss
T	Indicador de hora local
(\.\d+)?	La parte decimal de la fracción de segundo (opcional)
(([+-]\d{2}:\d{2})|Z)?	La parte de la zona horaria (opcional) La Z permite expresar las horas en formato UTC (Universal Time Coordinated – Tiempo Universal Coordinado), anteriormente GMT (Greenwich Meridian Time – Hora del Meridiano de Greenwich)

Tabla 4.4: Estructura del tipo de datos `xs:dateTime`

Ejemplo:

Para representar una hora que sea las 3:40 PM del horario estándar de la costa este de Estados Unidos, que va 5 horas por detrás del UTC, se escribiría 15:40:00-05:00. Esa misma hora, pero de la zona central de Australia, que va 9 horas y media por delante del UTC, se escribiría como 15:40:00+09:30.

Ejemplo:

Aunque se verá más adelante, se introduce el concepto de **faceta**, que es una restricción que permite generar nuevos tipos de datos a partir de uno existente, limitando su rango de valores posibles.

Un tipo derivado de `xs:date` que representa la fecha en la que se ha entregado una determinada práctica, siendo el valor de la faceta `xs:maxInclusive` la fecha límite.

```
<!-- Se usa maxInclusive para indicar una límite final de fecha -->
<xs:simpleType name="TipoFechaEntregaPractica">
  <xs:restriction base="xs:date">
    <xs:maxInclusive value="2009-12-31"/>
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Un tipo derivado de `xs:time` que representa la hora en la que un pasajero ha embarcado en un determinado vuelo, siendo los valores de las facetas `xs:minInclusive` y `xs:maxInclusive` la hora de comienzo y fin del embarque.

```
<!-- Se usa minInclusive y maxInclusive para indicar límites
    iniciales y finales de horas -->
<xs:simpleType name="TipoHorarioEmbarque">
  <xs:restriction base="xs:time">
    <xs:minInclusive value="05:00:00"/>
    <xs:maxInclusive value="05:30:00"/>
  </xs:restriction>
</xs:simpleType>
```

De texto

Tipo de datos	Descripción
string	Cadenas de texto
normalizadString	Cadenas de texto en las que se convierten los caracteres tabulador, nueva línea, y retorno de carro en espacios simples
token	Cadenas de texto sin los caracteres tabulador, ni nueva línea, ni retorno de carro, sin espacios por delante o por detrás, y con espacios simples en su interior
language	Valores válidos para xml:lang (según la especificación de XML)

NMTOKEN	Tipo de datos **para atributo** según XML 1.0, compatible con DTD
NMTOKENS	**Lista** separada por espacios de NMTOKEN, compatible con DTD
Name	Tipo de nombre según XML 1.0
QName	Nombre cualificado de espacio de nombres XML
NCName	QName sin el prefijo ni los dos puntos
anyURI	Cualquier URI
ID	Tipo de datos **para atributo** según XML 1.0, compatible con DTD. Han de ser valores únicos en el documento XML
IDREF	Tipo de datos **para atributo** según XML 1.0, compatible con DTD
IDREFS	**Lista** separada por espacios de IDREF, compatible con DTD
ENTITY	Tipo de datos **para atributo** en XML 1.0, compatible con DTD
ENTITIES	**Lista** separada por espacios de ENTITY, compatible con DTD
NOTATION	Tipo de datos **para atributo** en XML 1.0, compatible con DTD

Tabla 4.5: Tipos de datos textuales

Ejemplos:

Language	en-US
Name	horaSalida
QName	cliente:nombre
NCName	Nombre
anyURI	http://www.w3c.com
NMTOKENS	SP FR PR IT

Binarios

Tipo de datos	**Descripción**
hexBinary	Secuencia de dígitos hexadecimales (0..9,A,B,C,D,E,F)
base64Binary	Secuencia de dígitos en base 64

Tabla 4.6: Tipos de datos binarios

Booleano

Tipo de datos	**Descripción**
boolean	Puede tener 4 valores: 0, 1, true y false

Tabla 4.7: Tipos de datos booleanos

Todos los tipos de datos predefinidos:

- Pueden ser asignados tanto a elementos como a tributos.
- Pertenecen al espacio de nombres `http://www.w3.org/2001/XMLSchema`.

Ejemplo:

Se declararán tres elementos cuyos tipos de datos, supuestamente predefinidos, están incorrectamente declarados. Finalmente, se declara un elemento con un tipo de datos válido:

```
<xs:element name="elementoA" type="date"/>  ❶
<xs:element name="elementoB" type="w3c:date"/>  ❷
<xs:element name="elementoC" type="xsd:date"/>  ❸
<xs:element name="elementoD" type="xs:date"/>  ❹
```

❶ El tipo de datos `date` no se puede resolver como ninguno de los componentes de definición de tipos del esquema.

❷ El tipo de datos `w3c:date` tiene un prefijo, `w3c`, que no ha sido declarado.

❸ El tipo de datos `xsd:date` se encuentra en el espacio de nombres `http://www.w3.org/2001/XMLSchema-datatypes`, pero los componentes existentes en este espacio de nombres no son referenciables desde el esquema. Se ha indicado un espacio de nombres erróneo.

❹ El tipo de datos `xs:date` se encuentra en el espacio de nombres `http://www.w3.org/2001/XMLSchema`, que es el adecuado para los tipos de datos predefinidos.

4.8.4. Tipos de datos simples vs. complejos

Aparece aquí una nueva catalogación de tipos de datos, en simples y complejos.

- **Tipos de datos simples**:

- Todos los tipos de datos predefinidos son simples.
- En general representan valores **atómicos** (el número 7, el texto "Hola"), no listas. Excepcionalmente, se construyen **no atómicos**, como una lista de números primos (1, 2. 3, 5, 7, 11...). Todos los tipos predefinidos son atómicos.
- Se pueden asignar tanto a elementos que sólo tengan contenido textual como a atributos.
- También se pueden construir, derivando de un tipo base predefinido al que se le introducen restricciones.

Ejemplo:

Un tipo de datos simple derivado es el que represente un correo electrónico. Se basa en el tipo de datos predefinido `xs:string`, que representa cualquier cadena de texto, pero fuerza a que cumpla una serie de reglas (contener el carácter "@", el carácter ".".).

- **Tipos de datos complejos**:
  - Sólo se pueden asignar a elementos que tengan elementos descendientes y/o atributos. Estos elementos pueden tener contenido textual.
  - Por defecto, un elemento con un tipo de datos complejo contiene contenido complejo, lo que significa que tiene elementos descendientes.
  - Un tipo de datos complejo se puede limitar a tener contenido simple, lo que significa que sólo contiene texto y atributos. Se diferencia de los tipos de datos simples precisamente en que tiene atributos.
  - Se pueden limitar para que no tengan contenido, es decir, que sean vacíos, aunque pueden tener atributos.
  - Pueden tener contenido mixto: combinación de contenido textual con elementos descendientes.

Definición de tipos simples

Se usará el componente de definición de tipos simples, `<xs:simpleType>`. Se podrán limitar el rango de valores con el componente `<xs:restriction>`.

Se pueden asignar a elementos y a atributos.

xs:simpleType

Es el componente de definición de tipos simples.

Atributos optativos principales:

- `name`: nombre del tipo. Este atributo es obligatorio si el componente es hijo de `<xs:schema>`, de lo contrario no está permitido.
- `id`: identificador único para el componente.

Hasta ahora sólo se ha visto cómo asignar un tipo de datos predefinido (que siempre es simple) a un elemento.

Ejemplo:

Asignar el tipo de datos predefinido `xs:float` al elemento `<ingresosAnuales>`.

```
<xs:element name="ingresosAnuales" type="xs:float"/>
```

La declaración anterior de elemento con un tipo simple asignado es una forma simplificada de la que se muestra a continuación. Se recomienda usar la simplificada.

```
<xs:element name="ingresosAnuales">
  <xs:simpleType>
    <xs:restriction base="xs:float" />
  </xs:simpleType>
</xs:element>
```

Ahora se puede **construir un tipo de datos simple**, al que se le asignará un nombre, y que será un sinónimo del tipo de datos predefinido `xs:float`. Se usará el componente de restricción de valores de un tipo simple, `xs:restriction`. Este tipo no aporta nada nuevo `xs:float` y se muestra a modo de ejemplo. No se recomienda definir tipos de datos que sean sinónimos literales de predefinidos.

```
<xs:simpleType name="TipoIngresosAnuales">  ❶
    <xs:restriction base="xs:float" />
  </xs:simpleType>

<xs:element name="ingresosAnuales" type="TipoIngresosAnuales"/>  ❷
```

❶ Se define el tipo simple de nombre `TipoIngresosAnuales`. A partir de ahora se podrá referenciar en todo el esquema y es absolutamente equivalente al predefinido `xs:float`.

❷ Se le asigna ese tipo al elemento `<ingresosAnuales>`.

xs:restriction

Es un componente que define restricciones en los valores de un tipo.

Atributos obligatorios:

- `base`: nombre del tipo base a partir del cual se construirá el nuevo tipo, sea predefinido o construido.

Atributos optativos principales:

- `id`: identificador único para el componente.

Ejemplo:

Se declara un tipo simple que se utilizará para elementos que sean una contraseña. Será de tipo base `xs:string`, con unas **facetas** (se comentan a continuación) que fuercen su tamaño mínimo a 6 y máximo a 12.

```
<xs:simpleType name="TipoContrasenia">
  <xs:restriction base="xs:string">  ❶
    <xs:minLength value="6" />  ❷
    <xs:maxLength value="12" />  ❸
  </xs:restriction>
</xs:simpleType>
```

❶ Se está construyendo un nuevo tipo simple a base de restringir el rango de valores permitidos del tipo base.

❷ Se fuerza a que el nuevo tipo sea de cadenas de longitud mínima de 6. En una faceta.

❸ Se fuerza a que el nuevo tipo sea de cadenas de longitud máxima de 6. Es una faceta.

La declaración de un elemento de nombre `<clave>` como del tipo recién creado sería:

```
<xs:element name="clave" type="TipoContrasenia">
```

Un par de ejemplos de uso, uno válido y uno inválido, en un documento instancia XML son:

```
<clave>claveValida!</clave>        ✓
<clave>mala</clave>                ✗
```

Facetas para restringir rangos de valores

Se usan para limitar el rango de valores que tiene un tipo base, lo que produce la aparición de un tipo limitado o facetado.

En función del tipo base, existen unas posibles facetas que se pueden aplicar.

Faceta	Uso	Tipos de datos para donde se usa
`xs:minInclusive`	Especifica el límite inferior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
`xs:maxInclusive`	Especifica el límite superior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
`xs:minExclusive`	Especifica el límite inferior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
`xs:maxExclusive`	Especifica el límite superior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
`xs:enumeration`	Especifica una lista de valores aceptables.	Todos
`xs:pattern`	Especifica un patrón o expresión regular que deben cumplir los valores válidos.	Texto
`xs:whiteSpace`	Especifica cómo se tratan los espacios en blanco, entendiéndose como tales los saltos de línea, tabuladores y los propios espacios. Sus posibles valores son *preserve*, *replace* y *collapse*.	Texto
`xs:length`	Especifica el número exacto de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto

xs:minLength	Especifica el mínimo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
xs:maxLength	Especifica el máximo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
xs:fractionDigits	Especifica el número máximo de posiciones decimales permitidas en números reales. Ha de ser mayor o igual que 0.	Números con parte decimal
xs:totalDigits	Especifica el número exacto de dígitos permitidos en números. Ha de ser mayor que 0.	Numéricos

Tabla 4.8: Facetas

Ejemplo:

El elemento `<edadLaboral>` es un entero no negativo, que debe tener un valor mínimo de 16, incluido, y máximo 70, no incluido. En notación matemática se emplean los corchetes para identificar los intervalos cerrados (los extremos están incluidos) y paréntesis para identificar los intervalos abiertos (los extremos no están incluidos). En nuestro caso sería [16, 70)

```
<xs:element name="edadLaboral">
  <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:minInclusive value="16"/>
      <xs:maxExclusive value="70"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Si quisiéramos **definir un tipo de datos al que poder referenciar repetidas veces** y declarar un elemento `<edadLaboral>` de ese tipo, sería:

```
<xs:simpleType name="TipoEdadLaboral">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="16"/>
    <xs:maxExclusive value="70"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="edadLaboral" type="TipoEdadLaboral">
```

NOTA: Se recomienda llevar a cabo esta práctica, es decir, definir tipos de datos con nombre que puedan ser referenciados posteriormente. Se podría crear una librería de tipos de datos.

Ejemplo:

Se quiere definir un tipo de dato simple, `TipoEstaciones`, basado en el tipo predefinido `xs:token`, y que sólamente permita como valores los nombres de las cuatro estaciones.

```
<xs:simpleType name="TipoEstaciones">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Primavera" />
    <xs:enumeration value="Verano" />
    <xs:enumeration value="Otoño" />
    <xs:enumeration value="Invierno" />
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Se quiere un tipo de datos, `TipoCantidad`, basado en `xs:decimal`, que permita números con 2 dígitos decimales y 11 dígitos totales. El rango de valores implícito sería desde -999.999.999,99 a 999.999.999,99.

```
<xs:simpleType name="TipoCantidad">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="11"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Se define un tipo basado en `xs:string`, usado para registrar direcciones en el que se quiere que todos los caracteres de espaciado (espacio, tabulador, nueva línea y retorno de carro) se colapsen, lo que significa que si estos caracteres aparecen al principio o final de la cadena se eliminan, y si lo hacen en su interior, formando una hilera, se sustituyen en un solo espacio.

La definición de este tipo coincide con el tipo predefinido `xs:token`, derivado de `xs:string`.

```
<xs:simpleType name="TipoDireccionFormateada">
  <xs:restriction base="xs:string">
    <xs:whiteSpalce value="replace"/>
  </xs:restriction>
</xs:simpleType>
```

Actividad 4.6:

Define diferentes tipos simples a partir de las siguientes especificaciones. A continuación, crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos. Por último, escribe un documento instancia XML que sea válido con respecto a ese esquema. Repítelo para los distintos tipos definidos:

a. Un número real con tres decimales que represente las temperaturas posibles en la Tierra, suponiendo que van desde -75° a 75°, ambas inclusive.

b. Un `xs:token` que sólo pueda valer las siglas de los países vecinos de España, incluyendo a la propia España: ES, PR, FR, AN.

c. Un número real que represente salarios, con 5 dígitos enteros y 2 decimales.

d. Un mensaje de la red social Tweeter, conocido como tweet (trino), es una cadena de texto de una longitud máxima de 140 caracteres.

Una faceta que permite usos muy precisos es `xs:pattern`, que se analiza en detalle.

xs:pattern

Representa un patrón o expresión regular que debe de cumplir la cadena de texto a la que se aplique.

Ejemplo:

Se quiere definir un tipo de datos, basado en xs:string, que se caracterice por cadenas de tres letras mayúsculas.

```
<xs:simpleType name="TipoTresLetrasMayusculas">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][A-Z][A-Z]"/>
  </xs:restriction>
</xs:simpleType>
```

La sintaxis usada para estos patrones se basa en expresiones regulares. Las más usadas son:

Patrón	Significado	Ejemplo
.	Cualquier carácter	;
\w	Cualquier letra	M
\d	Un dígito	7
\D	Cualquier carácter no dígito	j
\s	Cualquier carácter de espaciado (tabulador, espacio…)	
\S	Cualquier carácter de no espaciado	C
\d{n}	n dígitos exactamente (Ej. cuatro dígitos exactamente)	3856

\d{n,m}	De n m dígitos (Ej. de dos o tres dígitos)	91
\d{n,}	n o más dígitos (Ej. Tres o más dígitos)	3500
[xyz]	Uno de los caracteres x, y o z (en minúscula)	Y
[A-Z]	Uno de los caracteres de la A a la Z (en mayúscula)	
[^abc]	Negación de un grupo de caracteres	D
[F-J-[H]]	Sustracción de un carácter de un rango	G
(a|b)	Alternativa entre dos expresiones	b
b?	Sucesión de 0 o una ocurrencias de una cadena	B
1*	Sucesión de 0 o más ocurrencias de una cadena	111
(cd)+	Sucesión de 1 o más ocurrencias de una cadena	cdcd

Tabla 4.9: Expresiones regulares

Como se ha visto en los patrones anteriores, en las facetas se utilizan cuantificadores (como también se vio en los DTD):

Cuantificador	Significado
?	0 ó 1 ocurrencias
*	0 o más ocurrencias
+	1 o más ocurrencias
{n}	n ocurrencias exactas
{n,m}	De n a m ocurrencias (siendo n < m)
{n,}	n o más ocurrencias

Tabla 4.10: Expresiones regulares para cuantificar

Hay ciertos caracteres que tienen un significado propio en las expresiones regulares: {, }, [,], (,), ?, *, +, -, |,^,.,\

Para indicar en una expresión regular la aparición literal de uno de estos caracteres, hay que anteponerle el carácter de escape \.

Ejemplo:

La expresión regular para una cadena que contenga las letras a o b o el carácter + sería:

```
[ab\+]
```

Ejemplo:

Un número de teléfono que se quiere se represente como 3 dígitos, un punto, 3 dígitos, un punto, tres dígitos y un punto. Por ejemplo 912.345.678:

```
<xs:simpleType name="TipoTelefonoPunteado">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}\.\d{3}\.\d{3}"/>
  </xs:restriction>
</xs:simpleType>
```

Actividad 4.7:

Define diferentes tipos simples a partir de las siguientes especificaciones.

a. Una cadena de texto que represente las matrículas españolas anteriores a la normalización del año 2000. El formato era: una o dos letras mayúsculas + un guion + cuatro dígitos + un guion + una o dos letras mayúsculas. (ej. Z-1234-AB)

b. Una cadena de texto que represente las cuatro posibles formas de pago: con tarjetà VISA, MasterCard, American Express y en Efectivo.

A continuación, crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos.

Por último, escribe un documento instancia XML que sea válido con respecto a ese esquema. Repítelo para los distintos tipos definidos.

Más sobre derivación de tipos simples: uniones y listas

Otra forma de derivar un tipo de datos consiste en definir uniones o listas de tipos de datos base:

- **Unión**: es un tipo de datos creado a partir de una colección de tipos de datos base. Un valor es válido para una unión de tipos de datos si es válido para al menos uno de los tipos de datos que forman la unión. Se construye con el componente `<xs:union>`.

  Ejemplo:

  Se quiere reflejar que las tallas de ropa se pueden identificar por números (38, 40, 42) o por letras, que son las iniciales en inglés (S, M, L).

```
<xs:element name="talla">
  <xs:simpleType>
    <xs:union memberTypes="TipoTallaNumerica TipoTallaTextual" />
  </xs:simpleType>
</xs:element>

<xs:simpleType name="TipoTallaNumerica">
  <xs:restriction base="xs:positiveInteger">
    <xs:enumeration value="38"/>
    <xs:enumeration value="40"/>
    <xs:enumeration value="42"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="TipoTallaTextual">
  <xs:restriction base="xs:string">
    <xs:enumeration value="S"/>
    <xs:enumeration value="M"/>
    <xs:enumeration value="L"/>
  </xs:restriction>
</xs:simpleType>
```

Cualquiera de estos dos fragmentos XML serían válidos:

```
<talla>42</talla>
```

Y también:

```
<talla>M</talla>
```

- **Lista**: es un tipo de datos compuesto por listas de valores de un tipo de datos base. La lista de valores debe aparecer separada por espacios. Se construye con el componente `<xs:list>`.

  Ejemplo:

  Se define un tipo `TipoListaNumerosBingo` como una lista de valores enteros positivos que empieza en el 1 y acaba en el 90.

```
<xs:simpleType name="TipoListaNumerosBingo">
  <xs:list>
    <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="90"/>
    </xs:restriction>
  </xs:list>
</xs:simpleType>

<xs:element name="numerosGanan" type="TipoListaNumerosBingo"/>
```

Un fragmento XML que sería válido para la definición precedente será:

```
<numerosGanadores>1 7 19 34 42 60 63 73</numerosGanadores>
```

Actividad 4.8:

Dado un tipo simple, `TipoColoresSemaforo`, basado en `xs:string` y que sólo pueda tomar los valores *Rojo*, *Amarillo* y *Verde*, define un nuevo tipo `ListaColoresSemaforo` que sea una lista cuyos elementos sean del tipo `TipoColoresSemaforo`. Por último, crea otro tipo que sea una lista de sólo tres colores de semáforo, `Tipo3ColoresSemaforo`.

Crea un esquema que contenga estos tipos y declarar un elemento del tipo recién construido `Tipo3ColoresSemaforo`.

Crea un documento XML instancia del esquema, que contenga un único elemento de ese tipo. Comprueba que es válido respecto al esquema.

4.8.5. Definición de tipos de datos complejos

Se usará el componente de definición de tipos complejos, `<xs:complexType>`. Sólo se pueden asignar a elementos. Un elemento se considera de tipo complejo si tiene atributos y/o descendientes.

El **contenido** de un elemento es aquello que va entre sus etiquetas de apertura y de cierre. Un elemento puede tener el siguiente **contenido**:

- **Contenido simple** (`xs:simpleContent`): el elemento declarado sólo tiene contenido textual, sin elementos descendientes.
- **Contenido complejo** (`xs:complexContent`): el elemento declarado tiene elementos descendientes. Puede tener o no contenido textual.

Modelos de contenido

En el momento en el que se habla de elementos descendientes, aparece otro elemento en los esquemas que son los **compositores** o **modelos de contenido**.

Indican la distribución que tienen entre sí los elementos descendientes de uno dado. Siempre aparecen conformando un tipo de datos complejo. Hay tres posibilidades distintas, cada una de ellas declarada con un componente:

- **Secuencia:**

  Los elementos aparecen en fila, unos detrás de otros, en un orden determinado. Se declara con el componente `<xs:sequence>`. Al igual que sucedía con los elementos, la secuencia puede aparecer un número variable de veces, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

  Ejemplo:

  El `<mensaje>` con elementos descendientes `<emisor>`, `<receptor>` y `<contenido>`.

```
<xs:sequence>
  <xs:element name="emisor" type="xs:string"/>
  <xs:element name="receptor" type="xs:string"/>
  <xs:element name="contenido" type="xs:string"/>
</xs:sequence>
```

- **Alternativa:**

  Los elementos aparecen como alternativa unos de los otros. Sólo se elije uno. Se declara con el componente `<xs:choice>`. La alternativa puede aparecer un número variable de veces, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

  Ejemplo:

  Al comienzo de una novela podrá haber un elemento `<prologo>`, o un elemento `<prefacio>` o un elemento `<introduccion>`, sólo uno de los tres, todos textuales.

```
<xs:choice>
  <xs:element name="prologo" type="xs:string"/>
  <xs:element name="prefacio" type="xs:string"/>
  <xs:element name="introduccion" type="xs:string"/>
</xs:choice>
```

- **Todos:**

  Los elementos aparecen en cualquier orden. Se declara con el componente `<xs:all>`. El uso de este modelo de contenido no se recomienda por la pérdida de control sobre el orden de los elementos. La aparición de todos los elementos puede aparecer 0 o 1 vez, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

  Ejemplo:

  Supóngase que existen dos elementos `<alfa>` y `<omega>`, que tienen que aparecer como descendientes de `<griego>`, pero el orden es indiferente. Sería:

```
<xs:all>
  <xs:element name="alfa" type="xs:string"/>
  <xs:element name="omega" type="xs:string"/>
</xs:all>
```

xs:complexType

Es el componente de definición de tipos complejos.

Atributos optativos principales:

- `name`: nombre del tipo. Este atributo es obligatorio si el componente es hijo de `<xs:schema>`, de lo contrario no está permitido.
- `mixed`: indica si se intercala contenido textual con los elementos descendientes:

  Posibles valores: ***false***, *true*.
- `id`: identificador único para el componente.
- `abstract`: indica si se puede usar directamente como tipo de un elemento de un documento instancia XML. Si vale cierto, significa que no puede usarse directamente y el tipo debe derivarse para generar otro tipo que sí se pueda usar.

  Posibles valores: *false*, *true*.
- `final`: indica si el tipo puede ser derivable por extensión por restricción o por ambas.

  Posibles valores: *extension*, *restriction*, *#all*.

Ejemplo:

Un tipo complejo para un elemento `<persona>` que contenga una secuencia de elementos descendientes `<primerApellido>`, `<segundoApellido>` y `<fechaNacimiento>`, de los tipos esperables.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="primerApellido" type="xs:string" />
      <xs:element name="primerApellido" type="xs:string" />
      <xs:element name="fechaNacimiento" type="xs:date" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Como ya se ha comentado previamente, se podría definir un `TipoPersona` y declarar un elemento de ese tipo:

```
<xs:complexType name="TipoPersona">
  <xs:sequence>
    <xs:element name="primerApellido" type="xs:string" />
    <xs:element name="primerApellido" type="xs:string" />
    <xs:element name="fechaNacimiento" type="xs:date" />
  </xs:sequence>
</xs:complexType>

<xs:element name="persona" type="TipoPersona">
```

4.8.6. Diferentes declaraciones de elementos

Para tipos simples y complejos, combinando las posibilidades de contenido (simple y complejo) y los elementos que condicionan el tipo y el contenido (atributos, elementos descendientes, contenido textual) tendríamos la siguiente tabla:

Tipo	Contenido	Elementos descendientes	Atributos	Contenido textual
Simple	Simple			✓
Complejo	Simple		✓	✓
Complejo	Complejo			
Complejo	Complejo		✓	
Complejo	Complejo	✓		
Complejo	Complejo	✓	✓	
Complejo	Complejo mixto	✓		✓
Complejo	Complejo mixto	✓	✓	✓

Tabla 4.11: Tipos de elementos

Declaración de elementos vacíos

Un elementos es vacío (EMPTY en los DTD) si no tiene contenido textual, ni elementos descendientes (aunque podría tener atributos).

Hay diversas maneras de representarlo:

- Como un tipo simple derivado de `xs:string` con longitud 0.
- Como un tipo complejo sin contenido, que no contiene elementos descendientes. Se aconseja esta.

Ejemplo:

Para el elemento `<br>` de HTML, se declara un tipo complejo sin contenido.

```
<xs:element name="br">
  <xs:complexType />
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xs:element name="br">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Declaración de elementos con contenido textual y atributos

Se declarara con un tipo de datos complejo con contenido simple que contenga algún atributo.

Lo que se hace realmente es tomar un tipo de datos simple y, mediante un componente de declaración de extensión, `<xs:extension>`, se le añade uno o más atributos, lo que hace que el tipo se convierta en complejo.

Ejemplo:

El elemento `<textarea>` de HTML, que acepta contenido textual y tiene atributos (`name`, `cols`, `rows`...). Un fragmento de un documento HTML sería:

```
<textarea name="comentarios" cols="10" rows="5">
  Introduzca aquí sus comentarios
</textarea>
```

La definición del tipo complejo para un elemento `<textarea>` sería:

```
<xs:element name="textarea">
  <xs:complexType>
    <xs:simpleContent>  ❶
      <xs:extension base="xs:string">  ❷
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="cols" type="xs:positiveInteger"/>
```

```
          <xs:attribute name="rows" type="xs:positiveInteger"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
```

❶ El componente de definición de contenido simple especifica que el elemento complejo no debe tener elementos descendientes.

❷ El componente de definición de extensión, `<xs:extension>`, extiende el tipo de datos indicado como base para que el contenido incluya declaración de atributos.

Actividad 4.9:

Define un tipo de datos que valide el siguiente fragmento de un documento instancia XML:

```
<temperatura escala="celsius">37</temperatura>
```

El tipo de datos del elemento `<temperatura>` es `xs:integer`, y el del atributo `escala`, uno derivado de `xs:string` que sólamente permita los valores *Celsius*, *Kelvin* o *Farenheit*.

Declaración de elementos sólo con atributos

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún atributo.

Ejemplo:

El elemento `<img>` de HTML, que tiene atributos (`src`, `width`, `height`...) pero no tiene elementos descendientes ni contenido textual.

```
<img src="amanecer.jpg" alt="Amanecer, sol naciente" />
```

La declaración sería:

```
<xs:element name="img">
  <xs:complexType>
    <xs:attribute name="src" type="xs:string"/>
    <xs:attribute name="alt" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xs:element name="img">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="src" type="xs:string"/>
```

```
        <xs:attribute name="alt" type="xs:string"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Declaración de elementos sólo con elementos descendientes

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún elemento.

Ejemplo:

El elemento `<ul>` de HTML, que tiene elementos descendientes `<li>` pero no tiene atributos ni contenido textual. Un fragmento en un documento HTML sería:

```
<ul>
  <li>Primer elemento</li>
  <li>Segundo elemento</li>
</ul>
```

La declaración del elemento es:

```
<xs:element name="ul">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="li" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xs:element name="ul">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="li" type="xs:string"
                      maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Declaración de elementos con atributos y elementos descendientes

Se declara con un tipo de datos complejo con contenido complejo que contenga algún elemento descendiente y algún atributo.

Ejemplo:

El elemento `<table>` de HTML, que tiene atributos, como `border`, y elementos descendientes, como `<tr>`, pero no tiene contenido textual. Un fragmento de un documento HTML podría ser:

```
<table border="1">
  <tr>
    <td>Primera celda</td>
    ...
  </tr>
  ...
</table>
```

La declaración será:

```
<xs:element name="table">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tr" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="border" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="tr">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="td" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja. En este ejemplo, no se llega a declarar el elemento `<td>`.

```
<xs:element name="table">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="tr" type="xs:anyType"
                      maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="border" type="xs:string" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Declaración de elementos con contenido textual y elementos descendientes

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún elemento descendiente y contenido textual.

Ejemplo:

El elemento `<p>` de HTML, suponiendo que no tuviese atributos, puede contener texto directamente, o texto en negrita `<b></b>`, cursiva `<i></i>`, etc.

```
<p>
  El perro de <b>San Roque</b> no tiene rabo porque <i>Ramón
Rodríguez</i> se lo ha robado.
</p>
```

Para indicar que el contenido es mixto, textual y elementos descendientes, se fija atributo `mixed="true"`. La declaración será:

```
<xs:element name="p">
  <xs:complexType mixed="true">  ❶
    <xs:choice minOccurs="0" maxOccurs="unbounded">  ❷
      <xs:element name="b" type="xs:string" />
      <xs:element name="i" type="xs:string" />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

❶ Aparece el atributo `mixed="true"` para indicar un contenido mixto.

❷ Aparece el modelo de contenido `<xs:choice>`, fijando el valor de sus atributos `minOccurs="0"` y `maxOccurs="unbounded"`, para permitir un número indefinido de ocurrencias del elemento `<b>` o del elemento `<i>`, en el orden que sea.

Ejemplo:

El mismo que el anterior, pero asegurando que los elementos `<b>` e `<i>` aparecen sólo una vez y en ese orden. Se usará el modelo de contenido `<xs:sequence>`.

```
<xs:element name="p">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="b" type="xs:string" />
      <xs:element name="i" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Declaración de elementos con atributos, elementos descendientes y contenido textual

Se declara con un tipo de datos complejo con contenido complejo que contenga elementos, atributos y contenido textual.

Ejemplo:

El elemento `<form>` de HTML, que tiene atributos (`name`, `method...`), elementos descendientes como `<input>` y contenido textual.

```
<form name="f1" method="post">
  Apellido: <input type="text" name="apellido" />
  ...
</form>
```

Para indicar que el contenido es mixto, textual y elementos descendientes, se fija el atributo `mixed="true"`. La declaración será:

```
<xs:element name="form">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="input"/>
    </xs:choice>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="method" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="input">
  <xs:complexType>
    <xs:attribute name="type" type="xs:string" use="required"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xs:element name="form">
  <xs:complexType>
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="input" type="xs:anyType"
                      maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="method" type="xs:string"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Extensión de un tipo complejo

La extensión de tipos complejos se asemeja a la herencia en la programación orientada a objeto: se pueden añadir nuevos elementos y atributos a tipos complejos ya existentes. Los elementos añadidos aparecerán al final de todos los elementos del tipo base.

Ejemplo:

Se dispone de un tipo complejo `TipoAlumno` con la siguiente definición:

```
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="Apellido" type="xs:string" />
    <xs:element name="Ciclo" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Se le quiere añadir un elemento numérico `<Curso>` y un atributo que sea el número de `Matrícula`.

Se tratará un tipo complejo que llamaremos `TipoAlumnoExtendido`:

```
<xs:complexType name="TipoAlumnoExtendido">
  ...
</xs:complexType>
```

El contenido del elemento que tenga ese tipo es de tipo complejo:

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
  ...
  </xs:complexContent>
</xs:complexType>
```

A continuación se describe la extensión, usando el componente de extensión, cuyo atributo `base` permite indicar el tipo de datos que sirve de base, en este caso `TipoAlumno`:

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    <xs:extension base="TipoAlumno">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Por último, se define el elemento `<Curso>` y el atributo `Matrícula` que queremos añadir. El atributo o atributos a añadir deberán ubicarse después de la definición de secuencia, alternativa u otros elementos hijo.

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    <xs:extension base="TipoAlumno">
      <xs:sequence>
        <xs:element ref="Curso" />
      </xs:sequence>
      <xs:attribute name="Matrícula" type="xs:positiveInteger"
```

```
                        use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Curso" type="xs:positiveInteger" />
```

La definición completa del tipo base y el tipo extendido es:

```
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="Apellido" type="xs:string" />
    <xs:element name="Ciclo" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    <xs:extension base="TipoAlumno">
      <xs:sequence>
        <xs:element name="Curso" type="xs:positiveInteger" />
      </xs:sequence>
      <xs:attribute name="Matrícula" type="xs:positiveInteger"
                    use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

La declaración de un elemento `<Alumno>` del tipo `TipoAlumnoExtendido` será:

```
<xs:element name="Alumno" type="TipoAlumnoExtendido" />
```

Se concluye este ejemplo mostrando la descripción de elementos que serían válidos de acuerdo al tipo recién definido y otros que serían inválidos.

Ejemplo:

El elemento es válido con respecto a su declaración. Tiene los elementos descendientes definidos, Apellido, Ciclo y Curso, en el orden adecuado. Igualmente tiene el atributo Matrícula.

```
<Alumno Matrícula="123">                    ✓
  <Apellido>Marín</Apellido>
  <Ciclo>ASIR</Ciclo>
  <Curso>1</Curso>
</Alumno>
```

El elemento no es válido con respecto a su declaración. Tiene los elementos descendientes definidos, Apellido, Ciclo y Curso, pero en un orden distinto al de la declaración.

```
<Alumno Matrícula="123">                    ✗
  <Apellido>Marín</Apellido>
  <Curso>1</Curso>
  <Ciclo>ASIR</Ciclo>
</Alumno>
```

El elemento no es válido con respecto a su declaración. No existe el atributo Matrícula, cuya aparición es requerida (obligatoria).

```
<Alumno>                                    ✗
  <Apellido>Marín</Apellido>
  <Ciclo>ASIR</Ciclo>
  <Curso>1</Curso>
</Alumno>
```

4.8.7. Modelos de diseño de esquemas XML

Existen varios modelos de estructuración de las declaraciones al construir esquemas, si bien conviene recordar que el orden en que aparecen los componentes en un esquema no es representativo.

- Diseño anidado o de muñecas rusas: se llama así porque se anidan declaraciones unas dentro de otras, como las muñecas Matrioskas. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Esto produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones.

  Con esta técnica los esquemas son más cortos pero su lectura es más compleja para el ojo humano.

  Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"/>
        <xs:element name="personaje" minOccurs="0"
                                     maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="amigoDe" type="xs:string"
                          minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="desde" type="xs:date"/>
              <xs:element name="calificacion" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- Diseño plano: se declaran los elementos y los atributos y se indica una referencia a su definición, que se realiza en otro lugar del documento. Es una técnica que recuerda a los DTD.

Ejemplo:

```
<?xml version="1.0" encoding="io-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definición de elementos de tipo simple -->
  <xs:element name="titulo" type="xs:string"/>
  <xs:element name="autor" type="xs:string"/>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="amigoDe" type="xs:string"/>
  <xs:element name="desde" type="xs:date"/>
  <xs:element name="calificacion" type="xs:string"/>

  <!-- definición de atributos -->
  <xs:attribute name="isbn" type="xs:string"/>

  <!-- definición de elementos de tipo complejo -->
  <xs:element name="personaje">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <xs:element ref="amigoDe" minOccurs="0"
                                  maxOccurs="unbounded"/>
        <xs:element ref="desde"/>
        <xs:element ref="calificacion"/>

        <!-- los elementos de tipo simple se referencian con el
            atributo ref -->
        <!-- la definición de la cardinalidad se hace cuando el
            elemento es referenciado -->
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="titulo"/>
        <xs:element ref="autor"/>
        <xs:element ref="personaje" minOccurs="0"
                                    maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="isbn"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- Diseño con tipos con nombre reutilizables: se definen tipos de datos simples o complejos a los que se identifica con un nombre (son plantillas, como las clases en la programación orientada a objeto). Al declarar elementos y atributos, se indica que son de alguno de los tipos con nombre previamente definidos (aquí elementos y atributos son instancias de las plantillas ya definidas, como objetos en la programación orientada a objeto).

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- definición de tipos simples -->
  <xs:simpleType name="TipoNombre">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="TipoDesde">
    <xs:restriction base="xs:date"/>
  </xs:simpleType>
  <xs:simpleType name="TipoDescripcion">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="TipoISBN">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{10}"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- definición of tipos complejos -->
  <xs:complexType name="TipoPersonaje">
    <xs:sequence>
      <xs:element name="nombre" type="TipoNombre"/>
      <xs:element name="amigoDe" type="TipoNombre"
                                  minOccurs="0"
                                  maxOccurs="unbounded"/>
      <xs:element name="desde" type="TipoDesde"/>
      <xs:element name="calificacion" type="TipoDescripcion"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TipoLibro">
    <xs:sequence>
      <xs:element name="titulo" type="TipoNombre"/>
      <xs:element name="autor" type="TipoNombre"/>

      <!-- definición del elemento personaje, usando el tipo
            complejo TipoPersonaje -->
      <xs:element name="personaje" type="TipoPersonaje"
                                  minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="TipoISBN" use="required"/>
  </xs:complexType>
```

```
  <!-- definición del elemento libro usando el tipo complejo
      TipoLibro -->
  <xs:element name="libro" type="TipoLibro"/>
</xs:schema>
```

4.8.8. Poniendo todo junto

Se va a desarrollar un ejemplo de esquema para recopilar los elementos más usados.

Ejemplo:

Se dispone del siguiente documento XML, *persona.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<persona dni="12345678-L" xsi:noNamespaceSchemaLocation="persona.xsd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <nombre>Juan Antonio</nombre>
  <apellido>Abascal</apellido>
  <estadoCivil>Soltero</estadoCivil>
  <edad>60</edad>
  <enActivo/>
</persona>
```

El esquema XML asociado se quiere que cumpla ciertas restricciones semánticas:

- El atributo `dni` es obligatorio y su formato es de 8 dígitos, un guion y una letra mayúscula.
- El estado civil puede ser: *Soltero*, *Casado*, *Divorciado* o *Viudo*. Por defecto es *Soltero*.
- La edad debe ser de 0 a 150.
- El elemento `<enActivo>` es optativo.

El esquema que cumple con esto, ubicado en el archivo *persona.xsd*, es:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Declaración de elementos -->
  <xs:element name="persona">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellido" type="xs:string"/>
        <xs:element name="estadoCivil" type="TipoEstadoCivil"
                    default="Soltero"/>
        <xs:element name="edad" type="TipoEdadHumana"/>
        <xs:element name="enActivo" minOccurs="0">
          <xs:complexType/>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="dni" type="TipoDni" use="required"/>
    </xs:complexType>
  </xs:element>
```

```
  <!-- Definición de tipos -->
  <xs:simpleType name="TipoDni">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{8}\-[A-Z]"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="TipoEstadoCivil">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Soltero"/>
      <xs:enumeration value="Casado"/>
      <xs:enumeration value="Divorciado"/>
      <xs:enumeration value="Viudo"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="TipoEdadHumana">
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="150"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

4.8.9. Construcción avanzada de esquemas

Se verán a continuación elementos complementarios en la construcción de esquemas, que permiten alcanzar altos niveles de sofisticación en su diseño.

Grupos de elementos y grupos de atributos

Se pueden definir grupos de elementos y grupos de atributos para poder referenciarlos en definiciones posteriores de tipos compuestos.

Estos grupos no son tipos de datos sino contenedores que albergan un conjunto de elementos o atributos que pueden ser usados en la definición de tipos complejos. Se utilizarán los componentes `<xs:group>` y `<xs:attributeGroup>`.

Ejemplo:

```
<!-- definición de un grupo de elementos -->

<xs:group name="ElementosPrincipalesLibro">
  <xs:sequence>
    <xs:element name="titulo" type="TipoNombre"/>
    <xs:element name="autor" type="TipoNombre"/>
  </xs:sequence>
</xs:group>



<!-- definición de un grupo de atributos -->
```

```
<xs:attributeGroup name="AtributosLibro">
  <xs:attribute name="isbn" type="TipoISBN" use="required"/>
  <xs:attribute name="disponible" type="xs:string"/>
</xs:attributeGroup>

<!-- Los grupos se usan en la definición de un tipo complejo -->
<xs:complexType name="TipoLibro">
  <xs:sequence>
    <xs:group ref="ElementosPrincipalesLibro"/>
    <xs:element name="personaje" type="TipoPersonaje"
           minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attributeGroup ref="AtributosLibro"/>
</xs:complexType>
```

Redefinición de tipos

Este elemento nos va permitir redefinir en un esquema un tipo de datos, simple o complejo, un grupo (de elementos) o un grupo de atributos, que hubieran sido ya definidos en otro documento de esquema al que se referencia. Es semejante a una extensión o herencia de una clase en programación orientada a objeto.

xs:redefine

Es un componente de redefinición de tipos de datos existentes en un documento de esquema que se incluirá. Permite redefinir tipos de datos y grupos de elementos y atributos. Funciona de manera parecida al componente de inclusión de esquemas, `xs:include`.

Elemento padre: `xs:schema`

Atributos obligatorios:

- `schemaLocation`: URI del documento de esquema al que se referencia.

Atributos optativos principales:

- `id`: especifica un identificador único para el elemento.

Ejemplo:

Se define en un esquema un tipo de datos complejo, `TipoTrayecto`, que contiene los elementos `origen` y `destino`. Posteriormente, en otro esquema se quiere hacer uso de este tipo como base para definir otro tipo que, además, contiene un elemento `duracion`.

El primer esquema, almacenado en el documento `esquemaInicial.xsd`, contendrá:

```
<xs:complexType name="TipoTrayecto">
  <xs:sequence>
    <xs:element name="origen"/>
    <xs:element name="destino"/>
  </xs:sequence>
</xs:complexType>
```

El segundo esquema, donde se redefine (reusa) el tipo de datos `TipoTrayecto`, se almacena en otro archivo, por ejemplo `esquemaAmpliado.xsd`. Al nuevo tipo de datos se le llamará como al que extiende, `TipoTrayecto`:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:redefine schemaLocation="esquemaInicial.xsd">
  <xs:complexType name="TipoTrayecto">
    <xs:complexContent>
      <xs:extension base="TipoTrayecto">
        <xs:sequence>
          <xs:element name="duracion"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
<xs:element name="ruta" type="TipoTrayecto"/>

</xs:schema>
```

Grupos de sustitución

Permiten definir elementos que son sinónimos. Un uso muy habitual el para definir los mismos elementos en varios idiomas.

Ejemplo:

Datos de un cliente en castellano y en inglés. El atributo `substitutionGroup` en un elemento indica que es del mismo tipo que el valor de ese atributo. En el ejemplo, `<name>` es equivalente a `<nombre>` y `<customer>` a `<cliente>`.

```
<xs:element name="nombre" type="xs:string"/>
<xs:element name="name" substitutionGroup="nombre"/>

<xs:complexType name="TipoInfoCliente">
  <xs:sequence>
    <xs:element ref="nombre"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="cliente" type="TipoInfoCliente"/>
<xs:element name="customer" substitutionGroup="cliente"/>
```

Dos fragmentos XML igualmente válidos:

```
<cliente><nombre>Isabel Sánchez</nombre></cliente>
```

Y también:

```
<customer><name>José R. Rodríguez</name></customer>
```

xs:any

Permite que en el documento XML aparezcan elementos que no han sido explícitamente declarados en el esquema.

Elemento padre: `xs:choice`, `xs:sequence`.

Atributos opcionales:

- `id`: identificador único del elemento.
- `maxOccurs`: indica el número máximo de ocasiones en las que el elemento puede aparecer como descendiente del elemento padre. El valor por defecto es 1, pero puede tomar cualquier valor mayor o igual que cero. Si no se quiere poner límite se usará *unbounded* (ilimitado).
- `minOccurs`: indica el número mínimo de ocasiones en las que el elemento puede aparecer como descendiente del elemento padre. El valor por defecto es 1, pero puede tomar cualquier valor mayor o igual que cero.
- `namespace`: especifica los espacios de nombres que contienen los elementos que pueden usarse. Puede valer:
    - `##any`: se permiten elementos de cualquier espacio de nombres. Es el valor por defecto.
    - `##other`: se permiten elementos de cualquier espacio de nombres que no sea el del elemento padre.
    - `##local`: se permiten elementos sin espacio de nombres.
    - `##targetNamspace`: se permiten elementos del espacio de nombres del elemento padre.
    - Una lista de {URI que referencien espacios de nombres, `##targetNamspace`, `##local`}: se permiten elementos de una lista (delimitada por espacios) de espacios de nombres.
- `processContents`: indica cómo debe realizar la validación el procesador de XML frente a los elementos especificados por este anyElement. Puede valer:
    - strict: el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
    - lax: igual que el strict, pero si no se obtiene el esquema no se producirá un error.
    - skip: el procesador XML no intenta validar ningún elemento de los espacios de nombres especificados.

Ejemplo:

Se declara un elemento `<coche>`, con dos atributos `marca` y `modelo`. Al usar el elemento `<xs:any>` en los documentos XML validados por este esquema se podrán añadir otros elementos, colocados detrás de `modelo`, al elemento `coche`.

```
<xs:element name="coche">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo" type="xs:string"/>
      <xs:any minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

xs:anyAttribute

Permite que en el documento XML aparezcan asociados a un elemento atributos que no han sido explícitamente declarados en el esquema.

Elemento padre: `xs:complexType`, `xs:restriction` (`xs:simpleContent` y `xs:complexContent`), `xs:extension` (`xs:simpleContent` y `xs:complexContent`), `xs:attibuteGroup`.

Atributos opcionales:

- `id`: identificador único del elemento.
- `namespace`: especifica los espacios de nombres que contienen los atributos que pueden usarse. Puede valer:
  - `##any`: se permiten atributos de cualquier espacio de nombres. Es el valor por defecto.
  - `##other`: se permiten atributos de cualquier espacio de nombres que no sea el del elemento padre.
  - `##local`: se permiten atributos sin espacio de nombres.
  - `##targetNamspace`: se permiten atributos del espacio de nombres del elemento padre.
  - Una lista de {URI que referencien espacios de nombres, `##targetNamspace`, `##local`}: se permiten atributos de una lista (delimitada por espacios) de espacios de nombres.
- `processContents`: indica cómo debe realizar la validación el procesador de XML frente a los atributos especificados por este anyAttribute. Puede valer:
  - strict: el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
  - lax: igual que el strict, pero si no se obtiene el esquema no se producirá un error.
  - skip: el procesador XML no intenta validar ningún elemento de los espacios de nombres especificados.

Ejemplo:

Se declara un elemento `<coche>`, con dos atributos `marca` y `modelo`. Al usar el elemento `<xs:anyAttribute>` en los documentos XML validados por este esquema se podrán añadir otros atributos al elemento coche.

```
<xs:element name="coche">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute />
  </xs:complexType>
</xs:element>
```

Documentación de esquemas

Para asociar a un esquema comentarios que puedan ser utilizados para generar una documentación con alguna herramienta de autor, se utilizan una serie de instrucciones: `xs:annotation`, `xs:appinfo` y `xs:documentation`.

xs:annotation

Permite especificar comentarios al esquema.

Elemento padre: cualquiera

Sintaxis: xs:annotation::= (xs:appinfo | xs:documentation)*

Atributos opcionales:

- id: identificador único del elemento.

xs:appinfo

Especifica información que vaya a ser usada por la aplicación.

Elemento padre: `xs:annotation`

Sintaxis: xs:appinfo::= Contenido XML bien formado

Atributos opcionales:

- `source`: una URI que especifica el origen de la información sobre la aplicación.

xs:documentation

Se usa para asociar comentarios textuales sobre el esquema.

Elemento padre: `xs:annotation`

Sintaxis: xs:documentation::= Contenido XML bien formado

Atributos opcionales:

- `source`: una URI que especifica el origen de la información sobre la aplicación.

Ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:annotation>
    <xs:appInfo>Nota</xs:appInfo>
    <xs:documentation xml:lang="sp">
      Este esquema tiene una nota
    </xs:documentation>
  </xs:annotation>
  ...
</xs:schema>
```

xs:include

Es un componente de inclusión de esquemas externos. Añade las declaraciones y definiciones de un esquema externo al esquema actual. El documento de esquema externo debe tener el mismo espacio de nombres que el esquema actual. Se utiliza para reutilizar tipos ya definidos, en ocasiones en librerías de tipos.

Elemento padre: `xs:schema`

Atributos obligatorios:

- `schemaLocation`: URI del esquema a incluir en el espacio de nombres del esquema contenedor.

Atributos optativos principales:

- `id`: especifica un identificador único para el elemento.

Ejemplo:

Se incluyen dos esquemas en otro que actúa como contenedor. El espacio de nombres de los esquemas incluidos debe ser el mismo, sino no funcionará la inclusión.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include
    schemaLocation="http://www.esquemas.com/empleados.xsd">
  <xs:include
    schemaLocation="http://www.esquemas.com/departamentos.xsd">
  ...
</xs:schema>
```

xs:import

Es un componente de importación de esquemas externos. Ofrece la misma funcionalidad que `xs:include`, a excepción que el esquema importado tiene diferente espacio de nombres que el actual.

Elemento padre: `xs:schema`

Atributos optativos principales:

- `id`: especifica un identificador único para el elemento.

- `namespace`: indica el URI del espacio de nombres a importar.
- `schemaLocation`: especifica el URI del esquema del espacio de nombres importado.

Ejemplo:

Se importa un espacio de nombres con el prefijo *xhtml:* para referenciar a un párrafo de XHTML, `<p>`, que aparece en `http://www.w3.org/1999/xhtml`.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xhtml="http://www.w3.org/1999/xhtml"
           targetNamespace="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="http://www.w3.org/1999/xhtml"/>
  ...
  <xs:complexType name="UnTipoComplejo">
    <xs:sequence>
      <xs:element ref="xhtml:p" minOccurs="0"/>
      ...
    </xs:sequence>
  </xs:complexType>
  ...
<xs:schema>
```

Control de la integridad referencial

Mediante una serie de instrucciones se va a poder simular el comportamiento de las restricciones de clave primaria, de clave ajena y de unicidad, que permiten preservar la integridad referencial en los sistemas gestores de bases de datos.

La integridad referencial asegura que un elemento que deba estar relacionado con otro, no pueda estarlo con un elemento inexistente.

Ejemplo:

En una empresa los empleados pertenecen a departamentos que se identifican por números: 10, 20, 30... La integridad referencial controlará que un empleado no pueda ser asignado a un departamento inexistente.

Para lograr este efecto usaremos las instrucciones `xs:key`, `xs:keyref`, `xs:unique`. Todas ellas se apoyan para su construcción en las instrucciones `xs:selector` y `xs:field`.

xs:key

Permite definir un elemento o atributo como clave. El valor de la clave no puede ser nulo ni repetirse, de manera que identifica de manera única al elemento/atributo al que se asocia. Asímismo, el elemento o atributo que constituya la clave no podrá omitirse.

Es un comportamiento equivalente a una clave primaria en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con xs:keyref, que permitirá a otro elemento o atributo referenciar a la clave.

Sintaxis: xs:key::= xs:selector (xs:field)+

Elemento padre: `xs:element`

Atributos obligatorios:
- `name`: nombre de la clave.

Atributos optativos principales:
- `id`: especifica un identificador único para el elemento.

Ejemplo:

Se tiene un documento XML que representa un `pedido`. El `pedido` está compuesto de varios `producto`, cada uno de los cuales tienen un `identificador` cuyo valor es único en el `pedido`. Cada `línea` de pedido está compuesta de una referencia a un `producto`, la `cantidad` de unidades pedidas y un `color` opcional. Una instancia XML será:

```
<?xml version="1.0"?>
<pedido xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="pedido.xsd">
<codigo>123ABBCC123</codigo>
  <productos>
    <producto>
      <identificador>557</identificador>
      <nombre>Pantalón</nombre>
      <precio moneda="euro">35.99</precio>
    </producto>
    <producto>
      <identificador>563</identificador>
      <nombre>Abrigo</nombre>
      <precio moneda="euro">65.99</precio>
    </producto>
  </productos>

  <lineas>
    <linea identificador="557">
      <cantidad>2</cantidad>
      <color>Azul</color>
    </linea>
    <linea identificador="557">
      <cantidad>1</cantidad>
      <color>Gris</color>
    </linea>
    <linea identificador="563">
      <cantidad>1</cantidad>
    </linea>
  </lineas>
</pedido>
```

Un esquema muy detallado que valida este XML es:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xsi:noNamespaceSchemaLocation="pedido.xsd">
  <xs:element name="pedido" type="TipoPedido">
    <!-- Definición de una referencia a un elemento clave -->
    <xs:keyref name="prodNumKeyRef" refer="prodNumKey">
      <xs:selector xpath="lineas/linea"/>
      <xs:field xpath="@identificador"/>
    </xs:keyref>
    <!-- Definición de un elemento clave -->
    <xs:key name="prodNumKey">
      <xs:selector xpath="productos/producto"/>
      <xs:field xpath="identificador"/>
    </xs:key>
  </xs:element>
  <xs:complexType name="TipoPedido">
    <xs:sequence>
      <xs:element name="codigo" type="xs:string"/>
      <xs:element name="productos" type="TipoProductos"/>
      <xs:element name="lineas" type="TipoLineas"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TipoLineas">
    <xs:sequence>
      <xs:element name="linea" type="TipoLinea"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TipoLinea">
    <xs:sequence>
      <xs:element name="cantidad" type="xs:integer"/>
      <xs:element name="color" type="TipoColor" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="identificador" type="xs:integer"/>
  </xs:complexType>
  <xs:complexType name="TipoProductos">
    <xs:sequence>
      <xs:element name="producto" type="TipoProducto"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TipoProducto">
    <xs:sequence>
      <xs:element name="identificador" type="xs:integer"/>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="precio" type="TipoPrecio"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="TipoColor">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Azul"/>
      <xs:enumeration value="Gris"/>
      <xs:enumeration value="Beige"/>
```

```
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="TipoPrecio">
      <xs:simpleContent>
        <xs:extension base="xs:decimal">
          <xs:attribute name="moneda" type="xs:token"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>
```

xs:keyref

Permite referenciar a claves primarias (xs:key) o claves únicas (xs:unique), compuestas por elementos y/o atributos.

Es un comportamiento equivalente a una clave ajena en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con xs:key (con la que se define la clave primaria) o con xs:unique (con la que se define la clave única).

Sintaxis: xs:keyref::= xs:selector (xs:field)+

Elemento padre: `xs:element`

Atributos obligatorios:

- `name`: nombre de la referencia a la clave.
- `refer`: nombre de la clave primaria o clave única a la que hace referencia

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

xs:unique

Permite definir un elemento o atributo como de valor único.

Es un comportamiento equivalente a una clave única en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con xs:keyref, que permitirá a otro elemento o atributo referenciar a la clave única.

Elemento padre: `xs:element`

Sintaxis: xs:unique::= xs:selector (xs:field)+

Atributos obligatorios:

- `name`: nombre de la clave única.

Atributos optativos principales:

- `id`: especifica un identificador único para el elemento.

xs:selector

Especifica una expresión XPath que selecciona un conjunto de elementos para usarlos en la definición de una clave, sea primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: `xs:key`, `xs:keyref`, `xs:unique`

Sintaxis: xs:selector::= (annotation)?

Atributos obligatorios:

- `xpath`: especifica una expresión XPath, relativa al elemento que se está declarando, que identifica los elementos hijos a los cuales se aplica la restricción de identidad.

Atributos opcionales:

- `id`: especifica un identificador único del elemento

xs:field

Especifica una expresión XPath que indica los elementos o atributos que formarán parte de la clave que se esté definiendo, sea primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: `xs:key`, `xs:keyref`, `xs:unique`

Sintaxis: xs:field::= (annotation)?

Atributos obligatorios:

- `xpath`: identifica un único elemento o atributo cuyo contenido o valor se usa para la restricción.

Atributos opcionales:

- `id`: especifica un identificador único del elemento.

4.9. Otros mecanismos para validar XML

Existen otros mecanismos para validar documentos XML:

- Relax NG (http://www.oasis-open.org/committees/relax-ng)

- Schematron (http://www.xml.com/pub/a/2003/11/12/schematron.html)

4.10. Otros lenguajes basados en XML

Esisten lenguajes bsasados en XML que se usan para propósitos específicos. Estos lenguajes tendrán su mecanismo de validación (DTD, esquema XML...) que fijará qué elementos y atributos concretos pueden aparecer, con qué valores, en qué orden...

4.10.1. SVG

SVG (Scalable Vector Graphics – Gráficos Vectoriales Escalables) es un lenguaje de representación de gráficos vectoriales bidimensionales. Desde el año 2001 es una recomendación del W3C, por lo que muchos navegadores son capaces de mostrar gráficos en este formato.

Ejemplo:

```
<?xml version="1.0"?>
<?xml-stylesheet href="formas.css" type="text/css"?>
<doc>
<canvas>
  <shape x1="100" y1="100" x2="300" y2="200" x3="200" y3="400">
    <cline x1var="x1" y1var="y1"
           x2var="x2" y2var="y2" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <cline x1var="x1" y1var="y1"
           x2var="x3" y2var="y3" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <cline x1var="x3" y1var="y3"
           x2var="x2" y2var="y2" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <controlpoint xvar="x1" yvar="y1"/>
    <controlpoint xvar="x2" yvar="y2"/>
    <controlpoint xvar="x3" yvar="y3"/>
  </shape>
  <shape x1="200" y1="200" x2="500" y2="300" cx1="100"
         cy1="150" cx2="450" cy2="100">
    <curve x1var="x1" y1var="y1"
           x2var="x2" y2var="y2"
           cx1var="cx1" cy1var="cy1"
           cx2var="cx2" cy2var="cy2"
           style="stroke:green; fill:none; stroke-width:4;"/>
    <cline x1var="x1" y1var="y1"
           x2var="cx1" y2var="cy1" style="stroke:blue; fill:none;
           stroke-width:1;"/>
    <cline x1var="x2" y1var="y2"
           x2var="cx2" y2var="cy2" style="stroke:blue; fill:none;
           stroke-width:1;"/>
    <controlpoint xvar="x1" yvar="y1"/>
    <controlpoint xvar="x2" yvar="y2"/>
    <controlpoint xvar="cx1" yvar="cy1"/>
    <controlpoint xvar="cx2" yvar="cy2"/>
  </shape>
</canvas>
</doc>
```

Nótese que utiliza etiquetas propias que describen elementos de dibujo, como `<canvas>` (lienzo), `<shape>` (forma) o `<curve>` (curva).

Por otra parte, igual que se ha visto en los ejemplos de XML, se aplica al documento una hoja de estilos CSS externa. Esto se ve en la segunda línea del código. Además, y de forma análoga a HTML, existen elementos que disponen del atributo `style`, que permitirá aplicarles estilos.

4.10.2. WML

WML (**W**ireless **M**ark-up **L**anguage – Lenguaje de Marcado Inalámbrico) es un lenguaje de representación de la información que se visualiza en las pantallas de dispositivos móviles que utilicen el protocolo WPA (**W**ireless **A**pplication **P**rotocol – Protocolo de Aplicaciones Inalámbricas). Un ejemplo de este lenguaje es:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
          "http://www.wapforum.org/DTD/wml13.dtd">
<wml>
  <card id="card1" title="Formulario">
    <onevent type="onenterforward">
      <refresh>
        <setvar name="elNombre" value=""/>
        <setvar name="elGenero" value=""/>
        <setvar name="lasAficiones" value=""/>
      </refresh>
    </onevent>
    <onevent type="onenterbackward">
      <refresh>
        <setvar name="elNombre" value=""/>
        <setvar name="elGenero" value=""/>
        <setvar name="lasAficiones" value=""/>
      </refresh>
    </onevent>
    <p>
      Encuesta simple<br/>
      ¿Cuál es tu nombre?<br/><input name="elNombre"/><br/>
      ¿Eres hombre o mujer?<br/>
      <select name="elGenero">
        <option value="Hombre">Soy hombre</option>
        <option value="Mujer">Soy mujer</option>
      </select><br/>
      ¿Cuáles son tus aficiones?<br/>
      <select name="lasAficiones" multiple="true">
        <option value="Viajes">Viajes</option>
        <option value="Deporte">Deporte</option>
        <option value="Lectura">Lectura</option>
        <option value="Cine">Cine</option>
        <option value="Gastronomía">Gastronomía</option>
      </select><br/>
      <anchor>
        <go method="get" href="procesarFormulario.asp">
          <postfield name="nombre" value="$(elNombre)"/>
          <postfield name="genero" value="$(elGenero)"/>
          <postfield name="aficiones" value="$(lasAficiones)"/>
        </go>
        Enviar
      </anchor>
    </p>
  </card>
</wml>
```

Como se ve, hay etiquetas que son específicas de este lenguaje, como `<card>`, `<onevent>` o `<setvar>`. Otras son análogas a las de otros lenguajes de marcas como HTML, por ejemplo `<input>`, `<select>` o `<br />`.

4.10.3. RSS

RSS (Really Simple Syndication – Sindicación Realmente Simple) es una familia de formatos de semillas web usadas para publicar información que se actualiza con frecuencia.

Más adelante se dedicará un capítulo a cubrir en detalle esta tecnología.

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
  <channel>
    <title>Ejemplo RSS</title>
    <description>Esto es un ejemplo de una semilla RSS</description>
    <link>http://www.domain.com/link.htm</link>
    <lastBuildDate>Fri, 11 May 2012 11:30:50 -0400 </lastBuildDate>
    <pubDate>Sat, 12 May 2012 09:00:00 -0400 </pubDate>
    <item>
      <title>Ejemplo de ítem </title>
      <description>Esto es un ejemplo de un ítem</description>
      <link>http://www.domain.com/link.htm</link>
      <guid isPermaLink="false">1102345</guid>
      <pubDate>Sat, 12 May 2012 09:00:00 -0400 </pubDate>
    </item>
  </channel>
</rss>
```

4.10.4. Atom

Atom (**Atom** Syndication Format – Formato Atómico de Sindicación) es un lenguaje usado como semillas web. Es una alternativa a RSS.

4.10.5. DocBook

Es un lenguaje de definición de documentos. Una herramienta interesante es XMLMind XML Editor (http://www.xmlmind.com/xmleditor).

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
       "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book lang="es" id="libro">
  <title>Un libro muy simple</title>
  <chapter id="capitulo1">
    <title>Capitulo 1</title>
    <para>Por algún sitio hay que comenzar</para>
    <para>Y este es un buen lugar</para>
```

```
    </chapter>
    <chapter id="capitulo2">
      <title>Capitulo 2</title>
      <para>Por algún sitio hay que continuar</para>
      <para>Y este parece otro buen lugar</para>
    </chapter>
  </book>
```

4.10.6. XBRL

XBRL (**E**xtensible **B**usiness **R**eporting **L**anguage – Lenguaje Extensible de Informes Financieros) es un estándar abierto que permite representar la información y la expresión de la semántica requerida en los informes financieros.

4.11. Otras formas de almacenar información

4.11.1. JSON

JSON (**J**avaScript **O**bject **N**otation – Notación de Objetos JavaScript) es, de igual manera que XML, un formato en modo texto para el almacenamiento y transmisión de información.

Se usa frecuentemente como alternativa a XML en el envío de datos, por ejemplo en **AJAX** (**A**synchronous **J**avaScript **A**nd **X**ML – JavaScript Asíncrono y XML). El motivo fundamental es que el procesado de un documento en formato JSON es más sencillo que el de un documento XML.

Aunque se suelen ver como antagónicas, en ocasiones se usan de manera conjunta ambas tecnologías.

Permite representar objetos en aplicaciones RIA (**R**ich **I**nternet **A**pplication – Aplicaciones Ricas de Internet) que usan JavaScript.

El término JSON se sustituye en ocasiones por el de LJS (**L**iteral **J**avaScript).

JSON se apoya en dos estructuras de datos:

- Una **colección** de pares nombre/valor. En algunos lenguajes de programación a esto se le denomina objeto, registro, diccionario, tabla hash o **array asociativo**.
- Una **lista** ordenada de valores, lo que en otros lenguajes se denomina array, vector, lista o secuencia.

Un objeto es un conjunto desordenado de pares etiqueta/valor. La definición de un objeto comienza con la llave de apertura, "{", y termina con la llave de cierre, "}". Cada nombre va seguido del carácter dos puntos ":" y los pares nombre/valor se separan por el carácter coma ",".

Un array es una colección ordenada de valores. Comienza por el carácter corchete de apertura "[" y termina por el carácter corchete de cierre "]". Los valores se separan por el carácter coma ",".

Un valor puede ser:

- una cadena de texto entre comillas dobles
- un número
- `true`, `false` o `null`
- un objeto
- un array

Una cadena de texto es una secuencia de cero o más caracteres Unicode, rodeados por dobles comillas. Se usa el carácter barra invertida (\) como marca de escape. Un carácter se representa como una cadena de un solo carácter. Es equivalente al *String* de Java.

Un número es como los números en Java. Puede tener parte entera, entera + decimal, entera + exponente y entera + decimal + exponente.

Sintaxis:

objeto::= { } | { miembros }

miembros::= par | par, miembros

par::= string : valor

array::= [ ] | [elementos]

elementos::= valor | valor, elementos

valor::= string | número | objeto | array | true | false | null

Existen herramientas de conversión automática entre JSON y XML:

- http://json.online-toolz.com/tools/xml-json-convertor.php
- http://jsontoxml.utilities-online.info

Ejemplo:

Se dispone de un array que contiene dos objetos (llámense país). Cada uno de ellos tiene varios pares nombre/valor (atributos con sus valores), a saber, pais, población y animales. A su vez, animales tiene como valor un array de valores.

```
[
  {
    "pais" : "Nueva Zelanda",
    "poblacion" : 3993817,
    "animales" : ["Oveja", "Kiwi"]
  },
  {
    "pais" : "Singapur",
    "poblacion" : 4553893,
    "animales" : ["Tigre"]
  }
]
```

Nº caracteres totales: 135 (se excluyen todos los espacios aparecidos para formatear)

Nº caracteres de información: 49

Porcentaje de caracteres de información respecto al total: 49 / 135 = 36,29 %

La información equivalente en formato XML sería:

```
<paises>
  <pais>
    <nombre>Nueva Zelanda</nombre>
    <poblacion>3993817</poblacion>
    <animales>
      <animal>Oveja</animal>
      <animal>Kiwi</animal>
    </animales>
  </pais>
  <pais>
    <nombre>Singapur</nombre>
    <poblacion>4553893</poblacion>
    <animales>
      <animal>Tigre</animal>
    </animales>
  </pais>
</paises>
```

Nº caracteres totales: 255 (se excluyen todos los espacios aparecidos para formatear)

Nº caracteres de información: 49

Porcentaje de caracteres de información respecto al total: 49 / 255 = 19,21%

4.11.2. YAML

YAML (acrónimo recursivo de **Y**AML **A**in't **M**arkup **L**anguage – YAML no es otro lenguaje de marcas, aunque también se refiere como **Y**et **A**nother **M**arkup **L**anguage), es un formato de almacenamiento de información o serialización de datos, ligero y de fácil lectura para el ojo humano. Comparte con JSON ciertos elementos como las **listas** y los **arrays asociativos**.

Ejemplo:

Una lista de frutas en **formato de bloque** donde cada elemento se denota por un guion "-"; y en **formato lineal**, donde los elementos se rodean de un corchete de apertura "[" y uno de cierre "]" y se separan entre sí por comas ",":

- Mandarina
- Fresa
- Sandía

[Mandarina, Fresa, Sandía]

Ejemplo:

Un array asociativo en **formato de bloque**, en el que aparece la etiqueta, el carácter dos pontos ":" y el valor; y en **formato lineal**, donde los pares etiqueta/valor se rodean por una llave de apertura "{" y una de cierra "]}" y se separan entre sí por comas ".":

nombre: Casilda

apellido: Marín

{nombre: Casilda, apellido: Marín}

A partir de estos elementos básicos se pueden construir estructuras más complejas: listas de arrays asociativos, arrays asociativos de listas...

Ejercicios propuestos

1. Se quiere guardar en formato XML la información relativa a tickets de compra, según las siguientes especificaciones:

- Datos del ticket
    - Código del ticket, en un atributo requerido de tipo id
    - Fecha y hora
    - Precio total:
        - Precio sin IVA total
        - Cantidad IVA
        - PVP total
        - La moneda se guarda como atributo de tipo token
    - Forma de pago: puede ser en efectivo o con tarjeta
        - Si es con tarjeta:
            - Tipo de tarjeta, en un atributo
            - 12 asteriscos y los cuatro últimos dígitos de la tarjeta, en un atributo
            - Nombre del cliente
- Datos del comercio
    - Nombre
    - Dirección completa
    - CIF
    - Teléfono
- Datos de la compra:
    - Líneas de compra:
        - Nombre del artículo
        - Cantidad
        - Precio unitario (sin IVA)
        - IVA
        - PVP

Se ofrece esta instancia XML de prueba:

```
<?xml version="1.0"?>
<ticket código="t00037">
  <datos_ticket>
    <fecha>
```

```
    <hora>
    <precio_total moneda="euro">
      <cantidad_sin_iva></sin_iva>
      <cantidad_iva></cantidad_iva>
      <cantidad_con_iva></cantidad_con_iva>
    </precio_total>
    <forma_pago número="************1234" tipo="Visa Clásica">
      Tarjeta
    </forma_pago>
    <cliente>Pedro Medario</cliente>
  </datos_ticket>

      <fecha>2000-02-02</fecha>
      <estadio espectadores="49000">Lansdowne Road</estadio>
      <local>Irlanda</local>
      <visitante>Inglaterra</visitante>
      <resultado local="17" visitante="6" />
    </partido>
    <partido numero="2">
      <fecha>2000-02-03</fecha>
      <estadio>
        El parque de los príncipes</estadio>
      <local>Francia</local>
      <visitante>Gales</visitante>
      <aplazado />
    </partido>
    ...
  </jornada>
  <jornada numero="2">
    ...
  </jornada>
  ...
</temporada>
```

Se pide:

- Escribir otro documento XML que se ajuste a las especificaciones dadas pero que contenga alguna variación en la estructura. Así se tienen dos instancias del mismo modelo.

2. Crea el DTD que valide las especificaciones dadas en el ejercicio anterior, de las cuales el documento XML generado no es más que un caso concreto.

   Se recomienda hacer el DTD incremental por secciones del documento, es decir, crear un DTD para un supuesto documento que incluyera sólo los datos del ticket y probarlo con un documento XML cuyo elemento raíz se sea `<datos_ticket>`. Haz lo mismo con los datos del comercio y con los datos de la compra. Por último, intégralos todos.

3. Crea el esquema XML equivalente al DTD del ejercicio anterior. También se puede hacer el diseño incremental.

4. Dado el siguiente fragmento XML, que representa la información relativa a los partidos jugados en una temporada en el torneo de rugby de las seis naciones, genera el DTD que mejor lo valide.

```
<?xml version="1.0"?>
<temporada año="2000">
  <jornada numero="1">
    <partido numero="1">
      <fecha>2000-02-02</fecha>
      <estadio espectadores="49000">Lansdowne Road</estadio>
      <local>Irlanda</local>
      <visitante>Inglaterra</visitante>
      <resultado local="17" visitante="6" />
    </partido>
    <partido numero="2">
      <fecha>2000-02-03</fecha>
      <estadio>
        El parque de los príncipes</estadio>
      <local>Francia</local>
      <visitante>Gales</visitante>
      <aplazado />
    </partido>
    ...
  </jornada>
  <jornada numero="2">
    ...
  </jornada>
  ...
</temporada>
```

Se cumplen las siguientes condiciones:

a. Una jornada tiene 3 partidos.

b. Una temporada tiene 5 jornadas.

c. Si un partido se juega, aparece la etiqueta `<resultado>`, con los puntos del equipo local y del visitante, así como el atributo `espectadores` del elemento `<estadio>`, que representa el público asistente; y si se suspende, aparece `<aplazado>` y no aparece `espectadores`.

d. El atributo `espectadores` tendrá un valor por defecto de 0 y como máximo puede ser 80.000.

e. Los países que juegan son siempre los mismos: Inglaterra, Francia, Irlanda, Escocia, Gales e Italia.

5. Genera el esquema XML que valide el documento del ejercicio anterior de una manera más precisa.

6. Genera un documento XML que represente un currículo. Deberá contener las siguientes secciones:

   a. Datos personales: nombre, apellidos, fecha y lugar de nacimiento, nacionalidad(es), número de identificación (nif o nie) y nombre de un archivo que represente la foto.

b. Datos de contacto: tipo de vía, nombre de la vía, número (optativo), portal (optativo), escalera (optativo), piso, puerta (optativo), código postal, país, email, teléfono móvil, teléfono fijo (optativo).

c. Datos de contacto adicionales: página web, cuentas de redes sociales: LinkedIn, Facebook, Twitter, etc.

d. Formación: para cada estudio realizado, nombre del mismo, lugar de realización, fecha de inicio y fecha de fin.

e. Idiomas: para cada idioma, nombre del mismo, nivel (alto, medio, bajo) de expresión oral (optativo), nivel de comprensión oral (optativo), nivel de expresión escrita (optativo), nivel de comprensión escrita (optativo). Hay que indicar de cada idioma si es materno.

f. Experiencia laboral: para cada experiencia, lugar de la misma, puesto desempeñado (optativo), fecha de comienzo y fecha de fin.

g. Competencias socio-profesionales: para cada competencia socio-profesional (liderazgo, trabajo en equipo, iniciativa, etc.), nombre de la misma y nivel de 1 (muy bajo) a 5 (muy alto).

h. Datos adicionales: aficiones, disponibilidad para viajar, vehículo propio, licencia(s) de conducir, etc.

Rellena el documento con los datos personales y comprobar que está bien formado.

7. Construye un esquema que valide el documento XML del ejercicio anterior. Se recomienda usar alguna herramienta de inferencia para generar el armazón del esquema y luego refinarlo con las reglas de negocio adecuadas.

8. Genera un documento XML de ejemplo que sea válido con respecto al siguiente DTD.

```
<!ELEMENT mensaje ( email | carta ) >
<!ELEMENT email ( cabecera, asunto?, texto+) >
<!ATTLIST email respuesta ( sí | no ) "no" >
<!ELEMENT carta ( encabezado, texto ) >
<!ATTLIST carta respuesta ( sí | no ) "no" >
<!ELEMENT cabecera ( emisor, receptor*, fecha?) >
<!ELEMENT asunto ( #PCDATA) >
<!ELEMENT texto ( #PCDATA | saludo )* >
<!ELEMENT encabezado ( emisor, receptor*, fecha ) >
<!ELEMENT emisor ( #PCDATA ) >
<!ELEMENT receptor ( #PCDATA ) >
<!ELEMENT fecha ( #PCDATA ) >
<!ELEMENT saludo ( #PCDATA ) >
```

9. Convierte el anterior DTD en un esquema.

10. Construye los tipos de datos de esquemas XML más adecuados para representar:

    a. Un número de teléfonos con prefijo internacional, como el +34.91.234.56.78.

    b. Una matrícula de coche actual española, como la 1234-BCD.

    c. Un dni o un nie, que sean alternativa el uno del otro.

    d. Un correo electrónico.

CAPÍTULO 5

Tratamiento y recuperación de datos

CONTENIDO

Introducción
Bases de datos XML nativas
XPath
XQuery
Otras tecnologías complementarias: XLink y XPointer
Bases de datos relacionales con XML
Manejo de XML desde Java
Ejercicios propuestos

5.1. Introducción

En este capítulo se revisarán una serie de contenidos bastante variados.

Se introducirán **las bases de datos XML nativas**, que son una alternativa a las relacionales en la que se sustituyen las tablas con sus campos por documentos XML con sus elementos y atributos.

Se utilizará una de ellas, **BaseX**, como herramienta para aprender dos lenguajes de consulta de documentos XML:

- Por un lado se conocerá **XPath**, un lenguaje sencillo de expresiones que permite acceder a partes de un documento XML.
- Por otro lado se aprenderá **XQuery**, otro lenguaje, que hace uso de XPath, que facilitará la manipulación de documentos XML. Se podrá extraer partes de un documento, generar documentos nuevos a partir de los datos de los originales, etc.

Se citarán dos tecnologías no muy usadas para conectar documentos XML: XLink y XPointer

Se verán ejemplos de **sistemas gestores de bases de datos que soporten el almacenamiento de XML**, como Oracle.

Por último, se conocerán brevemente algunas clases que ofrece un lenguaje de alto nivel como **Java para procesar documentos XML**: creación, recorrido, manipulación...

Este es un **capítulo accesorio** del libro, pero un contenido prioritario que no debería dejar de verse es el lenguaje de expresiones XPath, al menos en sus expresiones más sencillas, puesto que se usará como herramienta de otras tecnologías importantes como XSLT. Se aconseja para su aprendizaje el uso de la interfaz gráfica de BaseX, un sencillo sistema gestor de bases de datos XML nativo.

5.2. Bases de datos XML nativas

En oposición a los sistemas gestores de bases de datos tradicionales, como MySQL u Oracle, basados en tablas, compuestas de registros, que se relacionan entre sí, existen sistemas de bases de datos nativos XML, lo que significa que lo que se almacena son documentos XML

Algunos ejemplos son BaseX (http://basex.org) o eXist (http://exist-db.org) o Qizx (http://www.xmlmind.com/qizx).

Las bases de datos relacionales son más adecuadas para almacenar datos. Las bases de datos nativas XML son más adecuadas para almacenar documentos.

5.3.1. BaseX

BaseX es un motor de bases de datos nativo XML, ligero, de alto rendimiento en las operaciones y fácilmente escalable. Incluye, así mismo, procesadores de XPath y XQuery.

Con BaseX se puede crear una base de datos que está constituida por uno o más documentos XML. Ya no son tablas como en los sistemas gestores de bases de datos relacionales, sino documentos XML.

Luego, con diversos lenguajes de consulta, como XPath o XQuery, se podrá acceder al contenido de estos documentos.

Se puede visualizar el documento XML con el que se está trabajando de diferentes maneras, llamadas vistas, que se encuentran sincronizadas entre sí:

- En texto plano
- Como un mapa
- Como un árbol
- Como una serie de carpetas (tipo sistema de ficheros)
- Como una tabla
- Como un diagrama de dispersión

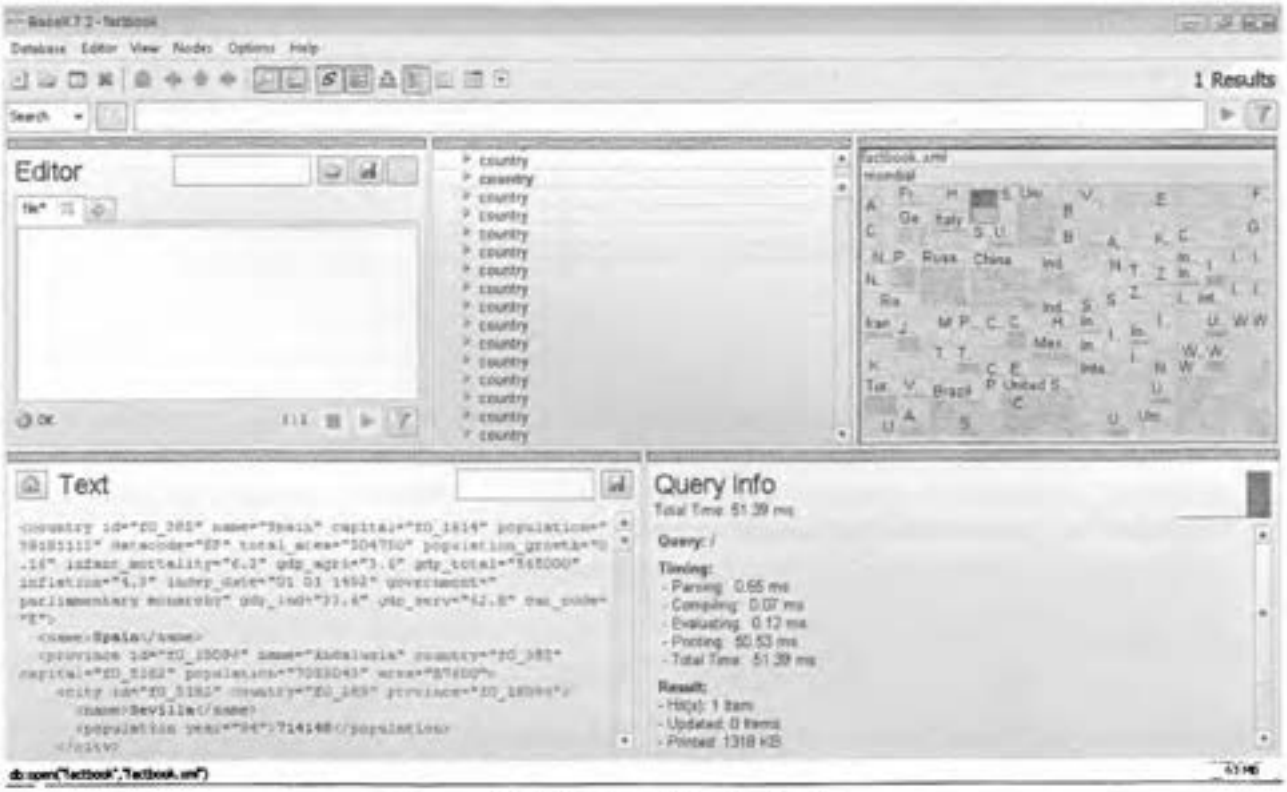


Figura 5.1: Interfaz gráfica de BaseX

En la *Figura 5.1* se pueden observar las secciones más importantes de la interfaz gráfica de BaseX:

- Barra de menús en la parte superior.
- Barra de herramientas (botones) inmediatamente debajo.
- Línea de comandos justo debajo.
- Editor de consultas.
- Diversas visualizaciones de los datos, en concreto de árbol, de mapa y de texto.
- Información sobre las consultas ejecutadas (*Query Info*).

En el interfaz gráfico de BaseX se permite ejecutar tres tipos de sentencias:

- Opción *Command*: Órdenes propias de cualquier sistema gestor. Existen comandos propios de un sistema gestor como `CREATE DB`, `OPEN`, `CREATE INDEX`, `CREATE USER`, `ADD`, `DELETE`, `REPLACE`...
- Opción *Search*: Expresiones XPath, un lenguaje de expresiones se verá en detalle.
- Opción *XQuery*: XQuery es un lenguaje de consulta que también se analizará.

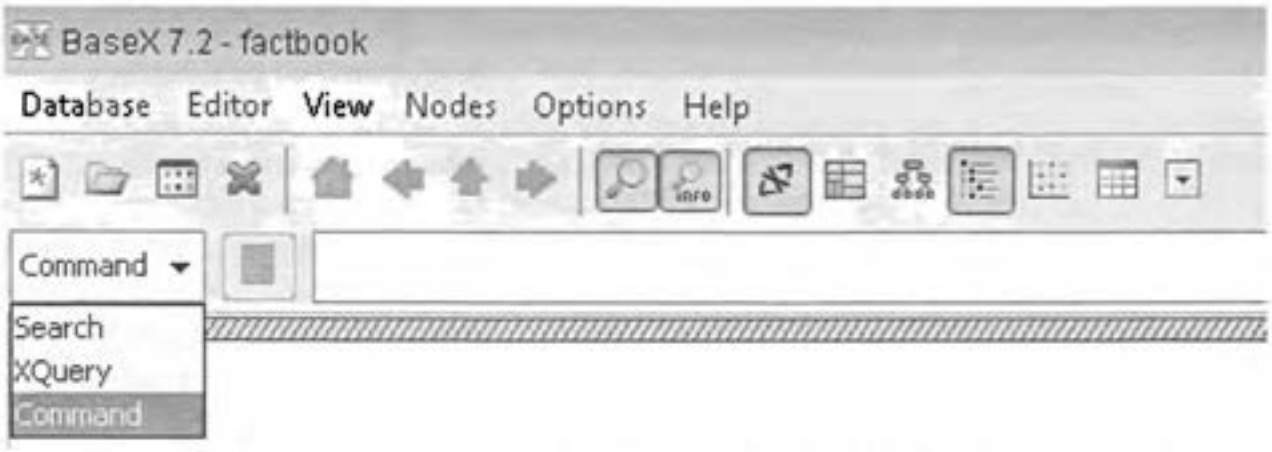


Figura 5.2: Tipos de sentencias en BaseX: Search (XPath), XQuery y del sistema gestor

Ejemplo:

Ejecución de una expresión XPath. Se muestra la expresión en la zona de consultas y luego se visualiza el resultado en todas las diferentes vistas abiertas en ese momento (texto plano, mapa y carpetas). Además, se muestra información sobre la consulta (tiempos de procesamiento, compilación, evaluación, impresión y total).

Figura 5.3: Resultado de una expresión XPath en BaseX

Se va a utilizar este programa, sencillo en su manejo, como herramienta para introducir dos lenguajes de consulta de documentos XML: XPath y XQuery.

Lo necesario para comenzar es saber cómo crear bases de datos XML nativas. Para ello, basta con seleccionar el menú *Database* y la opción *New*.

Aparecerá una ventana donde se solicita seleccionar un archivo XML o un directorio que contenga algún archivo XML. BaseX puede crear bases de datos asociadas a uno o a varios archivos XML.

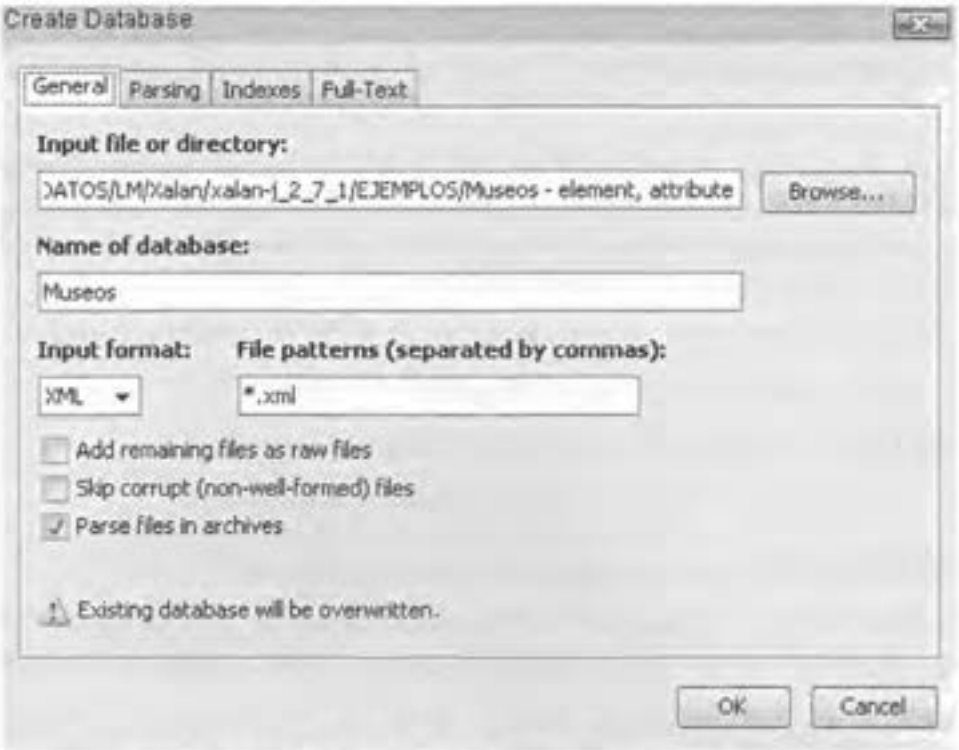


Figura 5.4: Creación de una base de datos con BaseX

Una vez seleccionado el archivo o directorio y asignado un nombre a la base de datos (Museos en la *Figura 5.4*), se accederá a la interfaz cuyas imágenes se han estado mostrando.

A partir de aquí, y según lo que se quiera ejecutar, órdenes del sistema gestor (Command), expresiones XPath (Search) o códigos XQuery (XQuery), se seleccionará una opción u otra del menú desplegable anteriormente citado.

5.3. XPath

XPath forma, junto con XSLT y XSL-FO (ambos se verán en el próximo capítulo), una familia de lenguajes llamada XSL, diseñados para acceder, transformar y dar formato de salida a documentos XML.

En el momento de la publicación del libro, la versión más reciente de XPath es la 2.0, que introduce algunas novedades respecto a la 1.0. Los ejemplos que aquí se verán son compatibles con la 1.0 y se indicará cuando se introduzca alguno específico de la 2.0.

Como ya se ha visto, un documento XML tiene una estructura de un árbol: un elemento raíz que tiene hijos, que a su vez tienen otros hijos, etc. A su vez, cada elemento puede tener atributos y/o contenido textual.

Más en concreto, se asemeja a un árbol genealógico, donde aparecerán conceptos como padre (se usará de manera genérica el género masculino, pero es intercambiable por madre), hijo, hermano, antepasado (o ascendiente), descendiente...

O también parecida a un sistema de ficheros: elementos que contienen otros elementos o datos, frente a directorios que contienen otros directorios o archivos. Para recorrer y listar la estructura de directorios usaríamos los comandos adecuados del sistema operativo: `cd`, `dir` o `ls`... Para recorrer un documento XML y extraer la información contenida en sus elementos se usará XPath.

XPath es un lenguaje de expresiones que no se usa de manera independiente, sino que se emplea en el contexto de un lenguaje anfitrión, como XSLT. Se trata de un lenguaje bastante sofisticado cuyo manejo en profundidad excede de los contenidos de este libro, por lo que aquí se presentarán su sintaxis y sus funcionalidades básicas.

Las especificaciones de XPath, desarrolladas por el W3C, se pueden consultar en:

- Lenguaje: http://www.w3.org/TR/xpath20
- Modelo de datos: http://www.w3.org/TR/xpath-datamodel

Direccionamiento

Las expresiones XPath, al igual que los direccionamientos en un sistema de ficheros, pueden ser absolutas (empiezan por /) o relativas (con respecto a nodo determinado, llamado **nodo de contexto**).

XPath trata un documento XML como un árbol de nodos (semejante a DOM).

Los tipos de nodos en XPath son:

- Raíz del documento (uno por documento): contiene todo el documento y se representa por el símbolo /. No tiene nodo padre y tiene al menos un nodo hijo, **el nodo documento**). Contiene también los comentarios e instrucciones de procesamiento, que no forman parte del documento en sí.
- Elemento: todo elemento de un documento XML es un nodo de XPath.
- Atributo: todo atributo de un documento XML es un nodo de XPath.
- Texto: todo contenido textual de un elemento es un nodo de XPath.
- Comentario.
- Instrucciones de procesamiento.
- Espacios de nombres.

Los tipos de datos básicos son: `string`, `number`, `boolean` y `node-set` (conjunto de nodos).

Las expresiones XPath más comunes son:

Expresión XPath	Coincidencia
`elemento`	Elemento de nombre `elemento`
`/elemento`	Elemento de nombre `elemento` ubicado en la raíz del documento
`e1/e2`	Elemento `e2` hijo directo del elemento `e1`
`e1//e2`	Elemento `e2` descendiente (hijo, nieto, bisnieto...) del elemento `e1`
`//elemento`	Elemento de nombre `elemento` ubicado en cualquier nivel por debajo de la raíz del documento
`@atributo`	Atributo de nombre `atributo`
`*`	Cualquier elemento (todos los elementos)
`@*`	Cualquier atributo (todos los atributos)
`.`	Nodo actual
`..`	Nodo padre
`espNom:*`	Todos los elementos en el espacio de nombres de prefijo `espNom`
`@espNom:*`	Todos los atributos en el espacio de nombres de prefijo `espNom`

Tabla 5.1: Expresiones básicas en XPath

Para probar las expresiones XPath, se podrá usar:

- Un entorno de desarrollo como el que ofrece BaseX, con ayudas al usuario sobre qué elementos descienden de cuáles. Se recomienda esta opción.
- Un analizador de expresiones XPath en línea, como por ejemplo:
  - http://www.whitebeam.org/library/guide/TechNotes/xpath.rhtm

Como ejemplo de trabajo se va a emplear un documento sobre Ciclos Formativos y Módulos Profesionales, llamado *formacionProfesional.xml*.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fP>
  <ciclos>
    <ciclo siglas="SMR">
      <nombre>Sistemas microinformáticos y redes</nombre>
      <familiaProfesional>Informática y comunicaciones
      </familiaProfesional>
      <duracion unidad="horas">2000</duracion>
      <grado>Medio</grado>
      <referenteInternacional clasificacion="CINE">3
      </referenteInternacional>
    </ciclo>
```

```
      <ciclo siglas="ASIR">
        <nombre>Administrador de sistemas informáticos en red</nombre>
        <familiaProfesional>Informática y comunicaciones
        </familiaProfesional>
        <duracion unidad="horas">2000</duracion>
        <grado>Superior</grado>
        <referenteInternacional clasificacion="CINE">5b
        </referenteInternacional>
        <ects>120</ects>
      </ciclo>
    ...
    </ciclos>

    <modulos>
      <modulo codigo="0222">
        <nombre>Sistemas operativos monopuesto</nombre>
        <duracion unidad="horas">80</duracion>
        <curso>1</curso>
        <ciclos>
          <ciclo siglas="SMR"/>
        </ciclos>
      </modulo>
      <modulo codigo="0373">
        <nombre>Lenguajes de marcas y sistemas de gestión de
        información</nombre>
        <duracion unidad="horas">70</duracion>
        <ects>7</ects>
        <curso>1</curso>
        <ciclos>
          <ciclo siglas="ASIR"/>
          <ciclo siglas="DAM"/>
          <ciclo siglas="DAW"/>
        </ciclos>
      </modulo>
      ...
    </modulos>
  </fP>
```

Acceso a elementos/atributos vs. acceso a su contenido textual

Cuando se escribe una ruta en XPath, por ejemplo a un cierto elemento `/persona/nombre`, lo que se devuelve es el conjunto de nodos llamados `nombre`, descendientes de `persona`, descendiente del nodo raíz. **Si no existiera esa ubicación, se devolvería el conjunto vacío.**

Hay analizadores de expresiones que devuelven directamente el contenido textual del nodo o nodos accedidos.

Ante la duda y para indicar al analizador que lo que se desea es el contenido textual, se utilizará la función **`text()`** a continuación de la expresión al nodo, de la forma:

```
/persona/nombre/text()
```

Para acceder a un atributo `dni` de `persona`, se escribirá `/persona/@dni`. Si se quiere acceder a su valor, se usará la función **`data()`** (en algunos analizadores no funciona), de la forma:

```
/persona/@dni/data()
```

En XPath 2.0 se usa la misma función **`string()`** para ambos casos, según sigue:

```
/persona/nombre/string()
/persona/@dni/string()
```

Acceso a elementos con rutas simples

Son rutas equivalentes a las que indicaríamos en un direccionamiento absoluto en un sistema Linux, es decir, desde la raíz hasta el elemento del cuál queremos extraer la información. La particularidad respecto a los sistemas de ficheros es que un nodo puede tener varios hijos que se llamen igual, mientras que en un sistema de ficheros no puede haber dos directorios hijos de un mismo directorio padre que se llamen igual.

Ejemplos:

- Todos los elementos hijos del nodo raíz, es decir, el elemento documento:

```
/*
```

- Todos los atributos de cualquier elemento del documento XML:

```
//@*
```

- Nombre de ciclos formativos:

```
/fP/ciclos/ciclo/titulo
```

- Duración de los módulos profesionales:

```
/fP/modulos/modulo/duracion
```

- Elementos que se llamen `nombre` ubicados en cualquier lugar del documento:

```
//nombre
```

- Siglas de los ciclos formativos:

```
/fP/ciclos/ciclo/@siglas
```

- Todos lo atributos de los módulos:

```
/fP/modulos/modulo/@*
```

Elementos del lenguaje

A partir de aquí hay que recurrir a los elementos de XPath, comunes a cualquier lenguaje de programación, que serán necesarios para generar expresiones más complejas. En particular, los operadores y las funciones.

- **Operadores**

Operadores booleanos

Operador	Codificación alternativa
and	
or	
not	
=	
!=	
>	>
<	<
>=	>=
<=	<=

Tabla 5.2: Operadores booleanos

Operadores aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
div	División
mod	Resto (módulo)

Tabla 5.3: Operadores aritméticos

Otros operadores

Operador	Significado
|	Unión de resultados

Tabla 5.4: Otros operadores

Ejemplo:

- Descendientes de `/fP`, de nombre `ciclo` o `modulo`, que contengan un descendiente directo `nombre`, cuyo valor se muestra.

```
/fP//(ciclo | modulo)/nombre
```

- **Funciones**

Funciones numéricas

Función	Devuelve	Ejemplo
round()	Redondeo	round(3.14) = 3
abs()	Valor absoluto	abs(-7) = 7
floor()	Suelo	floor(7.3) = 7
ceiling()	Techo	floor(7.3) = 8

Tabla 5.5: Funciones numéricas

Funciones de cadena

Función	Devuelve	Ejemplo
substring()	Subcadena	substring('Beatles', 1, 4) = 'Beat'
starts-with()	Cadena comienza por	starts-with('XML','X') = true
contains()	Cadena contiene	contains('XML','XM') = true
normalize-space()	Espacios normalizados	normalize-space(' The end ') = 'The end'
translate()	Cambia caracteres en una cadena	translate('The end','The','An') = 'An end'
string-length()	Longitud de una cadena	string-length('Beatles') = 7

Tabla 5.6: Funciones de cadena

Ejemplo:

- Nombre de los ciclos cuyas siglas empiezan por D:

```
/fP/modulos/modulo[starts-with(@siglas, 'D')]/nombre
```

Funciones que devuelven elementos por su posición

Función	Devuelve
`position() = n`	Elemento que se encuentra en la n-ésima posición
`elemento[n]`	Elemento que se encuentra en la n-ésima posición
`last()`	El último elemento de un conjunto
`last()-i`	El último elemento – i (ej. Si i=1 → el penúltimo)

Tabla 5.7: Funciones de elementos por su posición

Ejemplo:

- Todos los datos del segundo ciclo:

`/fP/ciclos/ciclo[position()=2]`

...o también

`/fP/ciclos/ciclo[2]`

- Todos los datos del último módulo:

`/fP/modulos/modulo[last()]`

Funciones que devuelven nodos

Función	Devuelve
`name()`	Nombre del nodo actual
`root()`	El elemento raíz
`node()`	Nodos descendientes del actual
`comment()`	Comentarios
`processing-instruction()`	Instrucciones de procesamiento

Tabla 5.8: Funciones que devuelven nodos

Ejemplo:

- Devuelve el nombre del nodo actual, es decir, `ciclo`.

`/fP/ciclos/ciclo/name()`

- Todos los comentarios existentes en el documento:

`//comment()`

- Todas las instrucciones de procesamiento existentes en el documento:

`//processing-instruction()`

- Todos los nodos del árbol cuyo nombre tenga una longitud de cuatro caracteres

`//*[string-length(name())=4]`

Funciones de agregado

Función	Devuelve
`count()`	Conteo de elementos
`avg()`	Media de valores
`max()`	Valor máximo
`min()`	Valor mínimo
`sum()`	Suma de valores

Tabla 5.9: Funciones que devuelven nodos

Ejemplo:

- Nombre de los módulos que se cursan en dos ciclos

```
/fp/modulos/modulo/ciclos[count(ciclo)=2]/../nombre
```

Acceso a elementos con filtros de valores literales

Se trata de una extensión de las expresiones anteriores mediante la cual se puede seleccionar elementos en función del valor de sus atributos o del propio elemento. En este caso los valores serán literales. Como ya se ha visto en algunos ejemplos previos, se indican las condiciones entre corchetes junto al elemento al cual se aplican. Los valores literales se indican entre comillas.

Ejemplos:

- Datos del módulo de código 0373:

```
/fP/modulos/modulo[@codigo="0373"]
```

o bien, si se quiere tratar el literal como un número.

```
/fP/modulos/modulo[@codigo=number("0373")]
```

- Nombre de los ciclos de grado medio:

```
/fP/ciclos/ciclo[@grado="Medio"]/nombre
```

- Duración de los módulos impartidos en el DAW:

```
/fP/modulos/modulo/ciclos/ciclo[@siglas="DAW"]/../../duracion
```

- Nombre de los ciclos que tengan un reconocimiento de ECTSs:

```
/fP/ciclos/ciclo/ects/../nombre
```

o bien `/fP/ciclos/ciclo[ects]/nombre`

Se pueden también combinar múltiples condiciones:

- Nombre de los ciclos cuyas siglas son ASIR o SMR:

```
/fP/ciclos/ciclo[@siglas="ASIR" or @siglas="SMR"]/nombre
```

- Nombre de los ciclos cuyas siglas se encuentren, en orden alfabético, entre ASIR y SMR

```
/fP/ciclos/ciclo[@siglas>"ASIR" and @siglas<"SMR"]/nombre
```

- Datos del segundo modulo de la lista de los cursados en primer curso:

```
/fP/modulos/modulo[curso="1"][2]
```

- Datos del tercer ciclo que tenga reconocimiento en ECTSs:

```
/fP/ciclos/ciclo[ects][3]
```

- Datos de los módulos en las posiciones 3 a 5 (sólo en XPath 2.0):

```
/fP/modulos/modulo[3 to 5]
```

- Nombre de los módulos cuya relación (cociente) entre duración y ects sea mayor de 10:

```
/fP/modulos/modulo[duración div ects &gt; 10]/nombre
```

Acceso a elementos con filtros de valores recuperados

Es el mismo caso que el anterior pero con valores recuperados mediante otras expresiones.

Ejemplos:

- Nombre de los ciclos en los que se cursen módulos de una duración de 80 horas o más:

Primero: En los elementos módulo se seleccionan aquellos que tengan una duración de 80 horas o más y se obtienen las siglas de los ciclos en los que se cursan:

```
/fP/modulos/modulo[duracion>=80]/ciclos/ciclo/@siglas
```

Segundo: Se muestran los nombres de los elementos `ciclo` cuyas siglas sean iguales a las extraídas en el paso primero:

```
/fP/ciclos/ciclo[@siglas=/fP/modulos/modulo[duracion>=80]/ciclos/ciclo/@siglas]
```

- Nombres de módulos que tengan una duración mayor que la de Sistemas Operativos. Se usa la función `number()` para obtener el valor numérico de la duración, de lo contrario haría la comparación como cadenas (donde se cumple que "100"< "20"):

```
/fP/modulos/modulo[number(duracion)>/fP/modulos/modulo[nombre="Sistemas Operativos"]/number(duracion)]/nombre
```

Acceso a elementos mediante ejes

En XPath, los ejes son expresiones que permiten acceder a trozos del árbol XML, apoyándose en las relaciones de parentesco entre los nodos.

Función	Uso
`self::*`	Devuelve el propio nodo de contexto. Equivalente a .
`child::`	Devuelve los nodos "hijo" del nodo de contexto.
`parent::*`	Devuelve el nodo padre del nodo contexto. Equivale a ..
`ancestor::*`	Devuelve los "antepasados" (padre, abuelo, hasta el nodo raíz) del nodo de contexto
`ancestor-or-self::*`	Devuelve los nodos "antepasados" del nodo de contexto además del propio nodo de contexto
`descendant::*`	Devuelve los nodos "descendientes" (hijo, nieto...) del nodo de contexto
`descendant-or-self::*`	Devuelve los nodos "descendientes" (hijo, nieto...) del nodo de contexto además del propio nodo de contexto Equivalente a //
`following::*`	Devuelve los nodos que aparezcan después del nodo de contexto en el documento, excluyendo a los nodos descendientes, los atributos y los nodos de espacio de nombres

`preceding::*`	Devuelve los nodos que aparezcan antes del nodo de contexto en el documento, excluyendo a los nodos ascendientes, los atributos y los nodos de espacio de nombres
`preceding-sibling::*`	Devuelve los "hermanos mayores" del nodo de contexto
`following-sibling::*`	Devuelve los "hermanos menores" del nodo de contexto
`attribute::*`	Atributos del nodo contexto. Equivale a @
`namespace::*`	Espacio de nombres del nodo de contexto

Tabla 5.10: Ejes en XPath

Ejemplos:

- Elementos hermanos menores del segundo módulo:

`/fP/modulos/modulo[2]/following-sibling::*`

- Descendientes del elemento raíz que se llamen nombre:

`/fP/descendant::nombre` o bien `/fP//nombre`

XPath 2.0

La novedad más llamativa es que dispone de los mismos tipos de datos predefinidos que los esquemas XML: `xs:string`, `xs:boolean`, `xs:date`...

Aparecen así mismo algunas funciones nuevas, como `lower-case()` o `upper-case()` para pasar cadenas de texto a minúsculas y mayúsculas respectivamente.

5.4. XQuery

XQuery en un lenguaje de consulta que permite extraer y procesar información almacenada en formato XML, habitualmente en bases de datos nativas XML o en tablas y campos de tipo XML en bases de datos relacionales.

Se parece al SQL (Standard Query Language – Lenguaje de Consultas Estándar) en algunas de las clausulas empleadas (`where`, `order by`) comunes a ambos lenguajes.

También **se asemeja a XPath, con el que comparte modelo de datos y soporta las mismas funciones y operadores**. Se podría considerar a XQuery como un superconjunto de XPath, ya que toda expresión XPath es una expresión XQuery válida.

Como sucede con otras tecnologías que en este libro se estudian de manera accesoria, a XQuery se le podría dedicar un libro entero, pero este no es el objetivo y aquí sólo se ofrecerá una visión introductoria.

Al igual que se ha hecho con XPath, para su estudio se utilizará el motor de bases de datos nativo BaseX. En este caso, con la base de datos de ejemplo basada en el documento *factbook.xml* (ubicado en la ruta etc/factbook.xml).

Se creará una base de datos que, en BaseX, se asocia a un documento XML, de nombre *mondial*, de manera que se puede hacer alusión al documento XML indistintamente como etc/factbook.xml o como *mondial*.

Lenguaje de expresiones

En XQuery todo es una expresión que se evalúa a un valor.

Por ejemplo, 7+3 es un código válido en XQuery que se evalúa a 10.

Otro ejemplo es una expresión condicional que devuelve un texto:

```
if (3 < 4) then "Verdadero" else "Falso"
```

Al evaluarse devolverá el texto Verdadero.

Tipos de datos

Los tipos de datos primitivos (predefinidos) o atómicos (no compuestos) son los mismos que los de los esquemas XML.

- Numéricos: enteros y reales.
- Booleanos.
- Cadenas de texto.
- Fechas, horas y duraciones.
- Tipos relacionados con XML, como QName.
- Nodos XML: nodo raíz, elemento, atributo, texto, comentario, instrucción de procesamiento y espacio de nombres.

Secuencias

Son listas de valores simples (atómicos).

Ejemplo:

Para entender las secuencias, se definirán varias variables que contienen secuencias y se utilizará la función `count()` para mostrar cuántos elementos tiene cada secuencia:

```
let $s1:= (2, 4, 6)  ❶
let $s2:= ($s1, $s1)  ❷
let $s3:= 10  ❸
let $s4:= ()  ❹
return (count($s1), count($s2), count($s3), count($s4))  ❺
```

❶ Se declara una variable `$s1` y se le asigna una secuencia que contiene 3 elementos.

❷ La segunda secuencia es la concatenación de la primera consigo misma, de manera que contiene 6 elementos.

❸ La tercera secuencia contiene un único elemento.

❹ La cuarta secuencia es vacía, no contiene elementos.

❺ Se devuelve el tamaño de cada una de las secuencias, que será: 3 6 1 0.

Ejemplo:

Con la ruta XPath `//city/name/text()` se obtienen una serie de nombres de ciudades (ubicadas en cualquier lugar del árbol XML), que se guardan en la variable `$city`, de ellas nos quedamos con las que empiecen por Q, se ordenan y se muestran.

```
for $city in doc('mondial')//city/name/text()
where starts-with($city, 'Q')
order by $city
return data($city)
```

Lectura de archivos

Con la función `doc()` se lee un documento XML que se indique como parámetro y devuelve el nodo raíz o los elementos que se indiquen mediante una expresión XPath.

Cláusulas FLWOR

FLWOR son las siglas de *For*, *Let*, *Where*, *Order by* y *Return* (en ingles se lee *flower*, flor), y son una serie de cláusulas que se usan para construir expresiones XQuery.

for: Indica qué elementos se van a seleccionar (habitualmente desde un documento XML de partida). Si va acompañada de la cláusula **at**, permite numerar los elementos que se van procesando.

Ejemplo:

Se genera la secuencia de números del 1 al 5 y se asigna iterativamente a la variable `$x`.

```
for $x in (1 to 5)
return <numero>{$x}</numero>
```

El resultado es:

```
<numero>1</numero>
<numero>2</numero>
<numero>3</numero>
<numero>4</numero>
<numero>5</numero>
```

Ejemplo con la cláusula `at`:

```
for $x at $i in doc("clasicos.xml")/clasicos/clasico/titulo
return <libro>{$i}. {data($x)}</libro>
```

El resultado es:

```
<libro>1. El señor de las moscas</libro>
<libro>2. El guardián entre el centeno</libro> ...
```

Se puede definir más de una expresión de entrada separada por comas:

Ejemplo:

Se declaran dos variables, `$alfa`, cuyos valores son 1 y 3, y `$beta` cuyos valores son 2 y 4. Se genera una salida cuyo elemento raíz es `<datos>` y que combina todos los valores de `$alfa` y `$beta` en pares de valores. Esta expresión se rodea del elemento `<datos>`, lo que permite que la respuesta aparezca dentro del elemento `<datos>`.

```
<datos>
{
  for $alfa in (1,3), $beta in (2,4)
  return <dato><alfa>{$alfa}</alfa><beta>{$beta}</beta></dato>
}
</datos>
```

La salida es:

```
<datos>
  <dato>
    <alfa>1</alfa>
    <beta>2</beta>
  </dato>
  <dato>
    <alfa>1</alfa>
    <beta>4</beta>
  </dato>

  <dato>
    <alfa>3</alfa>
    <beta>2</beta>
  </dato>
  <dato>
    <alfa>3</alfa>
    <beta>4</beta>
  </dato>
</datos>
```

let: Permite declarar variables a las que se les asignan valores.

Ejemplo:

Se declaran dos variables numéricas y se realiza una operación con ellas. Para este ejemplo no hubiera sido necesario declarar las variables, pero una vez declaradas se pueden usar en diferentes lugares.

Se puede declarar de dos maneras:

- con una cláusula `let` y tantas variables como se quiera declarar, separadas por comas.

```
let $x := 7, $y := 3 return 10*$x+$y
```

- con tantas clausulas `let` como variables se declaren.

```
let $x := 7 let $y := 3 return 10*$x+$y
```

Ambas expresiones se evaluarán a 73.

where: Permite introducir condiciones que deben de cumplir los elementos seleccionados por la cláusula for.

Ejemplo:

A partir del documento XML de ejemplo factbook.xml

```
for $continente in doc('mondial')/mondial/continent ❶
for $pais in doc('mondial')/mondial/country ❷
where $continente/@id = $pais/encompassed/@continent ❸
and $continente/@name="Europe" ❹
return data($pais/@name) ❺
```

❶ Se declara una variable `$continente` que contiene todos los elementos `/mondial/continent`.

❷ Se declara una variable `$pais` que contiene todos los elementos `/mondial/country`.

❸ Se vinculan datos de los elementos en `$continente`, de los que disponemos de su identificador almacenado en el atributo `id`, con los elementos en `$pais`, de los que disponemos del identificador del continente donde se ubican, almacenado en el atributo `continent` del elemento `encompassed`.

❹ Se hace sólo para los continentes cuyo nombre es Europa.

❺ Se devuelve el nombre de los países ubicados en Europa.

El resultado es: Albania Andorra Austria Belarus Belgium Bosnia and Herzegovina ...

order by: Permite ordenar los resultados de la consulta para su visualización

return: Devuelve los resultados

Ejemplo:

Transformar atributos en elementos con la función `data()`:

```
<continentes> ❶
{
```

```
    for $continentes in doc('factbook')/mondial/continent  ❷
    let $c:= $continentes/@name/data()  ❸
    order by $c  ❹
    return <continente>{$c}</continente>  ❺
  }
  </continentes>
```

❶ Se integra el resultado de la expresión dentro de un elemento `<continentes>`.

❷ Se declara una variable `$continentes` que contiene todos los elementos `/mondial/continent`.

❸ Se declara una variable `$c` que contiene el valor del atributo `name` de cada elemento `<continent>`.

❹ Se ordenan alfabéticamente los nombres de los continentes.

❺ De cada elemento `<continent>`, se devuelve el valor de su variable `name` rodeado de un elemento `<continente>` que se crea en este momento.

Otras cláusulas

declare function: Permite declarar funciones

if ... else: Permite declarar comportamientos condicionales

Funciones en XQuery

Como en casi todo lenguaje de programación, en XQuery existen funciones predefinidas. En muchos casos estas funciones coinciden con las existentes en XPath.

El URI del espacio de nombres de las funciones XQuery es el mismo que el de las funciones XPath. http://www.w3.org/2005/02/xpath-functions. El prefijo de estas funciones es `fn:`, aunque habitualmente se omite.

Entre las funciones predefinidas existentes, se pueden encontrar:

- De texto: uppercase, substring, contains, starts-with, replace, normalize-space ...
- Numéricas: max, abs, avg, sum, floor ...
- De fechas: current-date, current-time,, day-from-date, hours-from-time ...
- De nodos XML: root, data ...

Ejemplo:

Se devuelve el número de elementos `<members>` por elemento `<organization>`. La respuesta será una serie de números.

```
for $organizaciones in doc('factbook')/mondial/organization
return count($organizaciones/members)
```

Ejemplo:

Se devuelve la media de los valores obtenidos en el ejemplo anterior. Para ello se usa la función predefinida `avg` (en minúsculas), que se escribe rodeando con paréntesis la expresión del ejemplo anterior.

```
avg (
  for $organizaciones in doc('factbook')/mondial/organization
  return count($organizaciones/members)
)
```

Se pueden incluso combinar funciones, como en el caso siguiente en el que se redondea a dos decimales el resultado de la media anterior con la función `round-half-to-even()`:

```
round-half-to-even (
  avg (
    for $organizations in doc('factbook')/mondial/organization
    return count($organizations/members)
  ), 2)
```

También se pueden definir funciones por parte del usuario.

Ejemplo:

Se va a definir una función muy sencilla, aumento, que incrementa una cantidad (pasada por parámetro) en un porcentaje (pasado por parámetro). Se muestra la definición y el uso de la función. La función se declarará como parte del espacio de nombres local:

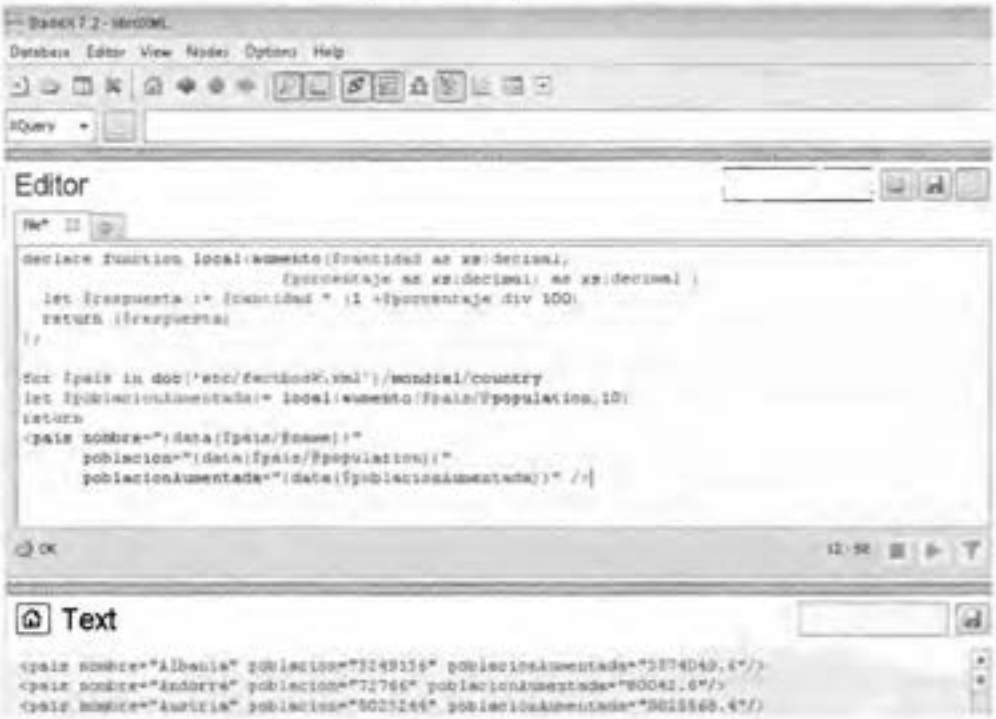


Figura 5.5: Resultado de un código XQuery en BaseX

Ejemplo:

```
for $agua in doc('etc/factbook.xml') /mondial/sea | /mondial/lake ❶
let $nombre:= normalize-space($agua/@name), $tipo:= $agua/name() ❷
order by $nombre ❸
return <agua tipo='{$tipo}'>{$nombre}</agua> ❹
```

❶ Se declara una variable $agua que contiene todos los elementos mar o lago (*sea* o *lake*) existentes en el documento factbook.xml, que sean descendientes de `/mondial`.

❷ Se declara una variable `$nombre` que contiene el valor del atributo `name`, perteneciente a cada `$agua` (sea mar o lago). También se declara una variable `$tipo` que contienen el nombre del elemento `$agua` (literalmente *sea* o *lake*).

❸ Esta lista de tuplas (`$nombre`, `$tipo`) se ordena por el valor de `$nombre`.

❹ Por último, se devuelve una lista de elementos `<agua>` con un atributo `tipo`, que tendrá el valor del `$tipo` correspondiente, y un contenido textual que tendrá el valor del `$nombre` correspondiente. Nótese que **para que se muestren los valores guardados en las variables hay que indicarlo rodeando las variables con corchetes**, si no se mostraría literalmente el nombre de la variable.

Figura 5.6: Resultado de un código XQuery en BaseX

5.5. Otras tecnologías complementarias: XLink y XPointer

Se citan aquí dos estándares del W3C relacionados con tecnologías que ya se han visto, sobre todo XML y XPath. No han tenido demasiada repercusión y su uso es muy restringido.

5.5.1. XLink

XLink (**X**ML **Link**ing Language – Lenguaje de Vinculación para XML) es un estándar del W3C para vincular documentos XML (http://www.w3.org/TR/xlink). Es una tecnología que no ha tenido un gran desarrollo e implantación.

Con XLink se pueden vincular documentos XML mediante **enlaces simples**, equivalentes a la etiqueta `<a>` de HTML que permiten vincular un documento con otro, y mediante **enlaces extendidos**, que permiten vincular varios documentos entre sí.

Ejemplo:

Se muestran varios enlaces simples.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<clasicos xmlns:xlink="http://www.w3.org/1999/xlink">  ❶
  <clasico titulo="El señor de las moscas">
    <sinopsis xlink:type="simple"  ❷
              xlink:href="http://www.clasicos/moscas.gif"  ❸
              xlink:show="new">  ❹
      Unos niños viajan en un avión que sufre un accidente...
    </sinopsis>
  </clasico>
  <clasico titulo="El guardián entre el centeno">
    <sinopsis xlink:type="simple"
              xlink:href="http://www.clasicos.com/guardian.gif"
              xlink:show="new">
      La visión muy particular de la vida que tiene un joven...
    </sinopsis>
  </clasico>
</clasicos>
```

❶ Se declara el espacio de nombres es `http://www.w3.org/1999/xlink`.

❷ Se indica que el tipo de vínculo es simple. Los vínculos simples son como los enlaces en HTML, permitiendo enlazar un documento sólo con otro recurso.

❸ Se indica el archivo al que se van a vincular con el atributo `xlink:href`.

❹ Se indica dónde se mostrará el recurso enlazado. En este caso en una nueva ventana.

Ejemplo:

Se vinculan cuatro documentos XML (cuya existencia se supone), uno que contiene información sobre el célebre pintor español Salvador Dalí (*dali.xml*), otro que contiene

información sobre la época en la que vivió durante la segunda república española (*segundaRepublica.xml*) y otros dos sobre personas con las que se relacionó, su pareja Gala (*gala.xml*) y su amigo, el poeta Federico García Lorca (*garciaLorca.xml*).

Se va a crear otro documento XML que servirá de vínculo entre los cuatro ya citados.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entorno xmlns:xlink="http://www.w3.org/1999/xlink"  ❶
         xlink:type="extended">  ❷
  <!-- se vincula al pintor con su época y sus personas allegadas -->
  <pintor xlink:type="locator" xlink:label="pintor"  ❸
          xlink:href="dali.xml"/>
  <allegado xlink:type="locator" xlink:label="pareja"  ❸
          xlink:href="gala.xml"/>
  <allegado xlink:type="locator" xlink:label="amigo"  ❸
          xlink:href="garciaLorca.xml"/>
  <historia xlink:type="locator" xlink:label="periodo"  ❸
          xlink:href="segundaRepublica.xml"/>
  <vinculo xlink:type="arc" xlink:from="pintor"  ❹
          xlink:to="pareja"/>
  <vinculo xlink:type="arc" xlink:from="pintor"  ❹
          xlink:to="amigo"/>
  <vinculo xlink:type="arc" xlink:from="pintor"  ❹
          xlink:to="periodo"/>
</entorno>
```

❶ Se declara el espacio de nombres es `http://www.w3.org/1999/xlink`.

❷ Se indica que el tipo de vínculo es extendido, lo que permite asociar múltiples documentos.

❸ Se indican los archivos que se van a vincular. Para ello se etiquetan los elementos implicados con el atributo `xlink:type="locator"`. Se categoriza cada fuente de información con el atributo `xlink:label` (pintor, pareja, amigo, etc) y se especifica el archivo donde se almacena la información con el atributo `xlink:href`.

❹ Por último, se crean los vínculos (arcos) entre las diferentes fuentes de información con elementos que contienen el atributo `xlink:type="arc"`. Se especificará también desde dónde, `xlink:from`, hasta dónde, `xlink:to`, va el arco.

Lista de atributos XLink:

Atributo	Descripción	Posibles valores
`xlink:type`	Tipo de enlace	*simple*, *extended*, *locator*, *arc*, *resource*, *title*, *none*
`xlink:href`	URI a la que se enlaza	
`xlink:role`	URI	
`xlink:arcrole`	URI	
`xlink:title`	Descripción del enlace	

`xlink:show`	Dónde se abre el enlace	*replace, new, embed, other, none*
`xlink:actuate`	Indica cuándo se muestra el recurso enlazado	*onLoad, onRequest, other, none*
`xlink:label`	Etiqueta de un elemento que se usará para vincular	
`xlink:from`	Valor del atributo `xlink:label` del elemento desde donde se vincula	
`xlink:to`	Valor del atributo `xlink:label` del elemento a donde se vincula	

5.5.2. XPointer

XPointer (**XML Pointer** Language – Lenguaje de Punteros XML) es una especificación del W3C para vincular fragmentos de documentos XML (http://www.w3.org/TR/WD-xptr). Al igual que el XLink, con el que conjuntamente se utiliza, es una tecnología que no ha tenido un gran desarrollo e implantación. Se apoya en XPath para identificar los fragmentos del documento XML que se enlazarán.

Ejemplo:

Una posible expresión XPointer sería:

```
libro.xml#xpointer(/libro/capitulo[@publico])
```

Indica que se enlaza con un documento llamado *libro.xml*. En él se buscarán los elementos `/libro/capitulo` que contengan un atributo llamado `publico`. Como cualquier expresión XPath, podría devolver un conjunto vacío de nodos.

Ejemplo:

```
libro.xml#xpointer(/libro/capitulo[@public])xpointer(/libro/capitulo[
@num="2"])
```

Indica que se enlaza con un documento llamado *libro.xml*. En él se buscarán los elementos `/libro/capitulo` que contengan un atributo llamado `publico`. Si devuelve un conjunto vacío de nodos, se evalúa la siguiente expresión, que devuelve los elementos `/libro/capitulo` cuyo atributo `num` valga 2.

Existen diversas funciones predefinidas que permiten acceder a diversos puntos en un documento XML: `here()`, `origin()`, `point()`, `start-point()`, `range()`... Su análisis se deja como tarea de profundización para el lector.

5.6. Bases de datos relacionales con XML

Igual que existen bases de datos XML nativas, existen sistemas gestores de bases de datos relacionales que permiten almacenar XML de alguna manera.

Algunos de estos sistemas gestores son:

- Microsoft SQL Server
- Oracle

Usaremos Oracle, en edición Express (XE), como herramienta para demostrar esto. Si bien los sistemas gestores de bases de datos de Oracle son aplicaciones con licencia propietaria, se pueden descargar gratuitamente del sitio de Oracle siempre que el fin de su uso no sea productivo.

La versión concreta a emplear es la 11g Release 2, que encontraremos en el sitio de Oracle (http://www.oracle.com/technetwork/products/express-edition/downloads/index.html), donde tendremos que crear una cuenta gratuita para poder descargarla.

Haciendo una instalación sin personalizar, a base de aceptar las opciones por defecto, podremos tener disponible el sistema gestor. Sólo se nos pedirá una única contraseña para los usuarios administradores por defecto, `SYS` y `SYSTEM`. Una vez instalado, utilizaremos el SQL*Plus, un cliente de línea de comandos, para ejecutar las sentencias de ejemplo. La conexión se realizará con el usuario `SYSTEM`. Aunque el emplear un usuario con privilegios de administración va contra la ortodoxia en el manejo de sistemas gestores de bases de datos, para un caso simplificado como éste nos valdrá.

En Oracle podremos disponer de tablas con campos de tipo XML, o bien de tablas en las que cada fila es, directamente, un documento XML. En ambos casos, se les puede asociar un esquema XML que valide el documento XML.

Campos de tipo XML

La instrucción de creación de una tabla Contratos con un campo de tipo XML (XMLTYPE) llamado contenido, será:

```
CREATE TABLE Contratos (
  idContrato NUMBER,
  fecha DATE,
  contenido XMLTYPE);
```

Actividad 5.1:

Se deja como trabajo de investigación para el lector la asociación de un esquema XML a un campo de tipo XML para realizar su validación. Se puede encontrar documentación al respecto en el sitio de Oracle, entre las páginas de sus extensísimos manuales, como por ejemplo en http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb05sto.htm.

Para insertar un registro que contenga datos relacionales, así como un valor para el campo XML contenido, ejecutaremos:

```
INSERT INTO Contratos VALUES (100, TO_DATE('01/01/1990'),
    XMLType('<?xml version="1.0"?>
                <contrato id="100">
                  <fecha>1990-01-01</fecha>
                  <lugar>Dublín</lugar>
                  ...
                </contrato>'));
```

Una consulta para mostrar el contenido del campo XML, apoyándose en la función getClobVal(), será:

```
SELECT C.contenido.getClobVal() contratoXML
FROM Contratos C;
```

Esta consulta devolvería el lugar del contrato extraído directamente del campo XML. Para ello, se utiliza la función **extract()** sobre el citado campo. Adicionalmente, se usa la función **getStringVal()** para tratar la información extraída como una cadena de texto. Si fuera de tipo numérico se usaría en su lugar **getNumVal()**.

```
SELECT C.contenido.extract('/contrato/lugar/text()').getStringVal()
FROM Contratos C;
```

Otra función equivalente a extract() es **extractValue()**, que permite extraer un dato de un campo XML sin tener que preocuparse de qué tipo es. La sintaxis cambia con respecto a extract(), ya que recibe dos parámetros, el campo de tipo XML y la expresión XPath que indicará qué información se extrae:

```
SELECT extractValue(C.contenido, '/contrato/fecha') Fecha
FROM Contratos C;
```

Se podría realizar otra consulta con la función **existsNode()**, que verifica si existe un cierto elemento en el campo XML. Devuelve 1 si existe y 0 si no.

Ejemplo:

Mostrar el valor del campo contenido (de tipo XML) de la tabla Contratos, para aquel contrato con identificador igual a 100 y que contenga un elemento lugar, descendiente de /contrato.

```
SELECT C.contenido.getClobval() contratoXML
FROM Contratos C
WHERE C.contenido.existsNode('/contrato[@id = "100"]/lugar') = 1;
```

También se pueden ejecutar consultas XQuery sobre campos de tipo XML. Para ello se utiliza la función **XMLQuery()**, que recibirá como parámetros la consulta XQuery, además del nombre del campo XML.

Ejemplo:

Mostrar el identificador del contrato (campo relacional) y el lugar del contrato (extraído del campo XML).

```
SELECT idcontrato, XMLQuery (
'for $c in /contrato
 order by $c/lugar
 return $c/lugar'
passing by value C.contenido
RETURNING CONTENT) DatosXMLRecuperados
FROM Contratos C;
```

Es interesante también la función **XMLTable()**, que mapea el resultado de una expresión XQuery, realizada en el FROM de la consulta SELECT, con una tabla relacional virtual (semejante a una vista) cuyos campos, de tipos de datos propios del sistema gestor con el que se trabaja, pueden ser referenciados.

Ejemplo:

A partir de una expresión XQuery sobre un campo XML, se construye una tabla virtual C2 con dos campos, Lugar (del tipo textual VARCHAR2) y Fecha (del tipo de fecha/hora DATE), que pueden ser referenciados.

```
SELECT C1.idContrato, C2."Lugar", C2."Fecha"
FROM Contratos C1,
   XMLTABLE('/contrato'
      PASSING C1.contenido
      COLUMNS
         "Lugar" varchar2(10) PATH '/contrato/lugar',
         "Fecha" DATE PATH '/contrato/fecha')
      C2;
```

Para modificar campos de tipo XML se procede de manera análoga a como se haría con un campo no XML. En este ejemplo se utiliza como condición de la clausula WHERE una referencia a un campo no XML:

```
UPDATE Contratos SET contenido = XMLType
       ('<?xml version="1.0"?>
          <contrato id="100">
            <fecha>1990-01-10</fecha>
            <lugar>Berlín</lugar>
            ...
          </contrato>')
WHERE idContrato = 100;
```

Para eliminar un registro, usando en esta ocasión una condición sobre el campo XML:

```
DELETE FROM Contratos C WHERE
C.contenido.extract('/contrato/lugar/text()').getStringVal()
    = 'Berlín';
```

Tablas de tipo XML

La creación de la tabla sería:

```
CREATE TABLE ContratosXML OF XMLTYPE;
```

La inserción de un registro sería:

```
INSERT INTO ContratosXML VALUES
    ('<?xml version="1.0"?>
          <contrato id="100">
            <fecha>1990-01-01</fecha>
            <lugar>Dublín</lugar>
            ...
          </contrato>'));
```

La consulta de selección del contenido de la tabla:

```
SELECT * FROM ContratosXML;
```

Una forma adicional de consultar la tabla XML es haciendo alusión a la meta-columna **object_value**, que muestra el contenido del documento XML almacenado en cada registro.

```
SELECT object_value FROM ContratosXML;
```

Para modificar el contenido de un documento almacenado en una tabla XML, se usará la función **updateXML()**, dentro de un proceso de modificación (UPDTE) de registros.

Ejemplo:

Se quiere modificar el valor del nodo `/contrato/lugar` de aquel contrato cuyo identificador valga 100, pasando a valer Múnich.

```
UPDATE ContratosXML C
  SET OBJECT_VALUE =
      updateXML(OBJECT_VALUE,
                '/contrato/lugar/text()', 'Múnich')
WHERE existsNode(OBJECT_VALUE, '/contrato[@id="100"]') = 1;
```

Con la función **deleteXML()** se pueden eliminar elementos de un documento XML. En sí mismo, se trata de un proceso de modificación (UPDATE) de un registro de la tabla XML.

Ejemplo:

Se quiere eliminar el elemento `/contrato/fecha` de aquel contrato cuyo lugar de celebración sea Múnich.

```
UPDATE ContratosXML C
  SET OBJECT_VALUE =
      deleteXML(OBJECT_VALUE,
                '/contrato/fecha')
  WHERE existsNode(OBJECT_VALUE,
                   '/contrato[lugar="Múnich"]')= 1;
```

Actividad 5.2:

Se deja como trabajo de investigación para el lector el uso de funciones como **insertChildXML()**, **insertXMLbefore()** o **appendChildXML()**, que permiten modificar la estructura del documento XML almacenado en un registro de una tabla XML, insertando nuevos elementos (nodos). Se recomienda el uso de la documentación de Oracle, que se puede encontrar en http://docs.oracle.com/cd/E11882_01/appdev.112/e23094/toc.htm.

5.7. Manejo de XML desde Java

Una forma alternativa de consultar y modificar documentos XML es desde lenguajes de programación.

Hoy en día, casi todos los lenguajes de alto nivel modernos (Java, C#, VisualBasic.NET...) incorporan mecanismos para poder acceder a documentos XML.

Los contenidos que vienen a continuación se consideran complementarios de lo hasta ahora visto y se basan en la posesión, por parte del alumno, de nociones de programación orientada a objeto, y en particular de Java, que es el lenguaje que se utilizará para ejemplificar los conceptos expuestos. Este lenguaje fue desarrollado por la compañía Sun, posteriormente integrada en Oracle.

Se ha utilizado para las pruebas el entorno de ejecución de Java, en su edición estándar 1.7.0_03 (http://www.oracle.com/technetwork/java/javase/downloads/index.html).

Se estudiarán dos APIs. Una, JAXP, integrada en la distribución de Java y que a su vez contiene a otras dos APIs más conocidas: SAX y DOM.

La otra, JAXB, es más moderna y se distribuye en paquetes independientes que deben ser incluidos en el proceso de compilación.

5.7.1. SAX

SAX (Simple API for XML – API Simple para XML), originally a Java-only API. SAX fue la primera API masivamente usada para manejar XML en Java. Es un estándar de facto.

Está basada en eventos y muy orientada a la lectura de documentos XML. Se desarrolló inicialmente para analizadores escritos en Java, aunque luego han aparecido implementaciones para otros lenguajes: C++, Python, Perl...

El modelo de recorrido de nodos basado en eventos difiere del basado en una estructura en árbol, ya que, según el analizador va leyendo el documento XML, va enviando notificación de ello al programa en tiempo real. No es necesario leer el documento entero para poder trabajar sobre los datos que se encuentran al principio el mismo. De hecho, el documento no reside en memoria (como veremos que ocurre en DOM), de manera que es la elección idónea para documentos grandes que no caben en la memoria disponible. Sin embargo, no sería adecuado si

se quisiera realizar alguna actualización en memoria de los datos para luego volcarlos en el documento.

SAX está compuesto de un conjunto de interfaces contenidas en el paquete `org.xml.sax`. Una de ellas es `XMLReader`, que representa un analizador XML.

Para analizar un documento XML con SAX, lo primero que hay que hacer es crear una instancia del `XMLReader` con la clase `XMLReaderFactory`, existente en el paquete `org.xml.sax.helpers`, mediante el método estático `createXMLReader()`.

Ejemplo:

Instanciación del analizador Xerces (parser de XML en Java), referenciado por su nombre:

```
try {
  XMLReader analizadorSAX = XMLReaderFactory.createXMLReader(
   "org.apache.xerces.parsers.SAXParser");
  AnalizadorSAX.parse("http://www.ejemplos.com/ejemplo1.xml")
// a partir de aquí, se analiza el documento ...
}
catch (SAXParseException e) {
  // el documento XML no está bien formado
}
catch (IOException e) {
  // no se puedo abrir el documento correctamente
}
catch (SAXException e) {
  // no se pudo crear el XMLReader
}
```

Después, hay que implementar la interfaz ContentHandler, que le dice al analizador qué acciones realizar cuando aparece una etiqueta de comienzo de elemento, un atributo, texto plano, etc.

Para ello hay que codificar los métodos de retrollamada (*callback*), que son:

- startDocument y endDocument: qué hacer cuando aparece un comienzo de documento y de fin de documento.
- startElement y endElement: qué hacer cuando aparece una etiqueta de apertura de un elemento y de cierre de elemento.
- characters: qué hacer cuando aparece texto.
- processingInstruction: qué hacer cuando aparece una instrucción de procesamiento.

5.7.2. DOM

DOM (**D**ocument **O**bject **M**odel – Modelo de Objetos de Documento) es una API para acceder y manipular documentos XML tratándolos como estructuras de árbol de nodos. Se define como un conjunto de recomendaciones que describen el modelo de objetos, independiente de

cualquier lenguaje de programación, usado para almacenar documentos jerárquicos en memoria (como XML, HTML o XSLT).

La jerarquía de objetos DOM almacena referencias entre los nodos de un documento, de manera que el documento completo debe ser leído y procesado antes de que pueda estar disponible para una aplicación DOM. Esto fuerza, además, a que el documento entero se encuentre en memoria, lo que supone, en ocasiones, una cantidad significativa de carga y convierte a esta técnica en inadecuada para procesar documentos de gran tamaño.

Para aplicaciones que requieren acceder aleatoriamente a diferentes posiciones del árbol y en diferentes momentos, o aplicaciones que requieren la modificación dinámica de la estructura de un documento XML, DOM es una tecnología muy adecuada.

Se utiliza la interfaz genérica `Node` como elemento principal. Representa un nodo del árbol, con tres enlaces definidos con su nodo padre, a sus nodos hijos y a sus nodos hermanos.

También existen interfaces específicas para los distintos tipos de nodo: elemento, atributo, instrucciones de procesamiento, etc. Todas ellas se derivan de `Node`.

Realmente la interfaz `Node` no se instancia directamente, pero al ser la interfaz raíz de todas las específicas, permite extraer información de cualquier objeto DOM sin saber su tipo.

Atributos principales:

- `nodeName`: cadena de texto que contiene el nombre del nodo.
- `nodeValue`: cadena de texto que contiene el valor del nodo.
- `parentNode`: objeto `Node` que contiene una referencia al nodo padre.
- `childNodes`: objeto `NodeList` que contiene una lista de nodos hijos.
- `firstChild`: objeto `Node` que contiene una referencia al primer hijo.
- `lastChild`: objeto `Node` que contiene una referencia al último hijo.
- `previousSibling`: objeto `Node` que contiene una referencia al hermano inmediatamente anterior.
- `nextSibling`: objeto `Node` que contiene una referencia al hermano posterior.
- `attributes`: lista de atributos con sus valores.
- `namespaceURI`: cadena de texto que contiene el URI del espacio de nombres del nodo.
- `prefix`: cadena de texto que contiene el prefijo del espacio de nombres de nodo.
- `localName`: cadena de texto que contiene el nombre local del nodo.

Métodos principales:

- `hasAttributes`: booleano que indica si el nodo tiene atributos o no.
- `insertBefore`: inserta un nodo antes del actual (como hermano precedente).
- `newChild`: crea un nodo hijo del actual.
- `removeChild`: elimina un nodo hijo del actual.
- `cloneNode`: crea un clon del nodo.

Interfaces específicas por tipo de nodo, todas derivadas de `Node`:

Tipo de nodo	Interfaz DOM	# Tipo Nodo
ATTRIBUTE_NODE	Attr	2
CDATA_SECTION_NODE	CDATASection	4
COMMENT_NODE	Comment	8
DOCUMENT_FRAGMENT_NODE	DocumentFragment	11
DOCUMENT_NODE	Document	9
DOCUMENT_TYPE_NODE	DocumentType	10
ELEMENT_NODE	Element	1
ENTITY_NODE	Entity	6
ENTITY_REFERENCE_NODE	EntityReference	5
NOTATION_NODE	Notation	12
PROCESSING_INSTRUCTION_NODE	ProcessingInstruction	7
TEXT_NODE	Text	3

Tabla 5.11: Interfaces DOM derivadas de `Node`

5.7.3 JAXP

JAXP (Java API for XML Processing – API de Java para Procesamiento de XML) es una API que permite a las aplicaciones Java procesar, transformar, validar y consultar documentos XML.

Viene incluida en JDK 1.6 como parte de los desarrollos del proyecto Xerces de Apache.

Contiene tres grupos de paquetes de clases:

- JAXP
  - Constantes
  - Tipos de datos
  - Espacios de nombres
  - Analizadores
  - Transformaciones
  - Transformaciones DOM
  - Transformaciones SAX
  - Transformación de flujos
  - XPath
- DOM
  - Interfaces DOM

  - Cargar y grabar
  - Eventos
  - HTML
  - Transversal
  - XPath
- SAX:
  - Interfaces SAX
  - Clases helper
  - Extensiones

Ejemplo:

Recorrer un documento XML con SAX.

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

class RecorrerSAX {
  public static void main(String[] args) {
    try {
      File x = new File(args[0]);  // archivo XML
      SAXParserFactory f = SAXParserFactory.newInstance();
      SAXParser p = f.newSAXParser();
      DefaultHandler h = new MyContentHandler();
      p.parse(x,h);
    } catch (ParserConfigurationException e) {
      System.out.println(e.toString());
    } catch (SAXException e) {
      System.out.println(e.toString());
    } catch (IOException e) {
      System.out.println(e.toString());
    }
  }

  /* Manejador de contenido */
  private static class MyContentHandler extends DefaultHandler {
    static String p = "_";


    public void startDocument() throws SAXException {
      System.out.println("Inicio del documento...");
    }
```

```
    public void endDocument() throws SAXException {
      System.out.println("Fin del documento...");
    }

    public void startElement(String ns, String sName, String qName,
      Attributes attrs) throws SAXException {
      String eName = sName;
      if (sName.equals("")) eName = qName;
        System.out.println("e"+p+eName);
      if (attrs!=null) {
        for (int i=0; i<attrs.getLength(); i++) {
          String aName = attrs.getLocalName(i);
          if (aName.equals("")) aName = attrs.getQName(i);
            System.out.println("a"+p+" "+aName+"="
              +attrs.getValue(i));
        }
      }
      p = p + "_";
    }

    public void endElement(String ns, String sName, String qName)
         throws SAXException {
      p = p.replaceFirst("__", "_");
    }

    public void characters(char buf[], int offset, int len)
         throws SAXException {
      String s = new String(buf, offset, len);
      System.out.println("c"+p+s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
         throws SAXException {
      String s = new String(buf, offset, len);
      System.out.println("i"+p+s);
    }
  }
}
```

Ejemplo:

Generar un árbol DOM a partir de un documento XML.

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;


class GenerarDOM {
  public static void main(String[] args) {
    try {
```

```
        File x = new File(args[0]);  // archivo XML
        DocumentBuilderFactory f
           = DocumentBuilderFactory.newInstance();
        DocumentBuilder b = f.newDocumentBuilder();
        Document d = b.parse(x);
        imprimeNodo(d, "");
      } catch (ParserConfigurationException e) {
        System.out.println(e.toString());
      } catch (SAXException e) {
        System.out.println(e.toString());
      } catch (IOException e) {
        System.out.println(e.toString());
      }
    }


    /* Función recursiva para imprimir los datos de los nodos */
    static void imprimeNodo(Node n, String p) {
      NodeList l = n.getChildNodes();
      NamedNodeMap m = n.getAttributes();
      int ml = -1;
      if (m!=null) ml = m.getLength();
      System.out.println(p+n.getNodeName()+": "+n.getNodeType()+", "
           +l.getLength()+", "+ml+", "+n.getNodeValue());
      for (int i=0; i<ml; i++) {
        Node c = m.item(i);
        imprimeNodo(c,p+" |-");
      }
      for (int i=0; i<l.getLength(); i++) {
        Node c = l.item(i);
        imprimeNodo(c,p+" ");
      }
    }
  }
```

El primer valor que se mostrará es en nodo raíz.

Los valores numéricos que aparecen acompañando al nombre de cada nodo son:

- Tipo de nodo
- N° de Hijos
- N° de atributos
- Valor del nodo

5.7.4 JAXB

JAXB (Java Architecture for XML Binding – Arquitectura Java para manejo de XML) es otra API, alternativa a JAXP, que pretende facilitar el manejo de documentos XML así como optimizar el rendimiento.

Los paquetes que incluyen las interfaces y clases que constituyen esta API no forman parte de la propia distribución de Java y deben descargarse de manera independiente (http://jaxb.java.net). La última versión en el momento de la publicación del libro es la 2.2.5.

De manera simplificada, las tareas que se desarrollarán con JAXB serán:

- Leer un documento XML a memoria (deserializar).
- Recorrer y/o manipular la estructura creada.
- Guardar en un archivo la estructura de memoria.

Los pasos para leer a un documento XML a memoria son:

- Vincular (*bind*) el esquema XSD:

  JAXB requiere que el documento XML que se quiera procesar tenga un esquema XSD asociado que lo valide.

  La vinculación del esquema consiste en la generación de una serie de clases en Java que representan el esquema y es realizada por una aplicación llamada **compilador de vinculación**, que viene con la distribución de JAXB. Se ejecuta independientemente desde la línea de comandos.

  La respuesta del compilador de vinculación son una serie archivos `.java`, que contienen interfaces y clases que implementan estas interfaces.

- Convertir el documento XML en objetos de Java, lo que se conoce como **deserialización** o ***unmarshal***.

  Se crea un **árbol de objetos de contenido** que representa al documento, pero no es un árbol DOM, sino que es una estructura más eficiente en el uso de la memoria.

  Los objetos de contenido son instancias de las clases producidas por el compilador de vinculación.

Si lo que se quisiera es escribir el contenido de un documento XML ya en memoria en un archivo, y suponiendo que la vinculación del esquema ya se ha realizado para llegar hasta aquí:

- Crear el árbol de contenido:

  Se puede hacer a partir de un documento XML existente por el proceso de *unmarshal*, o desde cero con la clase `ObjectFactory`, que permite crear los objetos del árbol de contenido.

- Volcar el árbol de contenido en un documento XML, lo que se conoce como **serialización** o ***marshal***:

  Este proceso es el opuesto al *unmarshal*.

Al usar esta API, además de los propios programas, se generan muchos archivos auxiliares, como las interfaces que representan los elementos del esquema XML y las clases que las implementan. Por ello no se mostrará el código de ninguna aplicación y se deja la tarea al alumno para profundización.

Ejercicios propuestos

1. Dado el documento XML *formacionProfesional.xml*, visto al principio del capítulo, construye las expresiones XPath para extraer la siguiente información:

- Referente internacional de los ciclos.
- Valor de los elementos `unidad`, se encuentren donde se encuentren en el árbol.
- Toda la información del penúltimo módulo.
- Nombre de los ciclos de grado superior.
- Nombre de los ciclos sin reconocimiento ECTS.
- Nombre del tercer módulo que sea de primer curso.
- Siglas de los ciclos cuyo referente internacional sea 5b.
- Códigos de los módulos que sean "hermanos pequeños" del módulo cuyo código es 0373
- Nombre de los ciclos donde se cursan módulos con 7 ECTSs.
- Nombre de los módulos que se cursen en más de un ciclo.

Se recomienda abrir el documento con BaseX para generar las expresiones.

2. A partir del documento *factbook.xml*, citado a lo largo del capítulo y que viene como ejemplo al instalar BaseX, genera las expresiones XPath para obtener la siguiente información.

   Puesto que las etiquetas de este fichero están en inglés, y para evitar dificultades de comprensión, se adjunta un pequeño listado de palabras clave con sus traducciones:

Término castellano	**Término inglés**	**Término castellano**	**Término inglés**
continente	continent	frontera	border
país	country	ubicado	located
ciudad	city	organización	organization
nombre	name	miembro	member
población	population	mar	sea
superficie	total_area	río	river
longitud	length	lago	lake
gobierno	government	desierto	desert
provincia	province	montaña	mountain
continente de un país	encompassed	isla	island

- *Nombre* de los *continentes.*
- *Nombre* del *río* que ocupa la décima posición.
- *Nombre* de la *isla* cuya superficie es *13935.*
- *Nombre* de los *países* que no tienen *frontera* con otros países (luego son islas).
- *Nombre* de *organizaciones* que tengan más de *5 miembros.*
- "Hermanos menores" de la *provincia* de *nombre Madrid*, descendiente del *país Spain.*
- *Nombre y gobierno* de los *países* cuya *población* sea menor que 1.000.000.
- *Nombre* de los *países* que tengan una *ciudad* llamada *Cordoba.*
- *Nombre* de los lagos *ubicados* en *Russia.*
- *Nombres* de los *países* cuyo territorio está en *Europe.*

3. Tomando como base el documento *factbook.xml*, citado a lo largo del capítulo y que aparece como ejemplo al instalar BaseX, teclea las siguientes expresiones XQuery y justifica su salida interpretando el resultado de cada línea de la expresión :

   a.

```
<credos>
{
  for $religionesUSA in doc('factbook')/mondial/country[@name="United
States"]/religions
  let $porcentajes:= $religionesUSA/@percentage
  order by $religionesUSA descending
  return <religión porcentaje="{$porcentajes}">
           {$religionesUSA/text()}
         </religión>
}
</credos>
```

   b.

```
for $islas in doc('factbook')/mondial/island[located/@country=
/mondial/country[@name="Spain"]/@id]
order by $islas/@name
return $islas/@name/data()
```

4. Con el mismo documento *factbook.xml*, genera las expresiones XQuery para obtener la siguiente información.

   a. Muestra los nombres de los países que tengan una población mayor que España (Spain), ordenados por el nombre del país. La salida será de la forma:

      `Bangladesh Brazil Burma China Egypt Ethiopia France...`

   b. Muestra el nombre de las montañas de más de 4000 metros de altura (condición en el **where**), así como la altura en forma de atributo, de la forma.

```
<cumbres>
  <montaña altura="6958">Aconcagua</montaña>
```

```
  <montaña altura="4506">Bjelucha</montaña>
  ...
<cumbres>
```

5. Estudia el siguiente código Java con comentarios, tratando de deducir su funcionamiento. A continuación, tecléalo y ejecútalo con algún documento XML de prueba para comprobar las suposiciones realizadas.

Clase auxiliar:

```
import org.xml.sax.*;

/* Clase auxiliar */
public class ContadorXML implements ContentHandler {

  private int numeroElementos;
  private int numeroAtributos;
  private int numeroInstruccionesProcesamiento;
  private int numeroCaracteres;

  public void startDocument( ) throws SAXException {
    numeroElementos = 0;
    numeroAtributos = 0;
    numeroInstruccionesProcesamiento = 0;
    numeroCaracteres = 0;
  }

  // Hay que contar o la etiqueta de apertura de un elemento
  // o la etiqueta de cierre, pero no ambas. Los elementos vacíos
  // se notifican de esta manera.
  public void startElement(String namespaceURI, String localName,
   String qualifiedName, Attributes atts) throws SAXException {
    numeroElementos++;
    numeroAtributos += atts.getLength( );
  }

  public void endElement(String namespaceURI, String localName,
   String qualifiedName) throws SAXException {}

  public void characters(char[] text, int start, int length)
   throws SAXException {
    numeroCaracteres += length;
  }

  public void ignorableWhitespace(char[] text, int start, int length)
   throws SAXException {
    numeroCaracteres += length;
  }

  public void processingInstruction(String target, String data)
   throws SAXException {
    numeroInstruccionesProcesamiento++;
  }
```

```
    // Una vez completamente procesado el documento,
    // se imprimen los resultados
    public void endDocument( ) throws SAXException {
      System.out.println("Num. elementos: " + numeroElementos);
      System.out.println("Num. atributos: " + numeroAtributos);
      System.out.println("Num. instrucciones procesamiento: "
       + numeroInstruccionesProcesamiento);
      System.out.println("Num. caracteres texto plano: "
       + numeroCaracteres);
    }

    // Métodos declarados en la interfaz ContentHandler que deben
    // implementarse pero que no hacen nada
    public void setDocumentLocator(Locator locator) {}
    public void startPrefixMapping(String prefix, String uri)
     throws SAXException {}
    public void endPrefixMapping(String prefix) throws SAXException {}
    public void skippedEntity(String name) throws SAXException {}
  }
```

La clase principal es:

```
  import org.xml.sax.*;
  import org.xml.sax.helpers.*;
  import java.io.IOException;

  /* Clase principal */
  public class EstadisticasDocumento {
    public static void main(String[] args) {
      XMLReader analizador;
      try {
       analizador = XMLReaderFactory.createXMLReader( );
      }
      catch (SAXException e) {
        // intento de uso del analizador Xerces referenciado
        // por su nombre
        try {
          analizador = XMLReaderFactory.createXMLReader(
           "org.apache.xerces.parsers.SAXParser");
        }


        catch (SAXException ee) {
          System.err.println("No se ha podido encontrar el analizador
SAX");
          return;
        }
      }

      if (args.length == 0) {
        System.out.println(
         "Uso: java EstadisticasDocumento <nombre_archivo.xml>");
      }

      // instalación del Content Handler
```

```
      // se referencia a la clase antes creada
      analizador.setContentHandler(new ContadorXML( ));

      // empezar el procesamiento
      for (int i = 0; i < args.length; i++) {

        // en la línea de comandos hay que indicar URIs
        // o nombres de archivos
        try {
          analizador.parse(args[i]);
        }
        catch (SAXParseException e) { // error de mal formación
          System.out.println(args[i] + " no es un documento bien
formado.");
          System.out.println(e.getMessage( )
           + " en la línea " + e.getLineNumber( )
           + ", columna " + e.getColumnNumber( ));
        }
        catch (SAXException e) { // otro tipo de error
          System.out.println(e.getMessage( ));
        }
        catch (IOException e) {
          System.out.println("No se ha podido informar sobre " +
args[i] + " debido a la IOException " + e);
        }
      }
    }
  }
```

6. El siguiente código Java será de utilidad en el capítulo siguiente en el que se estudien las transformaciones de documentos. Estúdiese ahora y úsese después.

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

class XSLTransformer {
  public static void main(String[] args) {
    try {
      File sf = new File(args[0]);  // archivo XML
      File rf = new File(args[1]);  // archivo de resultado
      File tf = new File(args[2]);  // hoja de transformaciones XSLT
      TransformerFactory f = TransformerFactory.newInstance();
      Transformer t = f.newTransformer(new StreamSource(tf));
      Source s = new StreamSource(sf);
      Result r = new StreamResult(rf);
      t.transform(s,r);
    } catch (TransformerConfigurationException e) {
      System.out.println(e.toString());
    } catch (TransformerException e) {
      System.out.println(e.toString());
    }
  }
}
```

CAPÍTULO 6

Transformación de documentos. XSLT

CONTENIDO

Introducción
XSLT
XSL-FO
Ejercicios propuestos

6.1. Introducción

XSL (eXtensible Stylesheet Language) es una familia de lenguajes, basados en XML, que consta de varios miembros:

- XSLT: un lenguaje para realizar transformaciones de documentos XML a otros formatos, incluido el propio XML.
- XPath: un lenguaje funcional (usado por XSLT) para acceder a secciones de un documento XML. Se estudió en el capítulo anterior.
- XSL-FO: un vocabulario XML para especificar semánticas de formateo.

En este capítulo se verá una tecnología fundamental, las **hojas de transformaciones XSLT**, que es un mecanismo para transformar documentos XML en otros documentos XML, HTML o texto.

Como complemento a las hojas de transformaciones, se utilizará el lenguaje de maquetado combinado con transformaciones, **XSL-FO**, que permite generar documentos en formatos de salida como PDF, PostScript, RTF... a partir de datos contenidos en documentos XML.

Este es un **capítulo prioritario** del libro. Los **contenidos más importantes** son las hojas de transformaciones XSL con sus instrucciones básicas, y el lenguaje de maquetado combinado con transformaciones, XSL-FO, con sus objetos de formateo básicos.

6.2. XSLT

XSLT (eXtensible Stylesheet Language for Transformations) es un estándar del W3C que aparece recomendado por primera vez por este organismo en 1999. La versión actual de XSLT en el momento de publicación de este libro es la 2.0.

XSLT está basado en XML, de manera que cada hoja de transformaciones es un documento XML bien formado.

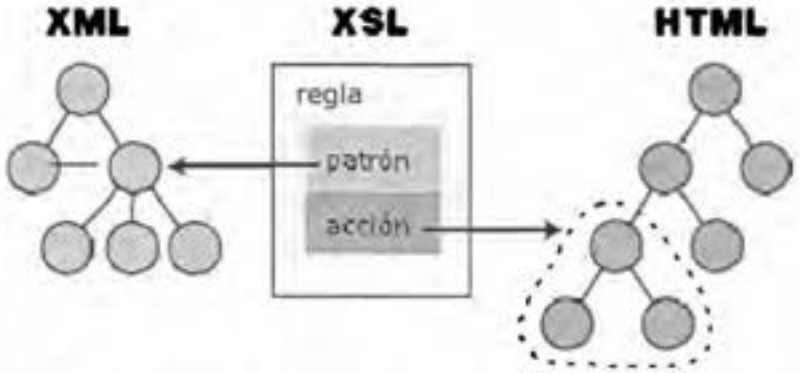


Figura 6.1: Transformación de XML en HTML mediante la aplicación de reglas

Se trata de un lenguaje de programación declarativo que permite generar documentos, en diferentes formatos de salida (XML, HTML, texto, PDF, RTF, etc), a partir de un documento

XML. Se describe como declarativo porque consiste en la declaración de una serie de reglas o plantillas que hay que aplicar a un documento XML para transformarlo en la salida deseada. Una regla asocia un patrón (expresión) con elementos del documento XML de partida y les aplica una serie de acciones.

Estas reglas se almacenarán en un documento de texto, que habitualmente tiene la extensión `.xsl` que, junto con el documento `.xml` de partida serán pasados como parámetros a un procesador XSLT, que generará como salida el nuevo documento transformado.

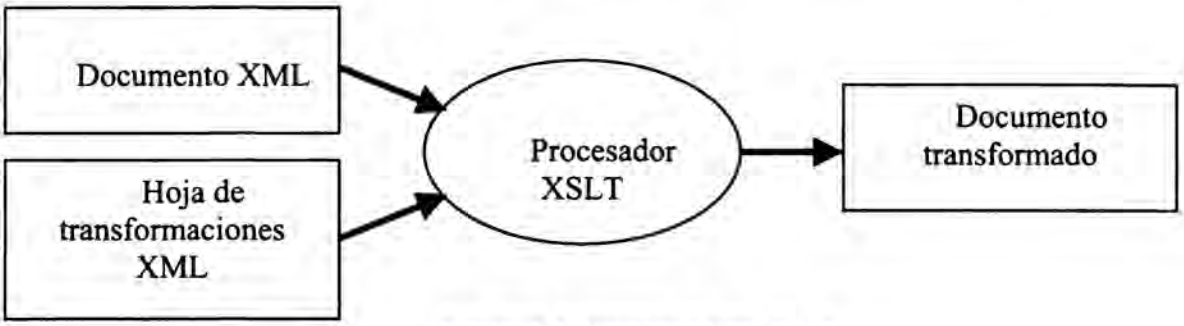


Figura 6.2: Procesamiento XSLT

Hojas de estilos: CSS vs. XSLT

El W3C ha definido dos familias de normas para hojas de estilos. La más antigua y simple es CSS, que permite definir propiedades de elementos de marcado (una letra en negrita o de un cierto tamaño, un recuadro con bordes verdes discontinuos...). Sin embargo, hay ciertas tareas que no se pueden acometer con CSS:

- No se puede cambiar el orden en el que los elementos de un documento HTML se visualizan (no puede ordenar elementos o filtrarlos según algún criterio).
- No se pueden realizar operaciones, como sumar los valores de todos los elementos `<precio>` de un documento.
- No se pueden combinar múltiples elementos, como todas las nóminas anuales de un empleado con el fin de obtener un total de ingresos y retenciones.

Estas limitaciones que aparecen en el uso de CSS se superan con XSLT. De hecho, en muchas ocasiones se usará CSS y XSLT de manera complementaria. Con XSLT se determinará qué contenido de un documento XML se quiere mostrar y en qué orden, realizándose si es necesario algún cálculo sencillo; con CSS se dará un formato visual agradable a la información mostrada.

Procesador XSLT

Son aplicaciones capaces de procesar documentos XML mediante hojas de transformaciones XSLT.

Cuando el procesador XSLT lee un documento XML genera una representación de dicho documento como un **árbol de nodos**. Los tipos de nodo son: elemento, atributo, texto, comentario, instrucción de procesamiento y espacio de nombres. Como ya se ha comentado

previamente, las relaciones entre los nodos son las mismas que en un árbol genealógico: padres, hijos, ascendientes, descendientes...

Los comentarios y las instrucciones de procesamiento son parte del árbol de nodos y deben ser tenidos en cuenta a la hora de contar nodos o iterar sobre ellos.

El procesador XSLT va recorriendo y procesando nodo a nodo. El nodo que se está tratando en un momento dado se denomina **nodo de contexto**.

Existen algunas de estos procesadores en línea:

- http://xslt.online-toolz.com/tools/xslt-transformation.php
- http://www.shell-tools.net/index.php?op=xslt
- http://markbucayan.appspot.com/xslt/index.html

Otros procesadores son instalables, como **Xalan** (http://xml.apache.org/xalan-j/), desarrollado en Java y ejecutable desde la línea de comandos. En el momento de la publicación de este libro, la última versión publicada es la 2.7.1. Los ejemplos que se exponen se procesarán con Xalan.

Así mismo, aplicaciones que se han introducido ya como editores de XML, de DTDs, de esquemas XML, validadores de XML... también permiten realizar transformaciones. Algunas de las más destacadas son **Altova StyleVision**, **<oXygen/>** o **XML Copy Editor**.

Proceso de transformación

En este tema se realizarán transformaciones desde XML a texto, XML o HTML, siendo estas últimas las más comunes. Para realizar la transformación se precisan dos documentos: el XML de partida y la hoja de estilos XSLT (como ya se ha indicado, formalmente **una hoja de estilos XSLT también es un documento XML**).

Una transformación XSL se puede llevar a cabo en tres ubicaciones distintas:

- **En el servidor web**: Mediante algún lenguaje de servidor (PHP, asp.NET, JSP, etc) que aplique la XSLT al XML y envíe de vuelta el HTML producido al cliente. Esta técnica está fuera de los contenidos de este módulo. Existe un proyecto llamado **Cocoon**, dentro de Apache, consistente en clases Java que realizan esta tarea.
- **En el cliente web**: El cliente web (Mozilla Firefox o Internet Explorer) realiza las transformaciones. Se puede hacer con **JavaScript**, aunque existen librerías como **Sarissa**, que incluye funcionalidades como tratamiento de documentos XML, transformaciones XSLT, procesamiento de expresiones XPath...

  Ejemplo:

  Se transforma mediante JavaScript un documento XML, llamado *datos.xml*, con una hoja de transformaciones, llamada *transforma.xsl*.

```
<html>
<head>
<script>

function cargaDocumentoXML(documento){
  if (window.XMLHttpRequest) {
```

```
      xhttp = new XMLHttpRequest();
    } else {
      Xhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    xhttp.open("GET", documento, false);
    xhttp.send("");
    return xhttp.responseXML;
  }

  function muestraTransformacion(){
    xml = cargaDocumentoXML("datos.xml");
    xsl = cargaDocumentoXML("transforma.xsl");
    // código para IE
    if (window.ActiveXObject) {
      ex = xml.transformNode(xsl);
      document.getElementById("ejemplo").innerHTML = ex;
    }
    // código para Mozilla Firefox, Opera, etc.
    else if (document.implementation &&
          document.implementation.createDocument) {
      xsltProcessor = new XSLTProcessor();
      xsltProcessor.importStylesheet(xsl);
      resultado = xsltProcessor.transformToFragment(xml, document);
      document.getElementById("ejemplo").appendChild(resultado);
    }
  }
  </script>
  </head>

  <body onload="muestraTransformacion()">
    <div id="ejemplo" />
  </body>
  </html>
```

- Mediante una aplicación diferente: se pueden usar programas de funcionamiento independiente para realizar estas transformaciones.

Pasos para la transformación

1. La hoja de transformaciones es analizada y se convierte en una estructura de árbol.
2. El documento XML es procesado y convertido en una estructura de árbol.
3. El procesador XSLT se posiciona en la raíz del documento XML. Este es el contexto original.
4. Los elementos (como las etiquetas HTML) que no formen parte del espacio de nombres de prefijo `xsl`, son pasados a la cadena de salida sin modificarse. Se denominan **elementos de resultado literal**.
5. El procesador XSLT sólo aplica una regla a cada nodo. Si tenemos dos reglas para el mismo nodo, el procesador sólo aplica una de ellas: la última en aparecer.

Salvo en el caso anteriormente citado, el orden en el que aparecen las plantillas en la hoja de transformaciones no es representativo. Lo que marca el orden es el recorrido, por parte del procesador, del árbol del documento XML.

Estructura básica de una hoja de transformaciones

Una hoja de transformaciones estará compuesta por:

1.- Una declaración de documento XML, puesto que lo es:

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

2.- Un elemento raíz, llamado `<xsl:stylesheet>` (sinónimo a `<xsl:transform>` que es igualmente válido, pero en los ejemplos no se usará).

```
<xsl:stylesheet version="1.0"
            xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
  ...
</xsl:stylesheet>
```

3.- El espacio de nombres es `http://www.w3.org/1999/XSL/Transform`, siendo `xsl` el prefijo.

4.- Existen otros **elementos**, llamados **de nivel superior** que, de aparecer en la hoja de transformaciones, siempre lo harán como hijos de `<xsl:stylesheet>`.

Son `<xsl:import>` (si aparece, debe ser el primer hijo del elemento raíz), `<xsl:include>`, `<xsl:namespace-alias>`, `<xsl:output>`, `<xsl:strip-space>`, `<xsl:preserve-space>`, `<xsl:attribute-set>`, `<xsl:key>`, `<xsl:param>` (puede ser parámetro global o local), `<xsl:variable>` (puede ser variable global o local) y `<xsl:template>`. Algunos se verán muy en detalle y otros levemente.

Algunas transformaciones especiales

Hay algunas transformaciones que tienen un comportamiento especial:

1.- Si la hoja de transformaciones no contiene plantillas (*template*) el contenido textual del documento XML se envía directamente a la salida, omitiéndose los valores de los atributos. Tampoco se envían comentarios ni instrucciones de procesamiento.

Ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

2.- Si la hoja de transformaciones sólo tiene una plantilla, asociada al nodo raíz del documento XML, pero sin ningún contenido, no se envía nada a la salida.

Ejemplo:

```
<?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="/">
  </xsl:template>
</xsl:stylesheet>
```

3.- Si la hoja de transformaciones no tiene ninguna plantilla asociada al nodo raíz, pero sí otras asociadas a otros elementos, se va recorriendo el árbol del documento XML y enviando el contenido textual a la salida y, cuando se encuentra un elemento con una plantilla asociada, se aplican las reglas definidas en la plantilla.

4.- En general, si la hoja de transformaciones con una alguna plantilla, se va recorriendo el árbol del documento XML en el orden en el que está escrito y, en el momento en que aparece alguna plantilla cuya expresión (atributo `match`) coincide con el elemento en el que nos encontramos, se le aplica la plantilla.

Se aplican las transformaciones que indique la plantilla. Si hubiera una plantilla vacía (sin acciones en su interior, el elemento al que afectara la plantilla y sus descendientes no se visualizarían.

Ejemplo:

Si la plantilla que se ajusta al elemento raíz no contiene instrucción alguna, la salida que se generará es vacía. Ya no se seguirá procesando la hoja de transformaciones aunque tuviera más plantillas.

```
<?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="/">
  </xsl:template>

  <!-- Esta plantilla nunca se aplicará -->
  <xsl:template match="/contratoAlquiler/arrendador">
    ...
  </xsl:template>
</xsl:stylesheet>
```

Cada regla (plantilla o template) en la hoja de transformaciones contiene un patrón XPath asociado, que coincide con los nodos del documento XML original a los cuales se aplicará la regla. Si hay varias plantillas que se ajustan a la misma expresión, se aplican las reglas de resolución de plantillas que se verán más adelante.

6.2.1. Enlace de documentos XML con hojas de estilo

Se puede asociar de forma permanente una hoja de estilo, sea CSS o XSLT a un documento XML mediante la instrucción de procesamiento `<?xml-stylesheet ?>`. Esta instrucción se ubica al principio del documento XML, después de la declaración `<?xml ... ?>`.

Cuando se visualiza en un navegador web un documento XML enlazado con una hoja de estilo CSS, se muestra el resultado de la aplicación de los estilos a los elementos existentes.

Ejemplo:

A un documento XML de componentes de un ordenador se le va a aplicar una hoja de estilos CSS, generando una salida en un navegador.

El código del documento componentes.xml, en cuya segunda línea se enlaza con la hoja de estilos componentes.css, es:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="componentes.css"?>

<componentes>
   <titulo>Componentes de un ordenador</titulo>
   <componente>
      <item>Palca base</item>
      <fabricante>ASUS</fabricante>
      <modelo>P3B-F</modelo>
      <precio>123.00</precio>
   </componente>
   <componente>
      <item>Video Card</item>
      <fabricante>ATI</fabricante>
      <modelo>All-in-Wonder Pro</modelo>
      <precio>160.00</precio>
   </componente>
   <componente>
      <item>Sound Card</item>
      <fabricante>Creative Labs</fabricante>
      <modelo>Sound Blaster Live</modelo>
      <precio>80.00</precio>
   </componente>
   <componente>
      <item>19 inch Monitor</item>
      <fabricante>LG Electronics</fabricante>
      <modelo>995E</modelo>
      <precio>290.00</precio>
   </componente>
</componentes>
```

El código de la hoja de estilos, componentes.css, es:

```
componentes {
  display: block;
}
```

```
titulo {
  display: block;
  font-family: arial;
  color: #008000;
  font-weight: 600;
  font-size: 22;
  margin-top: 12pt;
  text-align: center;
}

componente {
  display: block;
  }

item {
  display: block;
  font-family: arial;
  color: #000080;
  font-weight: 400;
  margin-left: 15pt;
  margin-top: 12pt;
  font-size: 18;
}

fabricante {
  display: block;
  font-family: arial;
  color: #600060;
  font-weight: 400;
  margin-left: 45pt;
  margin-top: 5pt;
  font-size: 18;
}

modelo {
  display: block;
  font-family: arial;
  color: #006000;
  font-weight: 400;
  margin-left: 45pt;
  margin-top: 5pt;
  font-size: 18;
}

precio {
  display: block;
  font-family: arial;
  color: #800000;
  font-weight: 400;
  margin-left: 45pt;
  margin-top: 5pt;
  font-size: 18;
}
```

La salida en un navegador, resultado de la aplicación de la hoja de estilos CSS al documento XML, es la siguiente:

Figura 6.3: Formateado de un documento XML con una hoja de estilos CSS

Cuando se visualiza en un navegador web un documento XML enlazado con una hoja de estilo XSLT, el navegadores muestran el resultado de la transformación. Si se utiliza la funcionalidad *Ver código fuente de la página*, el navegador mostrará el documento XML original.

6.2.2. Elementos básicos de una hoja de transformaciones

Para generar la mayor parte de las salidas que se pueda querer, bastará utilizar un pequeño conjunto de instrucciones de transformación.

xsl:stylesheet y xsl:transform

Son elementos equivalentes que se usan para definir el elemento raíz de la hoja de transformaciones. En los ejemplos del libro se empleará siempre `<xsl:stylesheet>`.

Atributos obligatorios:

- `version`: especifica la versión de XSLT del documento.
- `xmlns`: espacio de nombres del documento XML. Por defecto se usará `xsl`.

Atributos optativos principales:

- `exclude-result-prefixes`: lista de espacios de nombres (prefijos), separados por espacios, que no deben ser enviados a la salida.

Ejemplo:

Si no hay ninguna plantilla definida, el procesador manda a la salida el texto del documento XML de entrada. **Los atributos no se mostrarán**.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

xsl:output

Define el formato del documento de salida.

Es un elemento de nivel superior, y debe aparecer como hijo de `<xsl:stylesheet>` o `<xsl:transform>`.

Atributos optativos principales:

- `method`: define el formato de salida. Por defecto es XML (pero si el primer hijo del nodo raíz es `<html>` y no hay nodos de texto previos, entonces el formato de salida es HTML).

  Posibles valores: ***xml***, *html*, *text*.

- `version`: define la versión del formato de salida (sólo para formatos de salida `XML` y `HTML`).

- `encoding`: indica el juego de caracteres de la salida. Por defecto es *UTF-8*.

- `indent`: indica si la salida debe ser indentada de acuerdo a su estructura jerárquica.

- `omit-xml-declaration`: indica si la instrucción de procesamiento que declara el documento como XML se omitirá en la salida. Por defecto vale *no*, lo que significa que sí se enviará a la salida el `<?xml version="1.0"?>`.

  Posibles valores: ***no***, *yes*.

- `standalone`: indica si a la instrucción de procesamiento de declaración de tipo de documento como XML, `<?xml version="1.0"?>`, se le añade el atributo `standalone`. Por defecto vale *no*.

  Posibles valores: ***no***, *yes*.

Ejemplo:

Se indicará que la salida de una transformación estará en formato HTML. El `encoding` será *iso-8859-1* y se omitirá el envío a la salida de la declaración de documento XML.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"
          omit-xml-declaration="yes"/>
</xsl:stylesheet>
```

xsl:template

Es, junto con `<xsl:apply-templates>`, la instrucción principal de las hojas de transformaciones. Representa una plantilla con una serie de acciones que se realizarán si el patrón de la plantilla (atributo `match`) encaja con algún elemento o elementos del árbol XML. El

atributo `match` tiene el valor de una expresión XPath e indica qué elementos del árbol XML se verán afectados por la plantilla. A continuación se define la transformación a aplicar y, si no se indicara nada, se sustituiría el nodo por nada.

Cuando se aplica una plantilla a un nodo, se aplica únicamente al nodo seleccionado, pero se sustituye el nodo y todos sus descendientes por el resultado de la aplicación de la plantilla, lo que nos haría perder la información de los descendientes.

Si se quiere que antes de sustituir el nodo y todos sus descendientes se apliquen también a los descendientes otras plantillas que queramos, tendremos que utilizar la instrucción `<xsl:apply-templates>`.

Una plantilla es una regla, cuyas acciones se ejecutarán si el patrón que tiene asociado la plantilla encaja con algún elemento del árbol XML. Cuando una plantilla se aplica, se dice que ha sido instanciada.

Atributos optativos principales:

- `name`: indica un nombre para la plantilla. Si se omite, entonces el atributo `match` es obligatorio. No puede haber en una hoja de transformaciones dos plantillas con el mismo nombre, sino se producirá un error.
- `match`: indica un patrón XPath al que se le aplicará la plantilla. Si se omite, entonces el atributo `name` es obligatorio.
- `priority`: es un valor real desde -9 a 9, de menos a más prioritario. Se utiliza para aplicar las reglas de resolución de conflictos entre plantillas.
- `mode`: permite distinguir dos plantillas cuyo atributo `match` sea el mismo, de manera que en la invocación desde el `<xsl:apply-templates>` se pueda indicar explícitamente cuál usar.

Ejemplo:

Cuando se encuentre el elemento raíz del árbol se generará un documento HTML estático, idéntico para cualquier documento XML de entrada.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" />
<xsl:template match="/">
  <html>
  <body>
    <h1>Documento HTML estático</h1>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

xsl:apply-templates

Es, junto con `<xsl:templates>`, la instrucción principal del lenguaje de transformaciones XSLT. La correcta comprensión de su funcionamiento nos permitirá sacar el máximo partido de las transformaciones que deseemos realizar. Por este motivo, se presentarán diversos ejemplos.

Permite procesar el resto de nodos hijos del actual, es decir, seguir aplicando plantillas.

Atributos optativos principales:

- `select`: es una expresión XPath que especifica los nodos a seleccionar para ser procesados. Si se omite, todos los nodos descendientes del nodo actual serán seleccionados. Si se indica una expresión que no coincide con ningún conjunto de nodos, no se aplicará ninguna transformación.
- `mode`: permite distinguir entre varias plantillas con igual valor en su atributo `match`, por lo tanto sólo tiene sentido su existencia si existe el atributo `select`. Las plantillas que se aplicarían también deberán venir etiquetados con el mismo atributo `mode`.

Ejemplo:

En este caso se mostrará todo el contenido textual de los distintos elementos, descendientes de /, del documento XML de partida, formateado como título `<h1>`. En la plantilla principal, que se vincula con la raíz del documento XML de partida, se manda a la salida la estructura principal de una página HTML y se invoca la aplicación de plantillas **sin atributos**. Esto producirá el efecto ya comentado: se enviará a la salida todo el contenido textual de los elementos descendientes de /. Si hubiera más plantillas, se aplicarían cuando correspondiera.

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

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" />

<xsl:template match="/">
  <html>
  <body>
    <h1><xsl:apply-templates /></h1>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Ejemplo:

Dado el ejemplo del contrato de alquiler, se definen dos plantillas, la principal que se aplicará desde el elemento raíz, y otra que se aplicará al encontrar el elemento `arrendatario`. Por otro lado, en la plantilla principal se indica que se apliquen las plantillas que se encuentren a partir de la ubicación `contratoAlquiler`. Esto producirá como salida que se muestre el contenido textual de todos los descendientes de `contratoAlquiler`, menos del elemento `arrendatario`, para el cual se aplicará la plantilla definida.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xslt="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:output method="text" encoding="iso-8859-1" />
```

```
    <xsl:template match="/">
      <xsl:apply-templates select="contratoAlquiler"/>
    </xsl:template>

    <xsl:template match="arrendatario">
      Plantilla n° 1
      Ruta: arrendatario
    </xsl:template>

  </xsl:stylesheet>
```

La salida será:

```
  <?xml version="1.0" encoding="iso-8859-1"?>
    2012-11-10
    Sidney
    1000
    5

      Calle
      Cruz del sur
      73
      Izquierda
      10
      C

      John
      Smith
      Sydney
      Australia

      Plantilla n°: 3
      Ruta: arrendatario

        Que el ARRENDADOR es el único propietario del INMUEBLE
      Que el ARRENDADOR es el único propietario de los bienes muebles
  descritos en el Anexo I

      Causas de término del arrendamiento
        Acuerdo mutuo
        Fallecimiento del arrendador o el arrendatario si así se
hubiera pactado
        Fin del plazo estipulado o aplicación de la ley
```

Reglas de resolución de conflictos en la aplicación de plantillas

Cuando más de una plantilla podría ser aplicada a un elemento, existen una serie de reglas que resuelven en qué orden se aplicarán. Cada regla tendrá una prioridad, que puede ir de -9 a 9, pero que al aplicarse automáticamente tendrá los siguientes valores.

Patrón	Prioridad
"*", "@*", "text ()" y otros patrones para los que no se indica un nombre concreto	-0.5
"Prefijo:*", "@prefijo:*" y otros patrones para los que se indica el espacio de nombres, pero no un nombre concreto	-0.25
"producto", "@unidades" y otros patrones donde se indiquen nombres de elementos y atributos concretos	0
"proveedor/representante", "productos/producto/precio", "precio/@unidades" y otros patrones para los que se indique una jerarquía (además del nombre del elemento y/o atributo)	0.5

Tabla 6.1: Prioridad de patrones

Según la especificación de XSLT, cuando aparecen varias plantillas con la misma prioridad, o se produce un error o se aplica la última plantilla en aparecer. En la práctica, los procesadores XSLT aplican la última regla que aparece y siguen procesando.

En resumen, cuanto más específico es el patrón, mayor prioridad tiene.

Asignación explícita de prioridades a plantillas

Se puede asignar a las plantillas prioridades de manera explícita mediante el atributo `priority`. Si a una plantilla se le asigna una prioridad mayor que 0.5 (ej. 1) se asegura que esta plantilla tendrá prioridad en su aplicación que cualquier plantilla que no incluya una declaración explícita de prioridad.

Modos de las plantillas

En ocasiones una hoja de estilos tendrá que acceder varias veces a un mismo nodo, enviando a la salida diferentes resultados cada vez. En estos casos, se utilizará el atributo de las plantillas `mode`, que permitirá etiquetar cada plantilla, de manera que puedan ser invocadas por el texto de la etiqueta.

Actividad 6.1:

¿Cuál de los siguientes, A, B, C o D, es el resultado adecuado de una transformación al aplicar al siguiente documento XML la siguiente hoja de transformaciones XSLT?

```
<?xml version="1.0" ?>
<ListaProductos>
  <Titulo>Series XML</Titulo>
    <Producto>
      <Nombre>Unidad XML</Nombre>
      <PrecioUd>200</PrecioUd>
    </Producto>
</ListaProductos>
```

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="ListaProductos/Producto"/>
        <xsl:apply-templates select="ListaProductos/Auxiliar"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="/ListaProductos/Producto">
    - Nombre : <xsl:value-of select="Nombre" /><br/>
    - Precio ud. : <xsl:value-of select="PrecioUd" />€<br/>
  </xsl:template>

  <xsl:template match="ListaProductos/Producto">
    § Nombre : <xsl:value-of select="Nombre" /><br/>
    § Precio ud. : <xsl:value-of select="PrecioUd" />€<br/>
  </xsl:template>

  <xsl:template match="Auxiliar">
    Auxiliar : <br/>
    <xsl:value-of select="." /><br/>
  </xsl:template>

</xsl:stylesheet>
```

A.

```
<html>
  <body>
    - Nombre : Unidad XML<br>
    - Precio ud. : 200€<br>
  </body>
</html>
```

B.

```
<html>
  <body>
    § Nombre : Unidad XML<br>
    § Precio ud. : 200€<br>
  </body>
</html>
```

C.

```
<html>
  <body>
    - Nombre : Unidad XML<br>
    - Precio ud. : 200€<br>
    Auxiliary : <br>
  </body>
</html>
```

D.

```
<html>
  <body>
    § Nombre : Unidad XML<br>
    § Precio ud. : 200€<br>
    Auxiliary : <br>
  </body>
</html>
```

Ejemplo:

Se van a definir dos plantillas con igual atributo `match`. Al ser invocadas desde algún `<apply-templates>`, y según las normas de resolución de conflictos en la aplicación de plantillas, se aplicaría siempre la última en aparecer. Para poder evitar esto y aplicar la que nos convenga en cada momento, se empleará el atributo `mode`, tanto en la instrucción `<apply-templates>` como en `<templates>`. A cada plantilla se le asignará una etiqueta diferente (distinto valor para el atributo `mode`), lo que permitirá distinguir una de otra en la invocación.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xslt:stylesheet xmlns:xslt="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xslt:output method="text" encoding="iso-8859-1" />

  <xslt:template match="/">
    Introducción: <xslt:apply-templates
select="contratoAlquiler/vivienda" mode="breve" />

    Detallado: <xslt:apply-templates
select="contratoAlquiler/vivienda" mode="datallado" />
  </xslt:template>

  <xslt:template match="vivienda" mode="datallado">
    <xslt:apply-templates />
  </xslt:template>

  <xslt:template match="vivienda" mode="breve">
    <xslt:value-of select="via" />
  </xslt:template>

</xslt:stylesheet>
```

La salida será:

```
    Introducción: Cruz del sur

    Detallado:
    Calle
    Cruz del sur
    73
    Izquierda
    10
    C
```

AVISO: Todos los ejemplos que se muestran deben aparecer como contenido del elemento raíz de toda hoja de transformaciones, `<xsl:stylesheet>`, sin embargo, y por evitar repetición de código, se omitirá a partir de ahora en la mayoría de ejemplos. De igual forma, se omitirá la instrucción de procesamiento que declara un documento como XML, `<?xml ... ?>`.

Actividad 6.2:

En un documento XML que representa una factura existe un nodo de nombre `datosFactura` que contiene información general de la factura (fecha, forma de pago...) y además una lista de ítems (descripción, precio unitario, cantidad...) que conforman las factura.

Se quiere acceder a este nodo en dos ocasiones diferentes:

- Una para mostrar en contenido general de la factura
- Otra para visualizar los ítems de la factura

Escribe dos plantillas que tengan la ruta el nodo `datosFactura` como valor de su atributo `match`, pero se distinguirán una de otra con el atributo `mode`, que en el primer caso valdrá *general* y en el segundo *items*.

Ejemplo:

Se definen tres plantillas cuyo atributo `match` podría coincidir con el elemento `/contratoAlquiler/arrendatario`. Al ser invocadas desde el `<apply-templates>` de la plantilla principal, se produce un conflicto entre ellas. Aquí entran en juego las reglas de resolución de conflictos entre plantillas, que determinan que la expresión que aparece en el atributo `match` de una plantilla la hace más prioritaria si es una ruta completa (prioridad 0.5), ya sea relativa al elemento desde donde se invocó (`contratoAlquiler/arrendatario`), ya sea absoluta respecto a la raíz del documento XML (`/contratoAlquiler/arrendatario`). La plantilla menos prioritaria será aquella en la que sólo aparece el nombre del elemento (`arrendatario`) con una prioridad de 0. De manera que hay dos plantillas con prioridad 0.5 y una con prioridad 0. Siempre se aplicarán primero alguna de las de 0.5, pero ¿cuál? La que aparezca después en la declaración de plantillas.

```
    <xsl:template match="/">
      <xsl:apply-templates select="contratoAlquiler/arrendatario" />
    </xsl:template>

  <!-- Plantilla 1; Direccionamiento relativo respecto al nodo de
contexto; Prioridad 0.5 -->
    <xsl:template match="/contratoAlquiler/arrendatario">
      Plantilla n°: 1
      Ruta: /contratoAlquiler/arrendatario
    </xsl:template>

  <!-- Plantilla 2; Direccionamiento absoluto respecto a la raíz;
Prioridad 0.5 -->
    <xsl:template match="contratoAlquiler/arrendatario">
      Plantilla n°: 2
      Ruta: contratoAlquiler/arrendatario
    </xsl:template>
```

```
  <!-- Plantilla 3; Descripción de un elemento o atributo sin
direccionarlo; Prioridad 0 -->
    <xsl:template match="arrendatario">
      Plantilla nº: 3
      Ruta: arrendatario
    </xsl:template>
```

La salida será la correspondiente a aplicar la plantilla número 2.

Plantilla nº: 2

Ruta: contratoAlquiler/arrendatario

xsl:call-template

Permite invocar a una plantilla por su nombre. Al aplicarse la plantilla con nombre, el nodo de contexto no cambia, lo que significa que en la plantilla invocada hay que utilizar el mismo direccionamiento de elementos que había en la invocante

Atributos obligatorios:

- `name`: nombre de la plantilla a la que se llama.

Ejemplo:

Se partirá de una hoja de transformaciones con unas sola plantilla asociada al elemento raíz `/pedido`. En ella se muestra el atributo `nombre` de los diferentes `/productos/producto`. Se van a usar para esto dos etiquetas, `<xsl:for-each>` y `<xsl:value-of>`, que se estudiará a continuación.

```
  <xsl:template match="/pedido">
    <xsl:for-each select="productos/producto">
      Producto: <xsl:value-of select="nombre" />
    </xsl:for-each>
  </xsl:template>
```

A continuación, todo el contenido de la transformación se lleva inalterado a otra plantilla de nombre `listaProductos`, a la que se invoca por su nombre. Las expresiones XPath usadas dentro de esta nueva plantilla se mantienen como estaban (atributo `select` de las etiquetas `<xsl:for-each>` y `<xsl:value-of>`).

```
  <xsl:template match="/pedido">
    <xsl:call-template name="listaProductos" />
  </xsl:template>

  <xsl:template name="listaProductos">
    <xsl:for-each select="productos/producto">
      Producto: <xsl:value-of select="nombre" />
    </xsl:for-each>
  </xsl:template>
```

xsl:value-of

Visualiza el contenido del elemento indicado en el atributo `select`, y de todos sus descendientes.

Atributos obligatorios:

- `select`: Es una expresión XPath que especifica de qué elemento o atributo hay que extraer el contenido.

Atributos optativos principales:

- `disable-output-escaping`: Indica si los caracteres especiales, como < o &, deben ser enviados a la salida tal cual o en su forma de entidad, `<` o `&`. El valor por defecto es *no*, es decir, se generarán en su forma de entidad.

Ejemplo:

El resultado es una página HTML que se basa en un documento XML que contiene un elemento `/componentes`. Todo el contenido de ese elemento se mostrará en la página de salida formateado con `<h1>`.

```
<xsl:template match="/">
  <html>
  <body>
    <h1><xsl:value-of select="componentes" /></h1>
  </body>
  </html>
</xsl:template>
```

Versión simplificada para mostrar valores de atributos

Existe una forma simplificada de esta instrucción pero sólo se puede usar en un caso muy específico: cuando se está generando un nuevo elemento (etiqueta) en el documento de salida y se quiere dar un valor a alguno de sus atributos.

Ejemplo:

Se parte del siguiente documento XML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="estaciones3.xsl"?>
<imagenes>
  <imagen ancho="149" alto="149">
    <archivo extension="jpg">Spring</archivo>
    <texto>Primavera</texto>
  </imagen>
  ...
</imagenes>
```

Se va a generar como salida un nuevo elemento `<img>` que se corresponderá con cada uno de los elementos `<imagen>` del documento de entrada.

Se rodea con llaves el valor del atributo `select` de la orden `<xsl:value-of>`, es decir, `<xsl:value-of select="nombre" />` es equivalente a `{nombre}`.

```
<xsl:template match="/imagenes/imagen">
  <img src="{archivo}.{archivo/@extension}" width="{@ancho}"
        height="{@alto}" alt="{texto}" />
</xsl:template>
```

xsl:text

Permite escribir texto literal en la salida.

En las hojas de transformaciones, los caracteres de espacios en blanco (saltos de línea, tabuladores, espacios en blanco) se envían directamente a la salida. Para controlar este comportamiento, se usa la instrucción `<xsl:text>`.

Atributos optativos principales:

- `disable-output-escaping`: indica si los caracteres especiales, como < o &, deben ser enviados a la salida tal cual o en su forma de entidad, `<` o `&`. El valor por defecto es *no*, es decir, se generarán en su forma de entidad.

Ejemplo:

Se envía un carácter de salto de línea a la salida de dos maneras diferentes.

```
<xsl:template match="/">
  <xsl:text>Texto inicial.</xsl:text>
  <xsl:text>&#10;</xsl:text>  ❶
  Texto intermedio.
  <xsl:text>
</xsl:text>  ❷
  <xsl:text>Texto final.</xsl:text>
</xsl:template>
```

❶ Se envía un salto de línea a la salida recurriendo a su código `UNICODE`.

❷ Se envía un salto de línea a la salida introduciéndolo literalmente dentro de una etiqueta `<xsl:text>` de la hoja de transformaciones.

6.2.3. Instrucciones de control

Existen una serie de etiquetas que permiten reproducir el funcionamiento de las instrucciones de control de flujo de los lenguajes imperativos.

xsl:for-each

Permite iterar sobre una lista de elementos y aplicar transformaciones sobre ellos.

Atributos obligatorios:

- `select`: expresión XPath que define la lista de elementos sobre los que se va a iterar.

Las instrucciones `<for-each>` se pueden anidar para recorrer estructuras anidadas.

Ejemplo:

Se dispone de un documento XML de partida que representa una tabla con palabras en castellano y en inglés. Se va a convertir en un documento HTML de salida que dibuje la tabla, con las palabras ya mencionadas precedidas de la posición de las mismas en la tabla.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<tabla>
  <fila>
    <columna valor="Hola" />
    <columna valor="Adios"/>
    <columna valor="Buenos días"/>
  </fila>
  <fila>
    <columna valor="Hello"/>
    <columna valor="Good bye"/>
    <columna valor="Good morning"/>
  </fila>
</tabla>
```

La hoja de transformaciones será:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" encoding="iso-8859-1"/>

  <xsl:template match="/tabla">
  <html><body>
      <h1>TABLA</h1>
      <!-- empieza a generar la tabla si al menos tiene una fila -->
      <xsl:if test="count(fila) &gt; 1">
      <table border="1">
        <xsl:for-each select="fila">
        <tr>
          <xsl:for-each select="columna">
          <td><xsl:value-of select="./@valor" /></td>
          </xsl:for-each>
        </tr>
        </xsl:for-each>
      </table>
      </xsl:if>
  </body></html>
  </xsl:template>

</xsl:stylesheet>
```

La salida queda:

```
<html>
<body>
<h1>TABLA</h1>
<table border="1">
<tr>
  <td>Hola</td>
  <td>Adios</td>
  <td>Buenos d&iacute;as</td>
</tr>
<tr>
  <td>Hello</td>
  <td>Good bye</td>
  <td>Good morning</td>
</tr>
</table>
</body>
</html>
```

El mismo efecto se hubiera conseguido con la siguiente hoja de transformaciones. Lo que se hace es crear una plantilla que se aplique para cada fila y otra que se aplique para cada columna dentro de una fila.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" encoding="iso-8859-1"/>
  <xsl:template match="/tabla">
  <html><body>
      <h1>TABLA</h1>
      <!-- empieza a generar la tabla si al menos tiene una fila -->
      <xsl:if test="count(fila) &gt; 1">
        <table border="1">
          <xsl:apply-templates select="fila" />  ❶
        </table>
      </xsl:if>
  </body></html>
  </xsl:template>
  <xsl:template match="fila">
    <tr>
      <xsl:apply-templates select="columna" />  ❷
    </tr>
  </xsl:template>
  <xsl:template match="columna">
  <xsl:param name="fila" />
    <td><xsl:value-of select="./@valor" /></td>
  </xsl:template>
</xsl:stylesheet>
```

❶ Se invoca la plantilla para cada elemento `/tabla/fila`.

❷ Se invoca la plantilla para cada elemento `/tabla/fila/columna`.

xsl:sort

Envía a la salida datos del documento XML de partida ordenados por algún criterio.

Atributos optativos principales:

- `select`: es una expresión XPath que indica el nodo o conjunto de nodos por los que constituirán el criterio de ordenación.
- `lang`: indica qué idioma se utiliza al ordenar. Es un código de idioma.
- `data-type`: especifica el tipo de datos de los datos que se ordenarán.

  Posibles valores: ***text***, *number*, *QName*
- `order`: concreta si el orden es ascendente o descendente.

  Posibles valores: ***ascending***, *descending*
- `case-order`: especifica si las letras en minúscula (*lower-first*) o mayúscula (*upper-first*) aparecerán primero al ordenar.

  Posibles valores: *lower-first*, *upper-first*

Ejemplo:

Se desea mostrar los nombres de los módulos profesionales y las siglas del ciclo o ciclos a los que pertenecen. Se ordenará descendentemente por nombre de módulo.

```
<?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>
  <xsl:for-each select="fP/modulo/modulo">
    <xs:sort select="nombre" order="descending"/>
      <xsl:value-of select="nombre"/> 
      <xsl:value-of select="ciclos/ciclo/@siglas"/>
  </xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Actividad 6.3:

Modifica el código anterior de manera que se muestre la misma información de salida, pero esta vez ordenada ascendentemente por siglas de ciclos.

Se podría ordenar una salida por más de un criterio, anidando varias órdenes `<xsl:sort>`.

El ejemplo anterior, ordenando ascendentemente primero por siglas de ciclo y luego por nombre de módulo, sería:

```
<xsl:sort select=" ciclos/ciclo/@siglas" />
<xsl:sort select="nombre" />
```

xsl:if

Se utiliza para producir un comportamiento condicional. Sólo permite preguntar por una condición y actuar de una manera si la condición es cierta, es decir, no existe una rama del sino (*else*).

La plantilla que viene a continuación se aplicará si el resultado de evaluar la expresión XPath que aparece en el atributo test es cierta.

Atributos obligatorios:

- `test`: expresión XPath que ha de hacerse cierta para que se aplique la platilla que viene a continuación.

Ejemplo:

Se dibujará una tabla con colores alternos para las filas pares/impares.

```
<xsl:template match="/">
  <html>
  <body>
    <h2>Librería</h2>
    <table border="1">
    <tr style="color: navy;">
      <th>Título</th>
      <th>Autor</th>
    </tr>
    <xsl:for-each select="libreria/libro ">
      <xsl:if test="position() mod 2 = 1"> <!-- Filas impares -->
        <tr>
          <td><xsl:value-of select="titulo"/></td>
          <td><xsl:value-of select="autor"/></td>
        </tr>
      </xsl:if>
      <xsl:if test="position() mod 2 = 0"> <!-- Filas pares -->
       <tr style="color: light blue;">
          <td><xsl:value-of select="titulo"/></td>
          <td><xsl:value-of select="autor"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

xsl:choose, xsl:when, xsl:otherwise

Permiten implementar un comportamiento condicional con múltiples opciones y una opción por defecto. Es el equivalente a la instrucción `switch` de algunos lenguajes como *C* o *Java*.

Ejemplo:

Dado un elemento vacío, `<Notas>`, que puede o no tener contenido textual, cuando lo tiene se quiere mostrar y cuando no lo tiene se mostrará *Sin notas*. El documento XML de entrada es:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"?>
<ListaProductos>
  <Producto cod="XML001">
    <Nombre>Curso bádico de XML</Nombre>
    <PrecioUd>500</PrecioUd>
    <Notas>Limitado a 30 personas</Notas>
  </Producto>
  <Producto cod="XML002">
    <Nombre>Curso avanzado de XML</Nombre>
    <PrecioUd>150</PrecioUd>
    <Notas/>
  </Producto>
</ListaProductos>
```

La hoja de transformaciones será:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
    <body>
      <h1>Lista Productos</h1>
      <table border="1">
      <tr><th>Nombre</th><th>Precio ud.</th><th>Remarks
      </th></tr>
      <xsl:apply-templates select="ListaProductos/Producto" />
      </table>
    </body>
    </html>
  </xsl:template>

  <xsl:template match="Producto">
    <tr>
      <td><xsl:value-of select="Nombre" /></td>
      <td><xsl:value-of select="PrecioUd" /></td>
      <xsl:choose>
        <!-- Es cierto cuando el elemento Notas contiene texto -->
        <xsl:when test="Notas/text()">
          <td><xsl:value-of select="Notas" /></td>
        </xsl:when>
```

```
          <xsl:otherwise>
            <td>Sin notas</td>
          </xsl:otherwise>
        </xsl:choose>
      </tr>
    </xsl:template>

  </xsl:stylesheet>
```

6.2.4. Generación de nuevos elementos y atributos

Cuando la salida a generar es en formato XML, se podrá optar por enviar a la salida elementos/atributos existentes en el documento XML de entrada, o bien crearlos nuevos.

xsl:element

Permiten generar un nuevo elemento en el documento de salida (nodo).

Atributos obligatorios:

- `name`: especifica el nombre del elemento. Este nombre se puede construir dinámicamente extrayendo datos del documento XML de entrada.

  Ejemplo:

  ```
  <xsl:element name="{pais[1]/nombre}" namespace="{pais[1]/url}">
  ```

Atributos optativos principales:

- `namespace`: define el URI del espacio de nombres del atributo. Este URI se puede construir dinámicamente extrayendo datos del documento XML de entrada.
- `use-attribute-sets`: permite indicar una lista (separada por espacios) de conjuntos de atributos (attribute-sets) que contienen atributos para añadir al elemento.

Ejemplo:

```
<xsl:element name="p">
  <xsl:attribute name="id">primerParrafo</xsl:attribute>
  <xsl:attribute name="class">parrafos</xsl:attribute>
  Texto del párrafo...
</xsl:element>
```

Esta transformación generará en la salida:

```
<p id="primerParrafo" class="parrafos">Texto del párrafo...</p>
```

xs:attribute

Permite generar un nuevo atributo de un elemento.

Atributos obligatorios:

- `name`: especifica el nombre del atributo.

Atributos optativos principales:

- namespace: define el URI del espacio de nombres del atributo.

Se puede generar un atributo con un valor literal.

Ejemplo:

Se dispone de un documento XML de entrada, *estaciones.xml*, al que se le quiere aplicar una hoja de transformaciones, *estaciones.xsl*, de manera que se genere un documento de salida, *estacionesTransformado.xml*.

El documento de entrada es:

```
<!-- estaciones.xml -->
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="estaciones.xsl"?>
<imagenes>
  <imagen ancho="149" alto="149">
    <archivo extension="jpg">Spring</archivo>
    <texto>Primavera</texto>
  </imagen>
  <imagen ancho="149" alto="149">
    <archivo extension="jpg">Summer</archivo>
    <texto>Verano</texto>
  </imagen>
  <imagen ancho="149" alto="149">
    <archivo extension="jpg">Autumn</archivo>
    <texto>Otoño</texto>
  </imagen>
  <imagen ancho="149" alto="149">
    <archivo extension="jpg">Winter</archivo>
    <texto>Invierno</texto>
  </imagen>
</imagenes>
```

En la hoja de transformaciones se van a crear cuatro atributos para el elemento `<img>`. Son `src`, `width`, `height` y `alt`. El valor de cada uno de ellos será el de algún nodo o elemento del documento XML de origen. El código es:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" encoding="iso-8859-1" />

  <xsl:template match="/imagenes/imagen">
    <img>
      <xsl:attribute name="src">
        <xsl:value-of select="archivo" />.<xsl:value-of
select="archivo/@extension" />
      </xsl:attribute>
      <xsl:attribute name="width">
        <xsl:value-of select="./@ancho" />
      </xsl:attribute>
      <xsl:attribute name="height">
```

```
          <xsl:value-of select="./@alto" />
        </xsl:attribute>
        <xsl:attribute name="alt">
          <xsl:value-of select="./texto" />
        </xsl:attribute>
      </img>
    </xsl:template>

  </xsl:stylesheet>
```

El documento de salida, ya transformado, es:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<img src="Spring.jpg" width="149" height="149" alt="Primavera"/>
<img src="Summer.jpg" width="149" height="149" alt="Verano"/>
<img src="Autumn.jpg" width="149" height="149" alt="Otoño"/>
<img src="Winter.jpg" width="149" height="149" alt="Invierno"/>
```

Actividad 6.4:

Como ya se comentó en un ejemplo anterior, se podría usar la notación simplificada (con llaves de apertura y de cierre) para generar los valores de los nuevos atributos.

Ejemplo:

```
<img width="{@ancho}" />
```

Modifica la hoja de transformaciones previa para que todos los atributos se generen de esta manera.

---

xsl:comment

Genera un comentario en el documento de salida. Puede ser un texto literal o un dato extraído del documento XML de partida.

Ejemplo:

Diferencia entre un comentario en de la hoja de transformaciones (no se envía a la salida) frente a un comentario que se enviará a la salida.

```
<xsl:template match="/">
  <!-- Este es un comentario en la hoja de transformaciones -->
  <xsl:comment> Este texto se enviará comentado a la salida
  </xsl:commnet>
</xsl:template>
```

xsl:processing-instruction

Permite generar instrucciones de procesamiento en el documento de salida. Esto puede ser necesario, por ejemplo, en el caso de un documento de partida XML que quiera ser transformado mediante una hoja de transformaciones XSLT en otro documento XML, que a su vez vaya a ser transformado por una segunda hoja de transformaciones en un documento HTML. El documento XML intermedio, generado a partir del XML de partida y la primera hoja XSLT, deberá incluir una instrucción de procesamiento indicando qué hoja de estilos se le debe aplicar para obtener el HTML final.

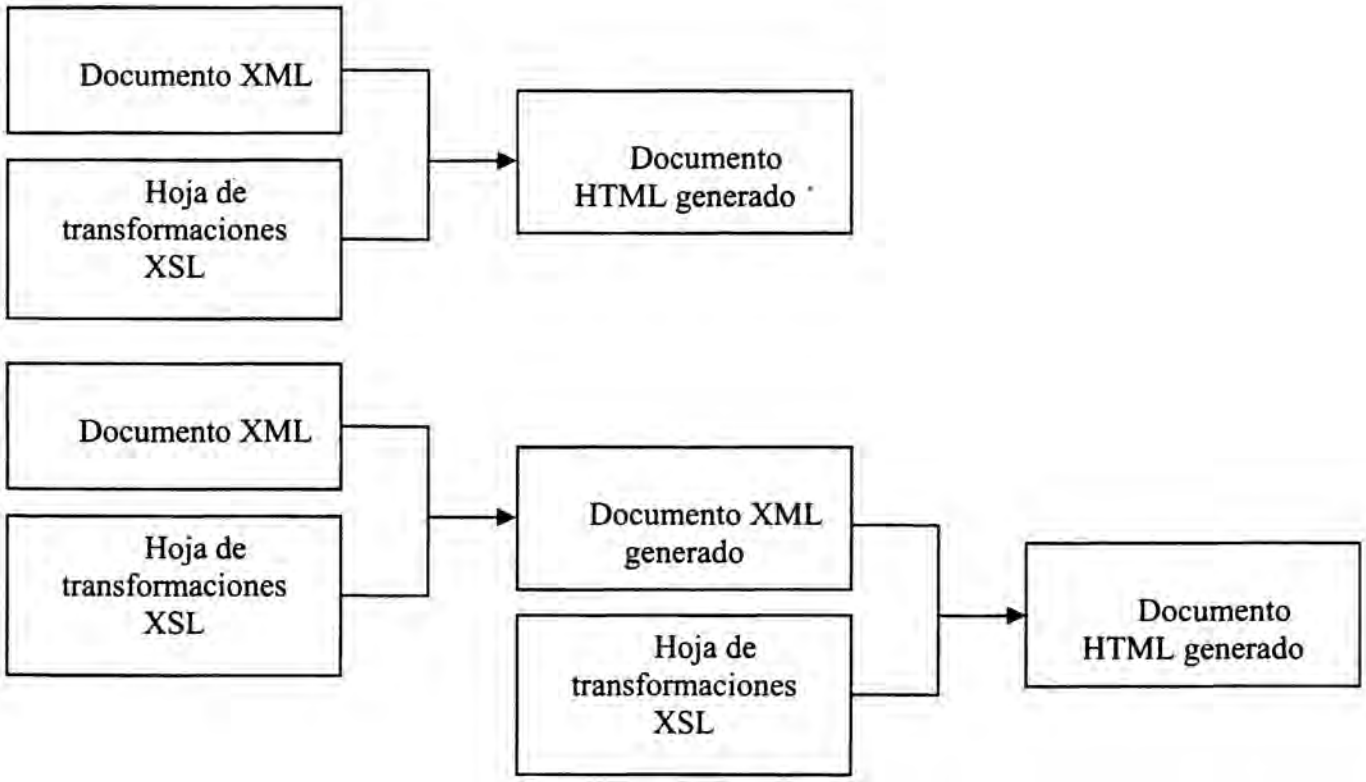


Figura 6.4: Transformaciones transformadas

Ejemplo:

El siguiente código generará en la salida una instrucción de procesamiento llamada `xml-stylesheet`, que contendrá dos atributos, `href` y `type`.

```
<xsl:processing-instruction name="xml-stylesheet">
  href="style.css" type="text/css"
</xsl:processing-instruction>
```

La salida generada es:

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

xsl:copy-of

Realiza una copia literal de un elemento a la salida. Los atributos y elementos descendientes se copiarán también. Su uso más habitual es cuando se quiere enviar a la salida un fragmento (o la totalidad) del documento XML de entrada sin aplicarle ninguna transformación.

Atributos obligatorios:

- `select`: especifica el elemento que se copiará.

Ejemplo:

Del documento XML de partida relativo a vuelos, se quiere extraer directamente la información de los aviones.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Vuelos>
  <Vuelo código="IB4001">
    <Compañía Siglas="IB">Iberia</Compañía>
    <Opera Siglas="AA">American Airlines</Opera>
    ...
    <Avión>
      <Marca>Boeing</Marca>
      <Modelo>757-200 WINGLETS</Modelo>
    </Avión>
  </Vuelo>

  <Vuelo código="QF107">
    <Compañía Siglas="QF">Qantas</Compañía>
    ...
    <Avión>
      <Marca>Boeing</Marca>
      <Modelo>747-400</Modelo>
    </Avión>
  </Vuelo>
</Vuelos>
```

La hoja de transformaciones sería:

```
<xsl:template match="/Vuelos">
  <Aviones>
    <xsl:copy-of select="Vuelo/Avión"/>
  </Aviones>
</xsl:template>
```

Y la salida generada:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Aviones>
  <Avión>
     <Marca>Boeing</Marca>
     <Modelo>757-200 WINGLETS</Modelo>
  </Avión>
  <Avión>
    <Marca>Boeing</Marca>
```

```
      <Modelo>747-400</Modelo>
    </Avión>
  </Aviones>
```

Actividad 6.5:

Con la instrucción `<xsl:copy-of>`, crea una hoja de transformaciones que envíe a la salida el documento XML de entrada íntegro, sea éste cual sea.

xsl:copy

Se emplea para realizar una copia de un nodo, pero no de sus descendientes ni atributos.

Atributos optativos principales:

- `use-attribute-sets`: Indica una lista de conjuntos de atributos, separados por espacio, que se aplicarán al nodo de salida.

Ejemplo:

El comportamiento de `<xsl:copy>` aquí es equivalente a crear una hoja de transformaciones sin plantillas: se envía sólo el contenido textual a la salida.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>

<xsl:template match="/">
 <xsl:copy>
  <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

Ejemplo:

Se van a enviar a la salida los elementos, sin atributos, ni texto.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="estaciones">
  <xsl:copy>
    <xsl:apply-templates select="estacion"/>
  </xsl:copy>
</xsl:template>
<xsl:template match="estacion">
  <xsl:apply-templates select="nombre"/>
  <xsl:apply-templates select="comienzo"/>
 <xsl:copy/>
</xsl:template>
```

```
<xsl:template match="nombre">
  <xsl:copy />
</xsl:template>

<xsl:template match="comienzo">
  <xsl:copy />
</xsl:template>

</xsl:stylesheet>
```

La salida será:

```
<?xml version="1.0" encoding="UTF-8"?>
<estaciones>
  <nombre/>
  <comienzo/>
  <estacion/>
  <nombre/>
  <comienzo/>
  <estacion/>
  ...
</estaciones>
```

Ejemplo:

Se envía a la salida el nodo de contexto, `/estaciones`, y se le asocia un **conjunto de atributos** (se verá más adelante) que contiene tres atributos, `numero`, `solsticios` y `equinoccios`.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>

<xsl:attribute-set name="datosEstaciones">
  <xsl:attribute name="numero">4</xsl:attribute>
  <xsl:attribute name="solsticios">2</xsl:attribute>
  <xsl:attribute name="equinoccios">2</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="/estaciones">
  <xsl:copy use-attribute-sets="datosEstaciones" />
</xsl:template>
</xsl:stylesheet>
```

La salida será:

```
<?xml version="1.0" encoding="UTF-8"?>
<estaciones numero="4" solsticios="2" equinoccios="2"/>
```

6.2.5. Elementos avanzados de una hoja de transformaciones

Existen una serie de elementos cuyo manejo permite afinar en el diseño de hojas de transformaciones, pero que no son esenciales para su creación.

xsl:preserve-space y xsl:strip-space

Son elementos de nivel superior que indican en qué elementos se deben mantener los espacios en blanco que contuviera su texto, con `<xsl:preserve-space>`, y en cuáles se deben suprimir, con `<xsl:strip-space>`.

Atributos obligatorios:

- `elements`: lista de nombres de elementos, separados por un espacio en blanco, que deben preservar/suprimir los espacios en blanco que contenga su texto asociado. Se puede indicar * para indicar todos los elementos. Se puede un espacio de nombres para indicar que se aplica a todos los elementos de ese espacio de nombres (ej. `prefijo:*`).

Ejemplo:

En el caso del documento de vuelos previamente citado, se podría indicar que los elementos `Compañía` y `Opera` mantengan sus espacios, mientras que `Marca` y `Modelo` los supriman.

```
<xsl:preserve-space elements="Compañía Opera" />
<xsl:strip-space elements="Marca Modelo" />
```

xsl:import

Es un elemento de nivel superior que se emplea para importar una hoja de transformaciones desde otra. Debe aparecer como primer hijo de `<xsl:stylesheet>` (o `<xsl:transform>`).

La hoja importada tiene menor precedencia que la que la importadora.

Atributos obligatorios:

- `href`: especifica la URI de la hoja de transformaciones que se va a importar.

xsl:apply-imports

Permite aplicar plantillas que se encuentren en una hoja de estilos importada. Se utiliza cuando se quiere forzar el uso de plantillas de la hoja importada, ya que si no se especificara nada, las plantillas de la hoja importadora tienen precedencia sobre las de la hoja importada.

Ejemplo:

Supóngase existente una hoja de transformaciones *importada.xsl*. Se importará y se aplicarán sus plantillas (`<xsl:apply-imports/>`) en la hoja de transformaciones siguinete:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:import href="importada.xsl"/>
<xsl:template match="/">
  <xsl:apply-imports/>
</xsl:template>
</xsl:stylesheet>
```

xsl:include

Es un elemento de nivel superior que se emplea para incluir textualmente una hoja de transformaciones desde otra. Debe aparecer como primer nodo hijo de `<xsl:stylesheet>` (o `<xsl:transform>`).

Atributos obligatorios:

- `href`: especifica la URI de la hoja de transformaciones que se va a incluir.

Ejemplo:

Un uso habitual de <xsl:include> será disponer de una hoja de transformaciones con una "librería de plantillas", que puedan ser usadas en distintos casos, e incluir esta librería desde una hoja de transformaciones principal:

```
<xsl:include href="libreria.xsl"/>
```

xsl:variable

Las variables en XSLT son similares a las constantes en otros lenguajes de programación, es decir, en el momento que se les ha asignado un valor, ya no puede ser modificado. Se usan para almacenar expresiones XPath complejas o trozos de código que vayan a usarse repetidas veces.

Atributos obligatorios:

- `name`: especifica el nombre de la variable.

Atributos optativos principales:

- `select`: define el valor de la variable. Si no aparece en la declaración de la variable, se asume que el valor es la cadena vacía.

Ejemplo:

Definir, por medio de una variable, una cabecera de una tabla HTML que se enviará a la salida en varios lugares de la hoja de transformación.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="cabecera">  ❶
  <tr>
    <th>Título</th>
    <th>Autor</th>
  </tr>
</xsl:variable>
```

```
<xsl:template match="/">
<html>
<body>
  <table>
    <xsl:copy-of select="$cabecera" />  ❷
    <xsl:for-each select="mediateca/libro">
      <tr>
      <xsl:if test="genero='Histórico'">
        <td><xsl:value-of select="titulo"/></td>
        <td><xsl:value-of select="autor"/></td>
      </xsl:if>
      </tr>
    </xsl:for-each>
  </table>
  <table>
    <xsl:copy-of select="$cabecera" />  ❷
    <xsl:for-each select="mediateca/video">
      <tr>
      <xsl:if test="genero='Histórico'">
        <td><xsl:value-of select="titulo"/></td>
        <td><xsl:value-of select="autor"/></td>
      </xsl:if>
      </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

❶ Se declara una variable `cabecera` que contiene la cabecera de una tabla.

❷ Se referencia la variable, como `$cabecera`, mandando su contenido a la salida.

xsl:with-param

Permite definir un parámetro que se le pasará a una plantilla. Dentro de la plantilla invocada debe haber una instrucción `<xsl:param>` asociada (con igual valor del atributo `name`) en la que se reciba el parámetro. De no haberla, la instrucción `<xsl:with-param>` se ignorará.

Atributos obligatorios:

- `name`: especifica el nombre del parámetro.

Atributos optativos principales:

- `select`: expresión XPath que define el valor del parámetro.

xsl:param

Se usa para declarar un parámetro. El parámetro es global si se declara como elemento de alto nivel (hijo de `<xsl:stylesheet>`).

Si se declara dentro de una plantilla es local y debe ir asociado, por el atributo `name`, a la instrucción `<xsl:with-param>` donde se declaró en la invocación de la plantilla.

Atributos obligatorios:

- name: especifica el nombre del parámetro.

Atributos optativos principales:

- select: expresión XPath que define el valor por defecto del parámetro, por si no recibe ninguno a través de la instrucción <xsl:with-param> asociada.

Ejemplo:

A partir del documento XML visto anteriormente que reflejaba una estructura de tabla en forma de filas y columnas, se quiere obtener el siguiente documento HTML de salida:

```
<html>
<body>
<h1>TABLA</h1>
<table border="1">
<tr>
  <td>(1, 1) - Hola</td>
  <td>(1, 2) - Adios</td>
  <td>(1, 3) - Buenos d&iacute;as</td>
</tr>
<tr>
  <td>(2, 1) - Hello</td>
  <td>(2, 2) - Good bye</td>
  <td>(2, 3) - Good morning</td>
</tr>
</table>
</body>
</html>
```

Se utiliza la siguiente hoja de transformaciones donde se define un parámetro en la segunda plantilla, que se pasará a la tercera, donde se recibe y se usa:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" encoding="iso-8859-1"/>

  <xsl:template match="/tabla">
  <html>
    <body>
      <h1>TABLA</h1>
      <!-- empieza a generar la tabla si al menos tiene una fila -->
      <xsl:if test="count(fila) &gt; 1">
        <table border="1">
          <xsl:apply-templates select="fila" />
        </table>
      </xsl:if>
    </body>
  </html>
  </xsl:template>
```

```
    <xsl:template match="fila">
      <tr>
      <xsl:apply-templates select="columna">
        <xsl:with-param name="fila" select="position()" />
      </xsl:apply-templates>
      </tr>
    </xsl:template>

    <xsl:template match="columna">
    <xsl:param name="fila" />
      <td>(<xsl:value-of select="$fila" />, <xsl:value-of
select="position()" />) - <xsl:value-of select="./@valor" /></td>
    </xsl:template>

  </xsl:stylesheet>
```

xsl:number

Se utiliza para determinar la posición de un determinado nodo, así como para formatear la salida numérica.

Atributos optativos principales:

- `count`: expresión XPath que determina qué nodos deben ser contados.
- `level`: controla cómo se asigna la secuencia numérica.

  Posibles valores: ***single***, *multiple*, *any*.
- `from`: expresión XPath que indica a partir de dónde se empieza a contar.
- `value`: especifica el número provisto por el usuario al que se le aplicará el formato, en lugar de una secuencia de números generada.
- `format`: define el formato de salida del número. Puede ser 1, 01, a, A, i o I.
- `grouping-separator`: especifica qué carácter se usa como separador de grupos de dígitos, siendo por defecto la coma (,).
- `grouping-size`: especifica cuántos dígitos forman cada grupo de los que se separará con el carácter de `grouping-separator`. Por defecto es 3.

Ejemplo:

Se quieren mostrar los nombres de los ciclos formativos en forma de lista de números romanos. La hoja de transformaciones será:

```
  <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" encoding="iso-8859-1"/>

    <xsl:template match="/ciclos/ciclo">  ❶
      <xsl:number count="*" format="i. "/>  ❷
      <xsl:value-of select="nombre"/>  ❸
    </xsl:template>
  </xsl:stylesheet>
```

❶ Se toman todos los elementos /ciclos/ciclo.

❷ Con la orden `<xsl:number>` se contabilizan y se formatea la salida anteponiendo un número romano y un punto a lo que se quiera visualizar.

❸ Por último, se selecciona el nombre de cada ciclo como dato a visualizar.

La salida será:

i. Sistemas microinformáticos y redes

ii. Administración de sistemas informáticos en red

iii. Desarrollo de aplicaciones multiplataforma

xsl:decimal-format

Permite definir características de formateo al convertir números en cadenas de texto, usando la función `fomat-number()`. Así, se pueden indicar cuál es el separador de decimales, de miles, etc. Es un elemento de nivel superior, hijo de `<xsl:stylesheet>`.

Atributos optativos principales:

- `name`: especifica el nombre de este formato.
- `decimal-separator`: indica el carácter separador de decimales. Por defecto es ".".
- `grouping-separator`: especifica qué carácter separador de miles. Por defecto es ",".
- `infinity`: indica la cadena usada para representar el valor matemático infinito. Por defecto es *infinity*.
- `minus-sign`: especifica el carácter para representar números negativos. Por defecto es "-".
- `NaN`: indica la cadena usada cuando el valor es no numérico. Por defecto es "NaN".
- `percent`: especifica el carácter para el signo de porcentaje. Por defecto es "%".
- `per-mille`: especifica el carácter para el signo de pormil. Por defecto es "‰".
- `zero-digit`: indica el carácter usado para representar el 0. Por defecto es "0".
- `digit`: especifica el carácter usado para indicar un lugar donde se requiere un dígito. Por defecto es "#".
- `pattern-separator`: indica el carácter que se usa para separar subpatrones positivos y negativos en un formato de patrones. Por defecto es ";".

xsl:namespace-alias

Esta instrucción se usa para remplazar un espacio de nombres existente en la hoja de transformaciones por otro diferente en el documento de salida. Su uso más habitual es en el caso en el que se desee generar una hoja de transformaciones XSLT como salida de una transformación.

Atributos obligatorios:

- `stylesheet-prefix`: especifica el espacio de nombres que se quiere sustituir.
- `result-prefix`: especifica el espacio de nombres por el que se quiere sustituir.

Ejemplo:

Se crea una hoja de transformaciones cuya salida será otra hoja de transformaciones. Los elementos que aparecerán en la salida con el prefijo `xsl`, en el hoja de transformaciones aparecerán con el prefijo `otro`.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:otro="http://www.w3.org/1999/XSL/Transform-otro">  ❶

<xsl:namespace-alias stylesheet-prefix="otro" result-prefix="xsl"/>❷
<xsl:param name="navegador" select="'InternetExplorer'"/>

<xsl:template match="/">
   <otro:stylesheet>
      <xsl:choose>
         <xsl:when test="$navegador='InternetExplorer'">
            <otro:import href="IE_Funciones.xsl"/>
            <otro:template match="/">
               <div>
                  <otro:call-template name="muestraTabla"/>
               </div>
            </otro:template>
         </xsl:when>
         <xsl:otherwise>
            <otro:import href="MF_Funciones.xsl"/>
            <otro:template match="/">
               <div>
                  <otro:call-template name="muestraTabla"/>
               </div>
            </otro:template>
         </xsl:otherwise>
      </xsl:choose>
   </otro:stylesheet>
</xsl:template>
</xsl:stylesheet>
```

❶ Se declara el espacio de nombres de prefijo `otro`.

❷ Se indica que, al realizar la transformación, en todas las etiquetas del espacio de nombres de prefijo `otro`, se sustituirá por el prefijo `xsl`.

xsl:attribute-set

Permite crear un paquete de atributos identificado por un nombre, de manera que puedan ser asignados conjuntamente a un elemento.

Atributos obligatorios:

- `name`: especifica el nombre del conjunto de atributos.

Atributos optativos principales:

- `use-attribute-sets`: lista separada por espacios de otros conjuntos de atributos que se integren es éste.

Ejemplo:

Se quiere generar una salida en el lenguaje de formateo FO (que se verá más adelante) de manera que varios elementos de la salida comparten atributos. Se definirá un conjunto de atributos y serán asignados a los elementos que los necesiten.

El código XML de entrada es:

```
<?xml version="1.0"?>
<articulo>
  <parrafo>
    <titulo>Primer apartado</titulo>
    ...
  </parrafo>
  <parrafo>
    <titulo>Segundo apartado</titulo>
    ...
  </parrafo>
</articulo>
```

En la salida, varios elementos `<fo:block>` comparten los atributos `font-size` y `font-weight` y `text-decoration`, cada uno de ellos con su valor asociado:

```
<?xml version="1.0"?>
<fo:block font-size="14pt" font-weight="bold"
          text-decoration="underline"
          xmlns:fo="http://www.w3.org/1999/XSL/Format">
  Primer apartado
</fo:block>
<fo:block font-size="14pt" font-weight="bold"
          text-decoration="underline"
          xmlns:fo="http://www.w3.org/1999/XSL/Format">
  Segundo apartado
</fo:block>
```

En la hoja de transformaciones que lo hace se define un conjunto de atributos, llamado `estiloTitulo`, que incluye los atributos que se usarán repetidas veces. Después se incluye en la declaración del elemento `<fo:block>` con el atributo `xsl:use-attribute-sets`:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

```
<xsl:attribute-set name="estiloTitulo">
  <xsl:attribute name="font-size">14pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="text-decoration">underline</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="parrafo/titulo">
  <fo:block xsl:use-attribute-sets="estiloTitulo">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

</xsl:stylesheet>
```

xsl:fallback

Especifica un código alternativo a ejecutar si el procesador XSLT no es capaz de tratar un elemento `xsl`.

Ejemplo:

Se quiere usar la instrucción `<xsl:loop>`, que en la práctica no existe, pero que podría existir en versiones sucesivas de XSLT. Se añade un código alternativo con `<xsl:for-each>` que se ejecute si el procesador "no sabe qué hacer" con `<xsl:loop>`:

```
<xsl:template match="/fP/ciclos/ciclo">
  <xsl:loop select="nombre">
    <xsl:fallback>
      <xsl:for-each select="nombre">
        <xsl:value-of select="."/>
      </xsl:for-each>
    </xsl:fallback>
  </xsl:loop>
</xsl:template>
```

xsl:message

Se usa para enviar mensajes, habitualmente de error, a la salida.

Atributos optativos principales:

- `terminate`: da la posibilidad de terminar con el procesamiento (*yes*) o seguir con él (*no*).

  Posibles valores: ***no***, *yes*

Ejemplo:

Si el nombre de un ciclo es vacío, lanza un mensaje de error a la salida y termina el procesamiento:

```
<xsl:template match="/fP/ciclos/">
  <xsl:for-each select=" ciclo">
    <p>Ciclo:
```

```
      <xsl:if test="nombre=''">
        <xsl:message terminate="yes">
          Error: Nombre de ciclo inexistente
        </xsl:message>
      </xsl:if>
      <xsl:value-of select="nombre"/></p>
    </xsl:for-each>
  </xsl:template>
```

xsl:key

Declara una clave que puede ser usada a lo largo de la hoja de transformaciones mediante la función `key()`.

Atributos obligatorios:

- `name`: especifica el nombre de la clave.
- `match`: determina los nodos a los que se aplicará la clave.
- `use`: representa el valor de la clave para cada uno de los nodos.

Ejemplo:

Dado el siguiente documento XML:

```
<?xml version="1.0"?>
<estaciones>
  <estacion img="primavera.png">
    <nombre>Primavera</nombre>
    <comienzo>21 de marzo</comienzo>
  </estacion>
  ...
</estaciones>
```

Y la hoja de transformaciones:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:key name="claveEstacion" match="estacion" use="nombre"/>  ❶
  <xsl:param name="nombreEstacion" select="'Primavera'"/>  ❷

  <xsl:template match="/">
    <imagenResultado>
      <xsl:value-of
select="key('claveEstacion',$nombreEstacion)/@img"/>  ❸
    </imagenResultado>
  </xsl:template>

</xsl:stylesheet>
```

❶ Se construye una clave respecto al nombre de la estación.

❷ Se define un parámetro, nombreEstacion, y se le asigna la cadena de texto 'Primavera'.

❸ Se usa la función `key()` para extraer el atributo `img` de la `estacion` cuyo nombre valga lo que el parámetro `nombreEstacion`, es decir, *Primavera*.

La salida resultante será:

```
<?xml version="1.0"?>
<imagenResultado>primavera.png</imagenResultado>
```

6.2.6. Instrucciones de XSLT 2.0

Por citar algunas instrucciones de la última versión de XSLT, la 2.0:

- xsl:analyze-string: Permite buscar cadenas, mediante expresiones regulares, en texto sin etiquetar.
- xsl:result-document:: Permite generar multiples documentos de salida a partir de una sola hoja de transformaciones.
- xsl:sort-key: Permite definir una clave de ordenación para ser usada con la instrucción `<xsl:sort>`.

6.3. XSL-FO

XSL-FO (XSL - Formatting Objects – Objetos de formateo XSL) es un lenguaje de marcas que permite especificar el aspecto que tendrán una serie de datos (en ocasiones extraídos de un documento XML de partida) al mostrarlos en con un determinado formato de salida, habitualmente un documento PDF (Portable Document Forrmat – Formato Portable de Documento), pero también RTF (Rich Text Format – Formato de Texto Enriquecido), PostScript y otros.

Se podría escribir un libro independiente dedicado en exclusiva a este lenguaje, de manera que lo que se hará aquí será introducir los conceptos generales y mostrar algunos ejemplos.

Procesadores XSL-FO

Son programas que reciben como entrada un documento XSL-FO y generan como salida un documento con posibles diversos formatos. Los más habituales son PDF, RTF, PostScript, texto plano…

Un uso muy frecuente, que sirve como nexo con los conocimientos de XSLT recién adquiridos, es la transformación de un documento de partida XML, mediante una hoja de transformaciones XSLT, en un documento XSL-FO. A su vez, este documento podrá actuar como entrada de un procesador XSL-FO y generar como salida un documento en el formato de salida que elijamos entre los ya citados (pongamos PDF).

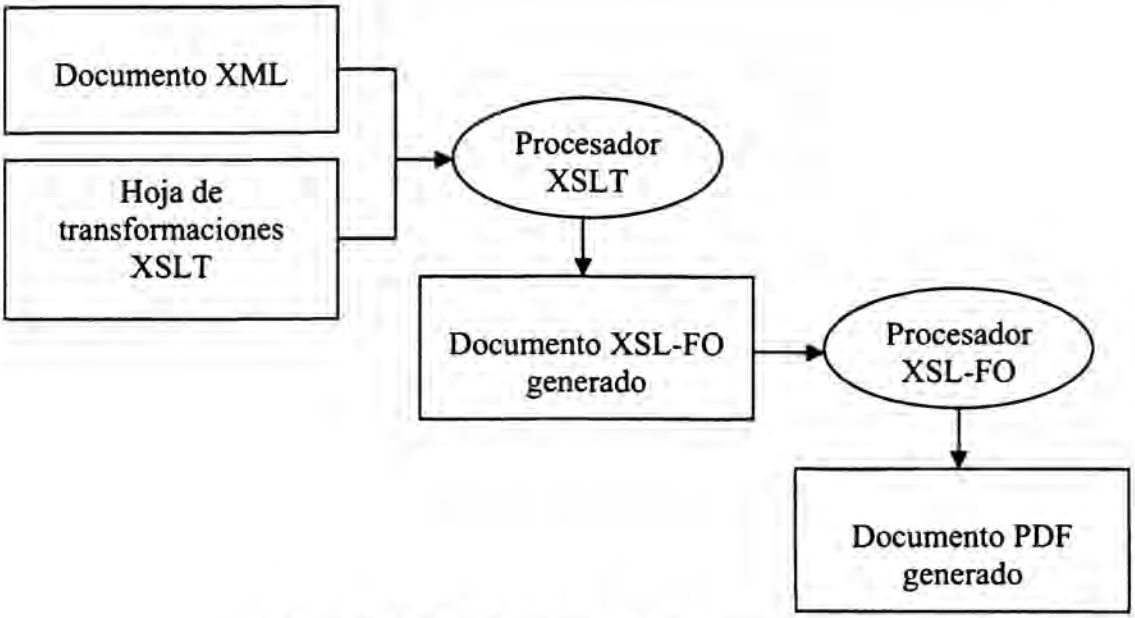


Figura 6.5: Transformación XSLT y procesamiento XSLT-FO

FOP

FOP (Formatting Objects Processor – Procesador de Objetos de Formateo) es una aplicación desarrollada en Java, dentro del proyecto de código abierto de Apache que, en su uso más frecuente, toma como entrada un documento XSL-FO y genera una salida en algún formato "visible" como PDF.

En el momento de la impresión del libro, la dirección donde encontrar FOP es http://xmlgraphics.apache.org/fop/. Aquí se puede descargar la distribución binaria (ejecutable), así como el código fuente por si se quisiera estudiar, modificar, etc. Igualmente se puede consultar la documentación de uso.

Para el fin que se persigue en este momento, la distribución binaria es suficiente. Se descargará la versión 1.0 en forma de archivo comprimido ZIP. Éste se descomprimirá y en la carpeta resultante aparecerá, entre otros archivos, el ejecutable fop.bat, que es un archivo por lotes de uso sencillo que nos oculta la invocación a la máquina virtual de java, las clases que deben estar en el *path*, etc.

Para los ejemplos del libro, se usará una sintaxis muy simplificada en función de la tarea que se quiere llevar a cabo:

- En un primer momento se usará para generar una salida en formato PDF a partir de un documento con formato FO. La sintaxis a aplicar es:

  ```
  fop -fo <doc_existente.fo> -pdf <doc_generado.pdf>
  ```

- Más adelante, lo que se pretende es generar el mismo documento de salida en formato PDF, pero el documento de entrada en formato FO se construirá dinámicamente a partir de un documento XML y una hoja de transformaciones XSLT. La sintaxis a aplicar en este caso es:

  ```
  fop -xml <doc_existente.xml> -xsl <doc_existente.xsl>
        -pdf <doc_generado.pdf>
  ```

También se podría optar por generar la salida en dos pasos:

1. Se genera el documento en formato FO a partir de un XML y una XSLT.

```
fop -xml <doc_existente.xml> -xsl <doc_existente.xsl>
    -foout <doc_generado.fo>
```

2. Se genera el documento visualizable en formato PDF a partir del recién generado documento FO.

```
fop -fo <doc_generado_paso1.fo> -pdf <doc_generado.pdf>
```

NOTA: La norma XSL-FO define un gran número de objetos de formateo con múltiples propiedades cada uno, pero la mayotía de los procesadores no es capaz de interpretarlos todos, o no es capaz de interpretar algunas de sus propiedades.

Alternativas a FOP son:

- RenderX (http://www.renderx.com/), con licencia propietaria, aunque existe una edición *Personal* para uso no comercial que se obtiene de manera gratuita.
- Antenna House (http://www.antennahouse.com/), con licencia propietaria.
- PassiveTex (http://projects.oucs.ox.ac.uk/passivetex/), con licencia de software libre.
- Altova VisionStyle, previamente citada, con licencia propietaria.

6.3.1. Objetos de formateo

Todas las etiquetas específicas de este lenguaje tienen que ver con elementos de maquetado: páginas, párrafos, bloques, tablas.

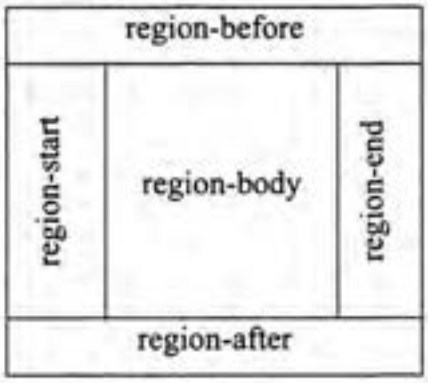
La unidad básica de diseño es el objeto de formateo (*formatting object* o también fo).

XSL-FO usa áreas rectangulares para mostrar la salida. Cualquier salida (texto, imágenes...) se mostrará en estas áreas.

Las áreas más destacadas son:

- **Páginas**: la salida de una transformación se organizará en páginas (múltiples para documentos impresos; una única de gran tamaño para navegadores). Las páginas se componen de regiones.
- **Regiones**: toda página XSL-FO contiene unas regiones determinadas:
    - región-body (cuerpo de la página)
    - región-before (cabecera de la página)
    - región-after (pie de página)
    - región-start (lateral izquierdo)
    - región-end (lateral derecho)

Las regiones contienen áreas de bloque.



- **Áreas de bloque**: define elementos de bloque de pequeño tamaño (normalmente asociadas a líneas) como párrafos, listas o tablas. Pueden contener otras áreas de bloque, pero la mayoría contienen áreas de línea.
- **Áreas de línea**: definen líneas de texto dentro de áreas e bloque. Contienen áreas secuenciales.
- **Áreas secuenciales**: definen texto en el interior de líneas (caracteres simples, viñetas, gráficos...).

Espacio de nombres fo

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
    xmlns:fo="http://www.w3.org/XSL/Format/1.0"
    result-ns="fo">
```

Estructura de una página XSL-FO

- El elemento raíz es `<fo:root>`.
- Descendientes directos de `<fo:root>` son `<fo:layout-master-set>` (patrones de las páginas del documento) y de 0 a N `<fo:page-sequence>` (páginas del documento).
- El elemento `<fo:layout-master-set>` contiene `<fo:simple-master-page>` (patrón de una página o conjunto de páginas). Cada una de ellas define el patrón para una página (márgenes, cabecera, pie...).

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <fo:layout-master-set>
    <fo:simple-page-master master-name="paginaEjemplo">
      <fo:region-body margin="1in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
```

```
    <fo:page-sequence master-reference="paginaEjemplo">
      <fo:flow flow-name="xsl-region-body">
        <fo:block>¡Hola mundo!</fo:block>
      </fo:flow>
    </fo:page-sequence>

  </fo:root>
```

Relación de XSL-FO con XSLT

Como se ha visto en el ejemplo anterior, se podría generar un documento XSL-FO sin producir ninguna transformación. La limitación viene del contenido del mismo, que tendría que ser literal, es decir, con los datos constantes.

Si lo que quisiéramos es, por ejemplo, generar una salida en formato PDF cuyos datos se extrajesen de un XML de partida, tendríamos que recurrir a una combinación de etiquetas de XSL-FO con las ya vistas de transformación XSLT.

Lo que se construirá es una hoja de transformaciones XSLT, Cuya salida será un documento XML, pero no uno cualquiera, sino uno con estructura XSL-FO con sus diferentes objetos de formateo.

Suponiendo que el mensaje de ¡Hola mundo! se encontrase en un documento XML muy sencillo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mensaje>¡Hola mundo!</mensaje>
```

El código combinado de XSL-FO y XSLT que generaría la salida vista en el ejemplo anterior será:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes" />

  <xsl:template match="/">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="paginaEjemplo">
          <fo:region-body margin="1in"/>
      </fo:simple-page-master>
    </fo:layout-master-set>

    <fo:page-sequence master-reference="paginaEjemplo">
      <fo:flow flow-name="xsl-region-body">
        <fo:block><xsl:value-of select="mensaje" /></fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
  </xsl:template>
</xsl:stylesheet>
```

Lo que se ha hecho es rodear el código XSL-FO con una hoja de transformaciones XSLT que tiene una plantilla principal donde se genera todo el código XSL-FO, extrayendo el texto del mensaje del documento XML de partida.

A partir de aquí, se podría generar un documento de salida, por ejemplo en formato PDF, cuyo aspecto sería:

Figura 6.6 Salida en formato PDF de una transformación XSL-FO

6.3.2. Objetos de formateo para la estructura de documentos

Son los objetos de formateo que permiten definir el diseño general de las páginas, las regiones de las mismas, los márgenes, cabeceras, pies, etc. El contenido propiamente dicho de las páginas vendrá definido por otros objetos de formateo que se verán más adelante.

fo:root

Es el elemento raíz de un documento XSL-FO.

Contenido::= `<fo:layout-master-set>` `<fo:declarations>` ?
`<fo:page-sequence>` +

Como atributo obligatorio se indicará el espacio de nombres de los elementos fo.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

fo:layout-master-set

Este objeto contiene todos los modelos de página (*master*), llamados `<fo:simple-page-master>`, usados en el documento.

fo:simple-page-master

Define el formato o distribución (*layout*) de una página o conjunto de páginas, sus dimensiones. Habrá uno de estos objetos por cada formato de página diferente que haya en el documento. Para un documento sencillo, un único modelo de página puede ser suficiente, pero los documentos complejos tienen habitualmente varios modelos (uno para la portada, otro para el índice, otro para las páginas pares, otro para las impares...).

Cada objeto `<fo:simple-page-master>` puede contener cinco regiones o secciones: cuerpo (*region-body*), cabecera (*region-before*), pie (*region-after*), margen izquierdo (*region-start*) y margen derecho (*region-end*). Sólo el cuerpo es imprescindible.

Estos objetos son referenciados por objetos `<fo:page-sequence-master>` o `<fo:page-sequence>`.

Propiedades más destacadas:

- `master-name`: indica el nombre del modelo (*master*).
- Similares a las existentes en los estilos CSS: margin, page-height, page-width, space-before, space-after, start-indent, etc.

fo:declarations

Permite agrupar declaraciones globales de la hoja de estilos. Es un contenedor de objetos de formateo que se usarán en el proceso de generación de la salida.

fo:page-sequence-master

Detalla en qué orden se emplean los objetos `<fo:simple-page-master>`.

Propiedades más destacadas:

- `master-name`: indica el nombre del modelo (*master*).

fo:single-page-master-reference

Hace alusión a un `<fo:simple-page-master>` que se insertará una única vez.

Propiedades más destacadas:

- `master-reference`: indica el nombre del `<fo:simple-page-master>`.

fo:repeatable-page-master-reference

Hace alusión a un `<fo:simple-page-master>` que se insertará repetidas veces.

Propiedades más destacadas:

- `master-reference`: indica el nombre del `<fo:simple-page-master>`.

fo:repeatable-page-master-alternatives

Indica la repetición de un conjunto de objetos `<fo:simple-page-master>`.

Contenido::= `<fo:conditional-page-master-reference>` +

Propiedades más destacadas:

- `maximum-repeats`: indica el número de repeticiones máximo que se pueden dar.

fo:conditional-page-master-reference

Es un modelo de página condicional y las condiciones que se tienen que dar para que se aplique. Cuando hay varios modelos que cumplen una misma condición (por ejemplo, ser la primera página y ser página impar), se aplica el primero en aparecer.

Propiedades más destacadas:

- `master-reference`: indica el nombre del `<fo:simple-page-master>`.
- `page-position`: indica el ordinal de la página a la que se aplicará el modelo. Un posible valor es first.
- `odd-or-even`: indica si el modelo se aplicará a páginas pares (*even*) o impares (*odd*).

Ejemplo:

Se definen tres modelos que se aplicarán según si la página es la primera o si es par o impar.

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="pagina_primera">
    <fo:region-body margin="1in" border="thin solid silver"
                    padding="6pt"/>
  </fo:simple-page-master>
  <fo:simple-page-master master-name="pagina_impar">
    <fo:region-body margin="1in" border-right="medium gray ridge"
                    padding-right="6pt"/>
  </fo:simple-page-master>
  <fo:simple-page-master master-name="pagina_par">
    <fo:region-body margin="1in" border-left="medium gray ridge"
                    padding-left="6pt"/>
  </fo:simple-page-master>

  <fo:page-sequence-master master-name="secuencia_personalizada">
    <fo:repeatable-page-master-alternatives>
      <fo:conditional-page-master-reference page-position="first"
                      master-reference="pagina_primera"/>
      <fo:conditional-page-master-reference odd-or-even="odd"
                      master-reference="pagina_impar"/>
      <fo:conditional-page-master-reference odd-or-even="even"
                      master-reference="pagina_par"/>
    </fo:repeatable-page-master-alternatives>
  </fo:page-sequence-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="secuencia_personalizada">
  ...
</fo:page-sequence>
```

AVISO: En algunos de los ejemplos siguientes se simplificará el código de los documentos XSL-FO, omitiendo los elementos generales como la declaración del documento XML, el elemento raíz `<fo:root>` y algunos más. Deben ser añadidos para el correcto tratamiento por parte de un procesador.

6.3.3. Objetos de formateo para el contenido de páginas

Son objetos que se usan para determinar el contenido que tendrá cada página del documento.

fo:page-sequence

Es un contenedor de objetos página de salida. Se definirá un objeto `<fo:page-sequence>` por cada página diferente que se quiera generar en el documento de salida. Los descendientes de cada uno de estos objetos proporcionarán el contenido de dichas páginas.

Cada objeto `<fo:page-sequence>` está asociado (referencia) a un objeto `<fo:page-sequence-master>` o `<fo:simple-page-master>`.

Contenido::= `<fo:title>` ? `<fo:static-content>` * `<fo:flow>`

Propiedades más destacadas:

- `master-reference`: indica el nombre del objeto `<fo:page-sequence-master>` o `<fo:simple-page-master>` al que se encuentra asociado.

fo:title

Es un objeto que especifica el título para una `<fo-page-sequence>`.

fo:static-content

Contiene elementos estáticos, como cabeceras o pies de página, que se repetirán en muchas páginas. Los diferentes contenidos estáticos que pueda haber en una `<fo:page-sequence>` aparecen antes que el `<fo:flow>`.

Propiedades más destacadas:

- `flow-name`: indica dónde se ubicará el contenido del `<fo:static-content>`.

  Valores reservados: *xsl-region-body*, *xsl-region-before*, *xsl-region-after*, *xsl-region-start*, *xsl-region-end*, *xsl-before-float-separator* y *xsl-footnote-separator*.

fo:flow

Contiene los elementos que se imprimirán en una página.

Propiedades más destacadas:

- `flow-name`: indica dónde se ubicarán los elementos del flujo.

  Valores reservados: *xsl-region-body*, *xsl-region-before*, *xsl-region-after*, *xsl-region-start*, *xsl-region-end*, *xsl-before-float-separator* y *xsl-footnote-separator*.

fo:block

Es un contenedor a nivel de bloque. Corresponde, aproximadamente, a un `<div>` de HTML. Se utiliza para formatear párrafos de texto, títulos, etc.

Propiedades más destacadas:

- Similares a las de estilos CSS: background-color, font-family, font-weight, margin, padding...

fo:inline

Es un contenedor que permite alojar objetos que se distribuyan secuencialmente, sin salto de línea. No puede contener elementos a nivel de bloque. Se utiliza en ocasiones para aplicar atributos de formateo a distintos objetos. Se asemeja a `<span>` en HTML.

6.3.4. Objetos de formateo para generar listas

Existen una serie de objetos de formateo que se utilizan para generar listas. Se usan de manera conjunta, así que se estudiarán como un bloque.

fo:list-block

Objeto usado para definir una lista.

Contenido::= `<fo:list-item>` +

fo:list-item

Objeto usado para definir cada elemento de una lista.

Contenido::= `<fo:list-item-label>` `<fo:list-item-body>`

fo:list-item-label

Objeto que contiene la etiqueta usada como marcador de lista (un número, un boliche, un guión...). Esta etiqueta se encuadrará, a su vez, dentro de un objeto `<fo:block>`.

Contenido::= (`<fo:block>` | `<fo:block-container>` | `<fo:table-and-caption>` | `<fo:table>` | `<fo:list-block>` | `<fo:list-item>`)+

fo:list-item-body

Objeto que contiene el cuerpo o contenido de cada elemento de lista (habitualmente, uno o más objetos `<fo:block>`).

Contenido::= (`<fo:block>` | `<fo:block-container>` | `<fo:table-and-caption>` | `<fo:table>` | `<fo:list-block>` | `<fo:list-item>`)+

Ejemplo:

Objetos de formateo relativos a listas:

```
<fo:list-block provisional-distance-between-starts="18pt"
               provisional-label-separation="3pt">

  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>&#x2022;</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>Alfa</fo:block>
    </fo:list-item-body>
  </fo:list-item>

  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>&#x2022;</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>Beta</fo:block>
    </fo:list-item-body>
  </fo:list-item>

  <fo:list-item>
    <fo:list-item-label end-indent="label-end()">
      <fo:block>&#x2022;</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>Gamma</fo:block>
    </fo:list-item-body>
  </fo:list-item>

</fo:list-block>
```

6.3.5. Objetos de formateo para generar tablas

De igual manera que sucedía con la generación de listas, existen una serie de objetos de formateo que se utilizan para generar tablas. Se usan de manera conjunta, así que se estudiarán como un bloque.

fo:table-and-caption

Es un objeto contenedor de todos los demás objetos relativos a la construcción de tablas, que son: `<fo:table>`, `<fo:table-caption>`, `<fo:table-column>`, `<fo:table-header>`, `<fo:table-footer>`, `<fo:table-body>`, `<fo:table-row>` y `<fo:table-cell>`.

Contenido::= `<fo:table-caption>`? `<fo:table-caption>`

fo:table-caption

Este objeto contiene el título de la tabla y se aparece como descendiente de `<fo:table-and-caption>`.

fo:table

Permite definir una tabla.

Contenido::= `<fo:table-column>*` `<fo:table-header>?` `<fo:table-body>`
`<fo:table-footer>?`

fo:table-column

Permite indicar el formato de las columnas de la tabla.

Propiedades más destacadas:

- `column-width`: indica el ancho de la columna

fo:table-header

Define la cabecera de la tabla.

Contenido::= (`<fo:table-row>+` | `<fo:table-cell>+`)

fo:table-footer

Define el pie de la tabla.

Contenido::= (`<fo:table-row>+` | `<fo:table-cell>+`)

fo:table-body

Es un objeto contenedor de las filas (`<fo:table-row>`) y celdas (`<fo:table-cell>`) de la tabla. En ese sentido, su estructura se asemeja a las tablas de HTML.

fo:table-row

Es un objeto que define una fila de una tabla. Su etiqueta equivalente en HTML es `<tr>`.

Contenido::= `<fo:table-cell>+`

fo:table-cell

Es un objeto que representa una celda de una tabla. Su etiqueta equivalente en HTML es `<td>`.

Contenido::= (`<fo:block>` | `<fo:block-container>` | `<fo:table-and-caption>` |
`<fo:table>` | `<fo:list-block>`) +

Ejemplo:

Una tabla con todos los objetos de formateo asociados:

```
<!-- Contenedor de la tabla y su título -->
<fo:table-and-caption>

  <!-- Título de la tabla -->
  <fo:table-caption>
    <fo:block>Miembros de la familia</fo:block>
  </fo:table-caption>
  <!-- Contenedor de la tabla -->
  <fo:table>
    <fo:table-column column-width="100mm"/>
    <fo:table-column column-width="100mm"/>

    <!-- Cabecera de la tabla -->
    <fo:table-header>
      <fo:table-cell>
        <fo:block font-weight="bold">Nombre</fo:block>
      </fo:table-cell>
      <fo:table-cell>
        <fo:block font-weight="bold">Relación</fo:block>
      </fo:table-cell>
    </fo:table-header>

    <!-- Cuerpo de la tabla -->
    <fo:table-body>
      <fo:table-row>
        <fo:table-cell>
          <fo:block>José</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block>Padre</fo:block>
        </fo:table-cell>
      </fo:table-row>
      <fo:table-row>
        <fo:table-cell>
          <fo:block>Carmen</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block>Madre</fo:block>
        </fo:table-cell>
      </fo:table-row>
      <fo:table-row>
        <fo:table-cell>
          <fo:block>Pilar</fo:block>
        </fo:table-cell>
        <fo:table-cell>
          <fo:block>Hija</fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-body>
  </fo:table>

</fo:table-and-caption>
```

6.3.6. Objetos de formateo para generar enlaces, imágenes

Existen otros muchos objetos de formateo para generar diferentes elementos gráficos. Aquí se verán unos cuantos ejemplos.

fo:basic-link

Corresponde a un enlace (equivalente a `<a>` en HTML). Puede ser a una ubicación interna al documento (que debe estar etiquetada con un id) o externa (una URL).

Propiedades más destacadas:

- `internal-destination`: identificador (atributo `id`) del objeto del documento a donde conduce el enlace
- `external-destination`: URI de la página a la que conduce el enlace.

Ejemplo:

Se muestran dos ejemplos de enlaces: uno interno a una imagen contenida en el documento, cuyo atributo `id` debe ser *foto*, y uno externo a la página de Goodle.

```
<fo:basic-link internal-destination="foto"
    text-decoration="underline" color="blue">Enlace a una foto
</fo:basic-link>

<fo:basic-link external-destination="url('http://www.google.com/')"
    text-decoration="underline" color="blue">Ir a Google
</fo:basic-link>
```

fo:external-graphic

Permite insertar una imagen. Se comporta de manera semejante al elemento `<img>` de HTML.

Propiedades más destacadas:

- Similares a los del elemento `<img>` de HTML: src, width, height

Ejemplo:

Se integra en el document o una imagen externa de nombre *paisaje.gif*. La ubicación del archivo de la imagen debe ser la adecuada para que el procesador la localice en el sistema de archivos.

```
<fo:block>
  <fo:external-graphic src="url('paisaje.gif')"/>
</fo:block>
```

fo:leader

Permite crear una línea horizontal. Se comporta de manera semejante al elemento `<hr>` de HTML.

Propiedades más destacadas:

- `leder-length`: longitud de la línea
- `leader-patten`: patrón (aspecto) de la línea.

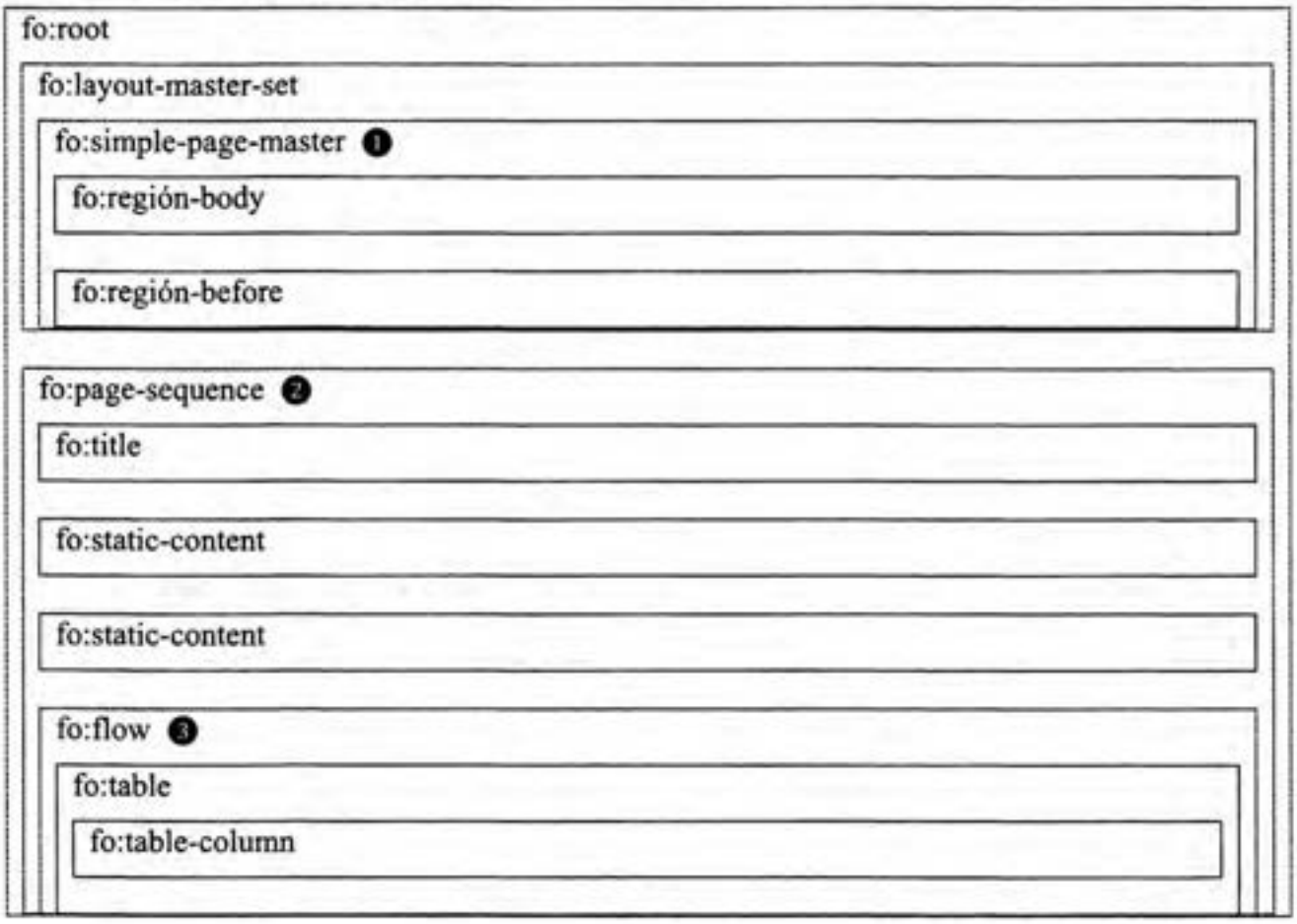
Ejemplo:

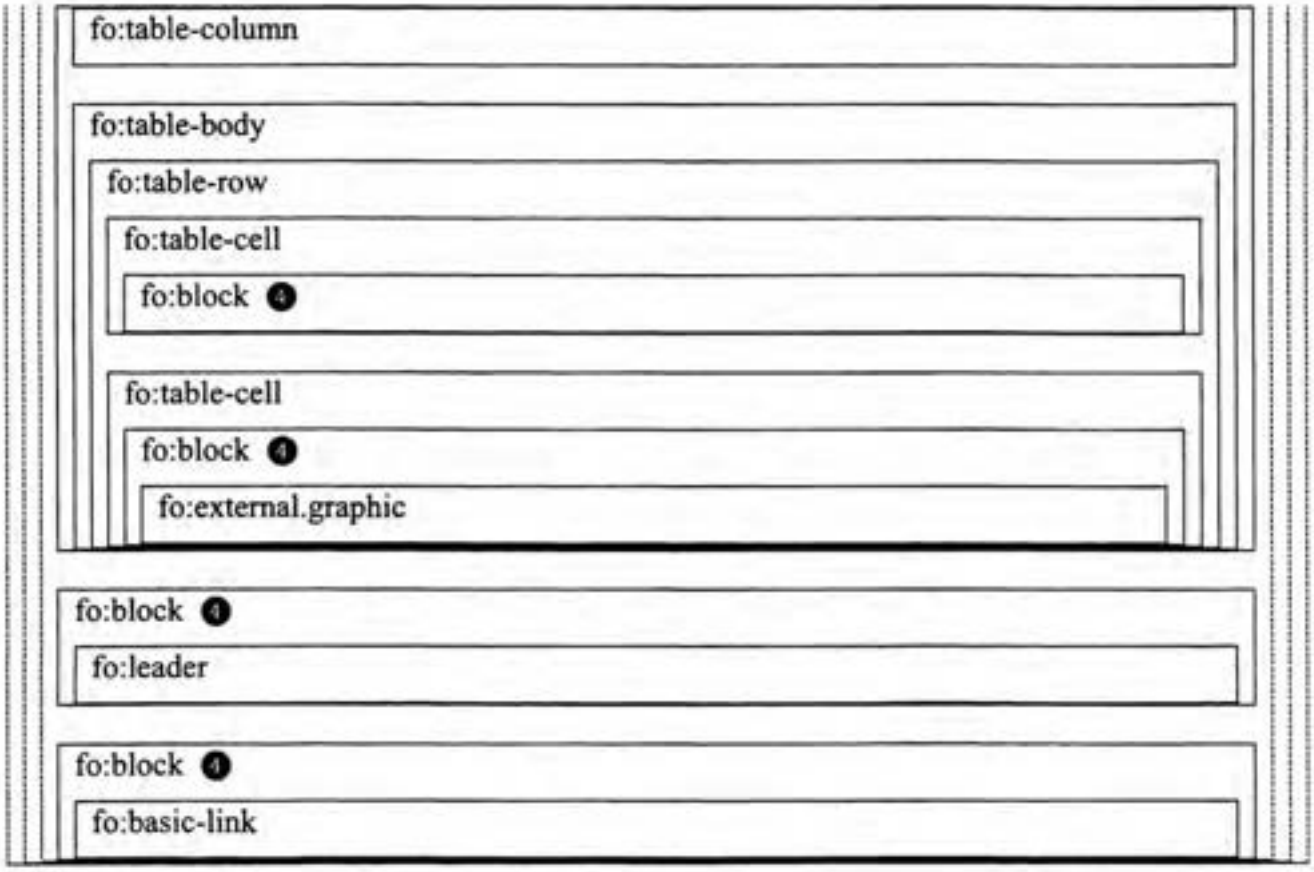
Se crea una línea con un carácter ✱ en medio y luego otra línea a continuación.

```
<fo:block text-align="center">
  <fo:leader leader-length="2in" leader-pattern="rule" />
  <fo:inline color="#E00000">&#x274B;</fo:inline>
  <fo:leader leader-length="2in" leader-pattern="rule" />
</fo:block>
```

❗ **NOTA:** Como se ha visto en los diferentes ejemplos, los objetos de formateo disponen de propiedades de aspecto muy parecidas (muchas se llaman exactamente igual) a las de los estilos CSS. La lista de propiedades es enorme, por lo que no se citará aquí. Se puede consultar en múltiples fuentes en internet, como por ejemplo en:
http://www.cafeconleche.org/books/bible2/chapters/ch18.html

Estructura página ejemplo





❶ En la `<fo:simple-page-master>`, lo primero se declara la `<fo:región-body>`.

❷ En una `<fo:page-sequence>`, aparece opcionalmente un `<fo:title>`. Sino, lo primero en aparecer son los `<fo:static-content>`.

❸ Luego viene el `<fo:flow>`. Todo contenido que va dentro del `<fo:flow>` debe ir, en última instancia, dentro de un `<fo:block>`.

❹ Cada `<fo:block>` genera un salto de línea, como un `<div>` en HTML. Para introducir contenido secuencial (*inline* o en línea) se utiliza `<fo:inline>` que es equivalente a `<span>` en HTML.

Ejercicios propuestos

1. Escribe una hoja de transformaciones que, a partir del documento formacionProfesional.xml anteriormente visto, **genere una salida en formato de texto** con la siguiente información para cada módulo:

   Módulo: Programación (8 horas semanales)
   Módulo: Lenguaje de marcas (4 horas semanales)

2. Genera una hoja de transformaciones XSLT que convierta el documento XML del currículo, generado en ejercicios anteriores, en un documento HTML con el siguiente aspecto.

José Andrés Sánchez García

Datos personales

Fecha de nacimiento: 07/02/1979
Lugar de nacimiento: Zaragoza
Estado civil: Casado

Formación

Descripción	Lugar	Periodo
Técnico superior en radiotelecomunicación	Escuela Superior de Electrónica	De 01/10/1995 a 30/06/1997
Master en dirección de empresas	Escuela de Negocios Metropolitana	De 10/01/2000 a 30/06/2001

Idiomas

- Castellano (Materna)
- Inglés
  a. EO: 3
  b. CO: 2
  c. EE: 3
  d. CE: 4
- Francés
  a. EO: 2
  b. CO: 2
  c. EE: 3
  d. CE: 3

Experiencia laboral

Lugar	Puesto	Periodo
Policía	Inspector	De 14/03/2004 a
Aeropuerto	Radiotelegrafista	De 12/09/2001 a 07/03/2004

Otras informaciones

Licencias de conducción -> B1 ☑
Vehículo propio ☑
Descripción personal
Tímido, perfeccionista, polifacético

Fecha actual: 01/06/2012

Recomendaciones:

a. Escribe primero el documento HTML "a mano", afinando el aspecto con hojas de estilo CSS. A partir de ahí, construye la hoja de transformaciones.

b. Crea una plantilla para la raíz del documento XML (`match="/"`) y desde ella invocar a plantillas para "dibujar" cada una de las zonas de la página HTML, es decir, crea una plantilla para cada uno de los nodos del árbol XML del cual se extrae información. Ejemplo: una plantilla para los datos generales de la persona, una platilla para la formación, otra para la experiencia laboral...

3. Genera dos hojas de transformaciones que conviertan el siguiente documento XML *museos.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<museos>
  <museo>
    <nombre>Louvre</nombre>
    <ciudad>París</ciudad>
    <pais>Francia</pais>
  </museo>

  <museo>
    <nombre>Británico</nombre>
    <ciudad>Londres</ciudad>
    <pais>Reino Unido</pais>
  </museo>

  <contenidoRelleno descripcion="Letras griegas">
    <elementos>
      <elemento>Alfa</elemento>
      <elemento>Beta</elemento>
      <elemento>Gamma</elemento>
      <elemento>Delta</elemento>
    </elementos>
  </contenidoRelleno>

  <museo>
    <nombre>Hermitage</nombre>
    <ciudad>San Petersburgo</ciudad>
    <pais>Rusia</pais>
  </museo>

  <museo>
    <nombre>Pérgamo</nombre>
    <ciudad>Berlín</ciudad>
    <pais>Alemania</pais>
  </museo>

  <museo>
    <nombre>Prado</nombre>
    <ciudad>Madrid</ciudad>
    <pais>España</pais>
  </museo>

</museos>
```

... en cada uno de los dos siguientes documentos XML de salida:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<museos>
  <contenidoRelleno descripcion="Letras griegas">
    <elementos>
      <elemento>Alfa</elemento>
      <elemento>Beta</elemento>
      <elemento>Gamma</elemento>
      <elemento>Delta</elemento>
    </elementos>
  </contenidoRelleno>
  <museo ubicacion="París (Francia)">Louvre</museo>
  <museo ubicacion="Londres (Reino Unido)">Británico</museo>
  <museo ubicacion="San Petersburgo (Rusia)">Hermitage</museo>
  <museo ubicacion="Berlín (Alemania)">Pérgamo</museo>
  <museo ubicacion="Madrid (España)">Prado</museo>
</museos>
```

...y:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<paises
  <pais nombre="Francia">
    <museo nombre="Louvre" ciudad="París"/>
  </pais>
  <pais nombre="Reino Unido">
    <museo nombre="Británico" ciudad="Londres"/>
  </pais>
  <pais nombre="Rusia">
    <museo nombre="Hermitage" ciudad="San Petersburgo"/>
  </pais>
  <pais nombre="Alemania">
    <museo nombre="Pérgamo" ciudad="Berlín"/>
  </pais>
  <pais nombre="España">
    <museo nombre="Prado" ciudad="Madrid"/>
  </pais>
</paises>
```

4. Crea un documento XML, con estructura libre, que contenga información relativa a un recibo de banco. Los datos mínimos que deben aparecer son:
   - Del banco: nombre, sucursal donde está la cuenta bancaria, logo del banco
   - De la cuenta bancaria: número (20 dígitos), moneda (por defecto euros)
   - Del titular de la cuenta: nombre
   - Del recibo en sí: fecha, cantidad total de la factura
     - para cada concepto (puede incluir varios): descripción del mismo y cantidad,

Se muestra un ejemplo de cómo podría quedar el recibo:

Banco Seguro Página 1 de 1

Fecha: 01/06/2012 Oficina: 4321 (Madrid) Moneda: EURO Nº recibo: r03_2012_06_01 Referencia: a1b2c3d4e5f6g7h8

RECIBO		
Entidad emisora: Ayto. de Madrid	NIF entidad emisora: B87654321	Titular recibo: María Blanco Iglesias
Movimientos		
• Impuesto de bienes inmuebles: 208,34 euros • Tasa de recogida de residuos: 26,83 euros		
Si desea devolver el recibo contacte con su oficina, con el servicio de atención telefónica o con la página web www.bancoseguro.com		
Importe: 208,34 euros	Nº cuenta: 1234.4321.55.1000220001	Fecha valor: 04/06/2012

Titular: María Blanco Iglesia Datos entidad Emisora: nif: B87654321 Referencia: F123456 - RM Madrid

5. Crea una plantilla XSLT que genere una salida HTML de diseño libre para el documento XML ya visto *formacionProfesional.xml*. Se recomienda diseñar primero "a mano" la página HTML que se quiere obtener, con sus estilos aparte, y luego generarla con la hoja de transformaciones.
6. Genera una hoja de transformaciones XSL-FO que convierta el documento XML *formacionProfesional.xml* en un documento con formato PDF cuyo aspecto se asemeje al del documento HTML obtenido en el ejercicio anterior.

CAPÍTULO 7

Sindicación de contenidos. RSS

CONTENIDO

Introducción a RSS
Estructura de un documento RSS
Elementos principales de un RSS
Generación de RSS
Validación del archivo RSS
Publicación del archivo RSS
Ejercicios propuestos

7.1. Introducción a RSS

RSS son las siglas de Really Simple Syndication – Sindicación Realmente Simple, y se trata de una tecnología que sirve para compartir o distribuir información en la Web. Cuando un usuario está interesado en un determinado tema, quiere recibir información de forma continua y actualizada, para lo cual, se suscribe a la fuente de los contenidos de dicho tema. De esta forma, el usuario no tiene que preocuparse de consultar periódicamente la fuente para comprobar si hay información nueva.

Inicialmente era necesario utilizar un software específico diseñado para leer estos contenidos RSS, lo que se conoce como feed reader, agregador de noticias o lector RSS, pero actualmente los navegadores incorporan el software necesario para leer los RSS.

La información de un sitio Web puede ser compartida de varias formas, una manera sencilla es como hemos descrito antes, mediante suscripción a la fuente con un agregador de noticias. Pero también se puede compartir insertando la información en otros sitios Web, de esta forma el receptor de las noticias se convierte a su vez en emisor, a esto se le conoce como **redifusión Web**. Este nuevo servicio es uno de los pilares básicos de lo que se ha llamado Web 2.0, formada por ese conjunto de aplicaciones Web que mejoran la difusión de información como las redes sociales, wikis, blogs, etc.

Resumiendo, RSS es una técnica para difundir de forma más eficaz los contenidos de un sitio Web. Pero también es un lenguaje derivado de XML, para construir los archivos que guardan el contenido a difundir. Sin embargo, no es el único lenguaje existente con este propósito, posteriormente apareció Atom, también derivado de XML pero con algunas ventajas sobre su predecesor. Por tanto, tenemos dos lenguajes, es decir, dos formatos de archivo que sirven para almacenar los contenidos a distribuir. No debemos confundir los dos significados de RSS, el primero hace referencia al concepto general de sindicación o redifusión Web y el segundo a un formato en particular de archivo que contiene la información a difundir.

Ventajas de la sindicación de contenidos:

- Los usuarios no necesitan comprobar si la información ha sido actualizada en los sitios donde se encuentran los contenidos de su interés (por ejemplo: noticias, próximos eventos o nuevos productos a la venta).

- El formato de los datos es texto plano, por tanto, ligero y rápido a la hora de ser transmitido, esto lo hace idóneo para dispositivos móviles.

- Mediante la sindicación se puede filtrar la información que nos interesa de cada sitio y no perder tiempo con el resto de información.

- Protege la cuenta de correo puesto que no es necesario utilizarla, así evitaremos correo no deseado.

Figura 7.1: Símbolo de RSS.

7.2. Estructura de un documento RSS

Como siempre empezamos con el prólogo:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
```

La primera línea es la declaración XML, que define la versión de XML y la codificación de caracteres utilizados. En este caso, el documento cumple con la especificación 1.0 de XML y utiliza el conjunto de caracteres Unicode.

La siguiente línea es la declaración de tipo de documento, que identifica el lenguaje derivado de XML que estamos usando, en este caso se trata de un documento RSS versión 2.0.

Si queremos aumentar el número de etiquetas disponibles debemos incluir espacios de nombres, algunas posibilidades son:

```
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<rss version="2.0" xmlns:g="http://base.google.com/ns/1.0">
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
```

También se trata de la raíz del documento, por ello, el contenido debe estar comprendido entre las etiquetas **<rss>**contenido**</rss>**.

Después del prólogo viene el contenido del documento.

La siguiente línea contiene el elemento **<channel>** que describe la fuente o canal RSS.

El elemento **<channel>** tiene tres elementos obligatorios: **<title>**, **<link>** y **<description>**

El elemento **<channel>** puede tener uno o más elementos **<item>** y cada elemento **<item>** define un artículo en el canal RSS.

Por último, las dos últimas líneas consisten en cerrar la etiqueta **<channel>** y la raíz **<rss>**.

Ejemplo:

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
  <channel>
    <title>Bienvenidos a mipagina</title>
    <link>http://www.juanmacr.es</link>
    <description>Ayuda al estudiante de informática</description>
    <item>
      <title>ASIR</title>
      <link>http://www.juanmacr.es/rss/asir.rss</link>
      <description>Ciclo de administración</description>
    </item>
  </channel>
</rss>
```

7.3. Elementos principales de un RSS

7.3.1. <channel>

Como se mencionó anteriormente, el elemento **<channel>** describe la fuente RSS y tiene tres elementos obligatorios:

<title>: Define el nombre del canal

<link>: Define el hipervínculo para el canal

<description>: Describe el contenido del canal

El elemento **<channel>** puede contener uno o más elementos **<item>**.

Los elementos opcionales en **<channel>** son:

Elemento	Descripción
<category>	Define una o más categorías para el canal
<cloud>	Permite ser informado inmediatamente de los cambios en el canal
<copyright>	Notifica sobre el contenido con derechos de autor
<docs>	Indica una dirección para la documentación del formato utilizado
<generator>	Especifica el programa utilizado para generar el canal
<image>	Presenta una imagen cuando los agregadores muestren un canal
<language>	Especifica el idioma en que está escrito el canal
<lastBuildDate>	Define la fecha de la última modificación del contenido del canal
<link>	Define el hipervínculo para el canal
<pubDate>	Define la última fecha de publicación en el canal
<rating>	La valoración PICS del canal
<skipDays> <skipHours>	Especifica los días/horas durante los cuales los agregadores deben omitir la actualización del canal
<textInput>	Especifica un campo de entrada de texto que aparece con el canal
<ttl>	Especifica el tiempo en minutos, que el canal puede permanecer en la caché, antes de actualizarse desde la fuente ("time to live")
<webMaster>	Define la dirección e-mail del webmaster del canal

Tabla 7.1: Elementos opcionales de <channel>

Vamos a explicar algunos de los más conocidos:

· El elemento **<category>**

Permite a los agregadores de RSS agrupar sitios basándose en la categoría. Se puede indicar jerarquía en las categorías usando barras.

Ejemplo:

<category>Noticias/economía/rescate**</category>**

· El elemento **<copyright>**

Permite identificar el material con derechos de autor.

Ejemplo:

<**copyright**>Garceta - 2012. Todos los derechos reservados</**copyright**>

· El elemento <**image**>

Permite que se muestre una imagen cuando los agregadores presentan un canal.

El elemento <**image**> tiene tres elementos secundarios obligatorios:

<url>: Indica la URL de la imagen

<title>: Indica el texto que se mostrará si la imagen no se pudo cargar

<link>: Indica el hipervínculo a la página web que ofrece el canal

Ejemplo:

```
<image>
  <url>http://www.mipagina.es/imagenes/logo.jpg</url>
  <title>miempresa</title>
  <link>http://www.mipagina.es</link>
</image>
```

· El elemento <**language**>

Permite especificar el lenguaje utilizado para escribir el documento RSS y agrupar los sitios basándose en el lenguaje.

Ejemplo:

<language>es-es**</language>**

· El elemento **<generator>**

Es muy común cuando el canal es generado automáticamente por alguna herramienta.

Ejemplo:

<generator>Joomla! - Open Source Content Management**</generator>**

7.3.2. <item>

Como se mencionó antes, cada elemento **<item>** define un artículo en el canal RSS.

El elemento **<item>** contiene tres elementos necesarios:

<title>: Define el título del artículo

<link>: Define el hipervínculo al artículo

<description>: Describe el artículo

Los elementos opcionales de **<item>** son:

Elemento	Descripción
<author>	Especifica el email del autor del artículo
<category>	Define la categoría/s a las que pertenece el artículo
<comments>	Permite enlazar a los comentarios sobre ese tema
<enclosure>	Permite incluir un archivo multimedia
<guid>	Define un identificador único para el artículo
<pubDate>	Define la fecha de la última publicación para el artículo
<source>	Especifica una fuente para el artículo mediante un link

Tabla 7.2: Elementos opcionales en <item>

· El elemento **<comments>**

Permite un elemento para vincular a los comentarios sobre ese tema.

Ejemplo:

<comments>http://www.viva_pilar.com/comentarios**</comments>**

· El elemento **<enclosure>**

Permite incluir un archivo multimedia en el artículo.

Tiene tres atributos obligatorios:

- url: Define la URI del archivo
- longitud: Define el tamaño en bytes del archivo
- type: Define el tipo de archivo multimedia

Ejemplo:

```
<enclosure url="http://www.mipagina/rss/noticias.mp3"
length="1200" type="audio/mpeg" />
```

7.4. Generación de RSS

Hay varias formas de crear el archivo para compartir información: si creamos nuestro sitio con un gestor de contenidos tipo Drupal, Wordpress o Joomla!, es el propio gestor quien se encarga de crear y mantener los archivos de sindicación en formato rss y atom, otra opción es escribirlos nosotros utilizando los elementos vistos anteriormente y por último podemos utilizar alguna herramienta para generación automática de feeds.

Por ejemplo, la herramienta RSS Builder es gratuita y sencilla de manejar.

Tenemos que rellenar los campos relativos al Feed o canal:

- Title: nombre del canal

- URL: dirección del sitio Web al que hace referencia el canal

- Copyright, lenguaje y descripción del canal

Se puede asociar una imagen al Canal para que aparezca en el navegador o en el lector.

Se puede asociar una hoja de estilos al canal.

"Topic" es la sección para los `<item>` o artículos.

Figura 7.2: Generación de RSS

Por cada artículo, debemos especificar el nombre, enlace a la página donde se encuentra el artículo, un identificador GUID, la categoría/as a las que pertenece el artículo y una descripción del contenido, que puede ser también el comienzo del artículo.

Podemos vincular el artículo con una página de comentarios y especificar el email del autor.

El programa establece automáticamente la fecha de publicación mediante la fecha del sistema en formato GMT.

La pestaña Enclosure permite añadir archivos multimedia relacionados con el artículo.

Guardamos y el archivo obtenido sería:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <generator>RSS Builder by B!Soft</generator>
    <title>FP</title>
    <link>http://www.juanmacr.es</link>
    <description>Ciclos Formativos de Informática</description>
    <language>es</language>
    <copyright>CC BY-NC-SA</copyright>
    <image>
      <title>FP</title>
      <link>http://www.juanmacr.es</link>
      <url>http://www.juanmacr.es/imagenes/jmcr.jpg</url>
    </image>
    <item>
      <title>dam</title>
      <pubDate>Thu, 28 Jun 2012 12:07:45 +0200</pubDate>
      <link>http://www.juanmacr.es/dam</link>
      <guid isPermaLink="true">http://www.juanmacr.es/dam</guid>
      <category>Educación/informática</category>
      <description><![CDATA[Técnico Superior en Desarrollo de
Aplicaciones Multiplataforma.
Este profesional será capaz de:
Desarrollar, implantar, documentar y mantener aplicaciones informáticas
multiplataforma, utilizando tecnologías y entornos de desarrollo
específicos, garantizando el acceso a los datos de forma segura y
cumpliendo los criterios de usabilidad y calidad exigidas en los
estándares establecidos.]]></description>
    </item>
    <item>
      <title>asir</title>
      <pubDate>Thu, 28 Jun 2012 11:16:42 +0200</pubDate>
      <link>http://www.juanmacr.es/asir</link>
      <guid isPermaLink="true">http://www.juanmacr.es/asir</guid>
      <category>Educación/informática</category>
      <description><![CDATA[Técnico Superior en Administración de
      sistemas informáticos en red.]]></description>
    </item>
  </channel>
</rss>
```

7.5. Validación del archivo RSS

Una vez construido nuestro archivo RSS podemos validarlo como siempre con la herramienta de validación del W3C.

Para validar feeds escritos en Atom o RSS la dirección es http://validator.w3.org/appc/

Como en temas anteriores podemos elegir entre validación por URI, si hemos subido el archivo al servidor, o en caso contrario validación pegando el archivo directamente.

En caso de no contener errores, se obtiene un resultado como el de la siguiente figura:

Figura 7.3: RSS Válido

Pueden aparecer al final unas recomendaciones a modo de avisos después de obtener un feed válido. Es conveniente seguirlas pero no es imprescindible para el funcionamiento correcto de nuestro archivo.

7.6. Publicación del archivo RSS

Una vez escrito y validado nuestro archivo hay que subirlo al servidor Web.

Ahora debemos hacer 3 cosas:

1. Insertar un enlace en la página de inicio que apunte al archivo RSS para poder consultarlo a través del navegador.

2. Insertar un elemento `<link>` para que los lectores o agregadores RSS puedan encontrar nuestro archivo RSS y leerlo, es decir, suscribirse a nuestro feed.

3. Enviar la dirección URI del archivo RSS a sitios Web, llamados directorios RSS, que se dedican a catalogar y almacenar feeds para que los buscadores los visiten.

Empezamos por incluir el enlace, para lo cual usamos el icono estándar de sindicación de contenidos:

Figura 7.4: Enlace RSS

Al pinchar en el enlace el navegador muestra los artículos:

Juanmacr - Ciclos

Ciclos Formativos de Informática

Ciclo Superior DAM
jueves, 28 de junio de 2012 12:07

Técnico Superior en Desarrollo de Aplicaciones Multiplataforma. Este profesional será capaz de: Desarrollar, implantar, documentar y mantener aplicaciones informáticas multiplataforma, utilizando tecnologías y entornos de desarrollo específicos, garantizando el acceso a los datos de forma segura y cumpliendo los criterios de usabilidad y calidad exigidas en los estándares establecidos.

Ciclo Superior ASIR
jueves, 28 de junio de 2012 11:16

Técnico Superior en Administración de sistemas informáticos en red. Este profesional será capaz de: Configurar, administrar y mantener sistemas informáticos, garantizando la funcionalidad, la integridad de los recursos y servicios del sistema, con la calidad exigida y cumpliendo la reglamentación vigente. Ejercerá su actividad en el área informática de entidades que dispongan de sistemas para la gestión de datos e infraestructura de redes, [intranet, internet y/o extranet]. Entre los puestos de trabajo que puede desempeñar destacan los siguientes: técnico en administración de sistemas, responsable de informática, técnico en servicios de internet, técnico en servicios de mensajería electrónica, personal de apoyo y soporte técnico, técnico en teleasistencia, técnico en administración de bases de datos, técnico en redes, supervisor de sistemas o técnico en entornos web.

Ciclo Superior DAW
jueves, 28 de junio de 2012 12:07

Técnico Superior en Desarrollo de Aplicaciones Web.

Ciclo Medio SMR
jueves, 28 de junio de 2012 12:07

Técnico en Sistemas Microinformáticos y Redes.

Módulo: Lenguajes de Marcas
jueves, 28 de junio de 2012 12:07

Lenguaje de Marcas y Sistemas de Gestión de Información.

Figura 7.5: Vista del canal en Firefox

Ahora viene lo más importante, la suscripción a nuestro feed, para lo cual se necesita un lector RSS. Los lectores pueden ser programas específicos para agregar contenidos, por ejemplo, Feedreader o aplicaciones de un buscador, por ejemplo, Google Reader.

Lo primero es añadir un enlace en nuestro sitio para localizar el archivo RSS:

```
<link href="archivo.rss" rel="alternate"
      type="application/rss+xml" title="RSS 2.0" />
```

Ahora abrimos nuestro lector RSS y agregamos la dirección del archivo RSS, en Feedreader se hace mediante el botón Nuevo Canal:

Figura 7.6: Agregar un canal

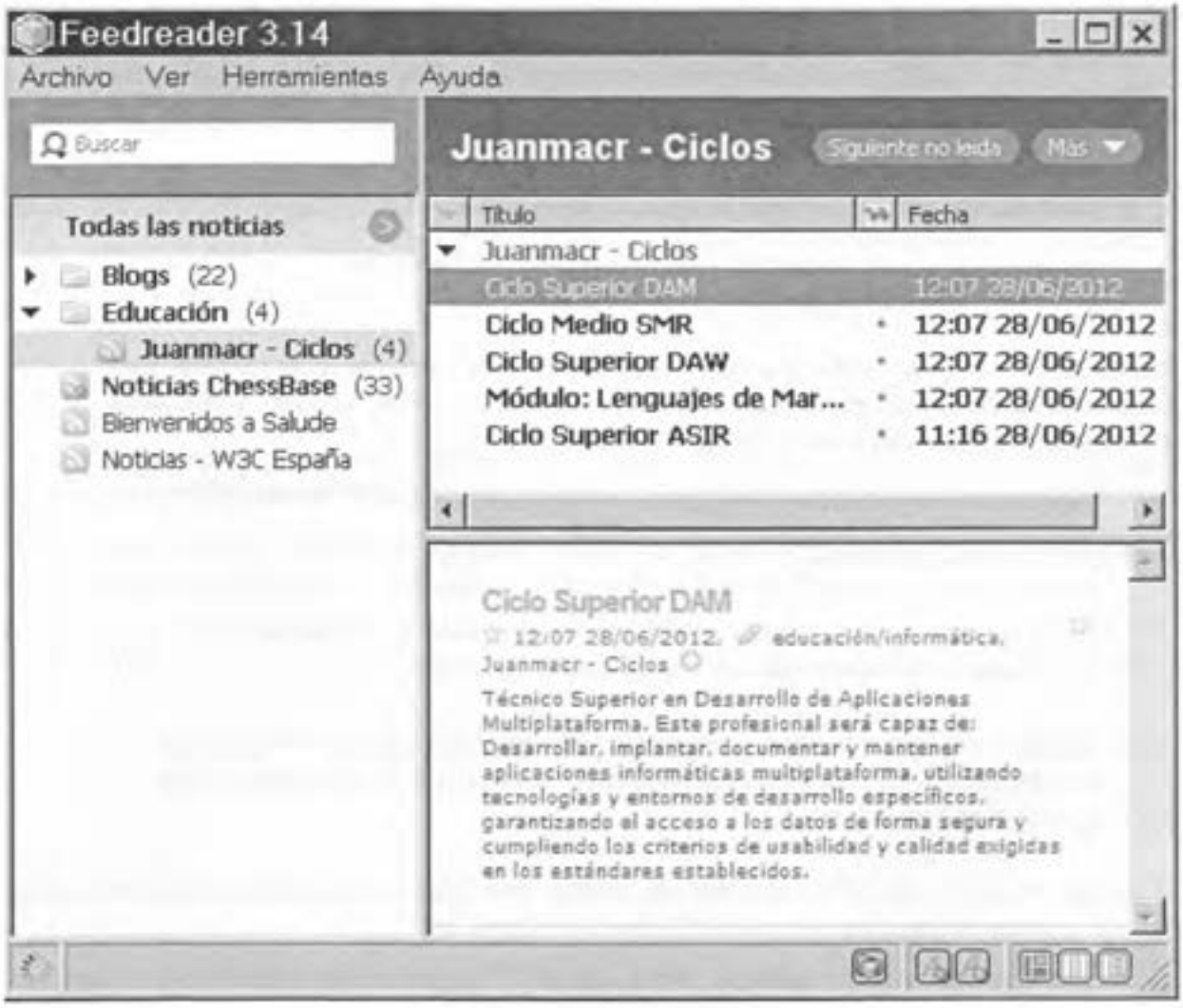


Figura 7.7: Vista del canal con Feedeader

En Google Reader se hace con el botón SUSCRIBIR, una vez agregado el canal, se vería de la siguiente forma:

Figura 7.8: Vista del canal en Google Reader

Como podemos observar en las últimas figuras, los lectores permiten crear carpetas para organizar las suscripciones por temas, ordenar por fechas o fuentes, también tienen otras opciones como ver los artículos en forma de lista o completos, etc.

Para consultar la fuente de donde proceden, basta con hacer clic sobre un artículo y nos redirecciona al sitio Web de origen.

Por último, podemos enviar la dirección de nuestro feed a directorios RSS, con el fin de mejorar los resultados de búsqueda, estos son algunos ejemplos:

http://bitadir.com, http://technorati.com, http://bitacoras.com, http://www.5z5.com, http://www.anse.de/rdfticker/addchannel.php, http://www.syndic8.com/suggest.php, http://www.rss-directory.us/, etc.

También podemos enviar la dirección de nuestro sitio Web a buscadores, como por ejemplo Bing: http://www.bing.com/webmaster/SubmitSitePage.aspx

(Esto se conoce como técnicas SEO, Search Engine Optimization, que significa, posicionamiento en buscadores)

Ejercicios propuestos

Ejercicio 1.

Crea el archivo RSS para sindicación de contenidos sobre tu sitio Web (debe contener al menos 5 artículos).

Súbelo al servidor e inserta un enlace en la página principal apuntando al archivo.

Comprueba como se ven los artículos con diferentes navegadores (observa las diferencias de presentación y las opciones que proporcionan).

Ejercicio 2.

Inserta el elemento <link> en la cabecera de la página principal, con la dirección de tu archivo RSS.

Inicia sesión en Google Reader y suscríbete a tu sitio Web.

Comprueba cómo se ven los artículos.

Realiza suscripciones a otros feed y ordena las suscripciones con carpetas.

Ejercicio 3.

Envía la dirección de tu archivo RSS a alguno de los directorios RSS sugeridos en el tema.

Comprueba después de cierto tiempo que tu feed ha sido incorporado al directorio.

Ejercicio 4.

Para simplificar el proceso de suscripción al usuario, incorpora un botón especial en tu página principal que permita suscribirse automáticamente a Google Reader.

Para hacerlo, basta con seguir los pasos indicados en las herramientas para webmasters de Google, en la dirección http://www.google.com/webmasters/add.html.

CAPÍTULO 8

Sistemas de gestión de información. ERP

CONTENIDO

Introducción
Inteligencia del negocio
ERP
CRM
Ejercicios propuestos

8.1. Introducción

En este capítulo se conocerán diferentes herramientas de **BI** (Business Intelligence – Inteligencia del negocio) que permite registrar información sobre distintos aspectos de una determinada actividad. Uno de los exponentes más claros de estas aplicaciones de inteligencia del negocio son los **ERP**, sistemas modulares integrados de gestión de empresas.

Un componente habitual de los ERP, que en ocasiones se distribuye de manera independiente, son los **CRM**, tipo de aplicación destinada a la gestión de clientes.

Este es un **capítulo accesorio** del libro. El **contenido más importante** es la comprensión de la estructura de un ERP y los elementos que lo forman.

8.2. Inteligencia del negocio

En el pasado, las empresas y organizaciones realizaban tareas de una manera artesanal, aprendiendo realmente poco del funcionamiento de sus propios procesos. Esto era debido a la dificultad de almacenar la información destacada de estos procesos y, sobre todo, a procesarla y recuperarla con rapidez.

Desde la aparición de los ordenadores personales, individuos y organizaciones han sido capaces de registrar datos en sistemas de información más o menos sofisticados. Desde simples archivos de texto hasta sofisticadísimos sistemas gestores de bases de datos, sistemas expertos, sistemas **CBR** (**C**ase **B**ased **R**easoning – Razonamiento Basado en Casos), etc.

Estos pasos que se han ido dando en la evolución de los sistemas de información han permitido la automatización de muchas tareas y la extracción de conclusiones muy interesantes.

En el momento en el que se ha dispuesto de una gran cantidad de datos, fácilmente procesables y recuperables, se han podido estudiar los diferentes procesos de cualquier actividad, sus casos de éxito y sus casos de fracaso. Esto ha permitido aprender en profundidad el funcionamiento de estos procesos, optimizarlos, transformarlos y, en ocasiones, eliminarlos. Se ha generando así la llamada inteligencia de negocio.

Se entienden entonces por aplicaciones de inteligencia de negocio aquellas que automatizan y agilizan procesos, registran resultados, favorecen la extracción de conclusiones, etc.

Aunque inicialmente estas herramientas fueron desarrollándose por las propias empresas para uso interno, poco a poco se fue generalizando su uso y aparecieron compañías que desarrollaban aplicaciones de inteligencia de negocio genéricas, pero adaptables o parametrizables a las necesidades de cada potencial compañía que las usara.

En cuanto a su ámbito de aplicación, ha habido herramientas muy generales, que cubrían muchos aspectos de un negocio y otras más específicas, centradas en alguna actividad más concreta.

Se pueden citar una serie de familias de aplicaciones con sus particularidades:

- **ERP** (**E**nterprise **R**esource **P**lanning – Planificación de Recursos Empresariales), son aplicaciones integrales (pueden llegar a cubrir las necesidades de un negocio en casi todas las áreas) y modulares (se suelen distribuir por paquetes o módulos que interaccionan entre sí de manera transparente al usuario).

- **CRM** (**C**ustomer **R**elationships **M**anagement – Gestión de Relaciones con Clientes), son aplicaciones que, aunque en ocasiones se integran en sistemas ERP, han ido teniendo un amplio rango de diferentes funcionalidades, lo que ha derivado en su desarrollo independiente de aquéllos.
- **CM** (**C**hange **M**anagement – Gestión del Cambio), son herramientas cuyo fin es facilitar a los empleados de una compañía los cambios estructurales que puedan tener lugar. Recogen información sobre los propios procesos de cambio.
- **BA** (**B**usiness **A**nalytics – Analítica del Negocio), son herramientas destinadas a analizar datos, resultados, tendencias y poder así sacar conclusiones sobre la evolución del negocio.
- **ETL** (**E**xtract, **T**ransform and **L**oad data – Extraer Transformar y cargar datos) o integración de datos, son aplicaciones orientadas a la fusión y homogeneización de datos de orígenes variados.
- **SFA** (**S**ales **F**orce **A**utomation system – sistemas de Automatización de Ventas), son aplicaciones que suelen formar parte de un CRM y permiten registrar todas las fases del proceso de una venta.
- **BPM** (**B**usiness **P**rocess **M**anagement – Gestión de Procesos de Negocio).
- **MRP** (**M**aterial **R**esource **P**lanning – Planificación de los Recursos Materiales).
- **SRM** (Supplier **R**elationship **M**anager – Gestión de la Relaciones con proveedores).
- **KM** (**K**nowledge **M**anagement – Gestión del Conocimiento).

8.3. ERP

ERP es la denominación genérica de cualquier sistema de gestión de la información interna y externa de una organización, incluyendo finanzas, contabilidad, ventas/compras, relaciones con clientes, gestión de almacenes...

El objetivo de estos sistemas es facilitar el **flujo de información** entre los diferentes estamentos de la empresa, más aún en empresas muy grandes y compartimentadas, sin el cual se producirían duplicidades en los esfuerzos realizados.

Los ERP suelen ser muy **versátiles**, funcionando en distintos sistemas operativos con distintas configuraciones de hardware. Habitualmente se apoyan en un sistema gestor de bases de datos para almacenar su información.

Características de un ERP:

- Integral: cubre todas las necesidades de gestión de una empresa
- Modular: las aplicaciones se organizan por módulos independientes pero combinables
- Parametrizable: con capacidad de adaptarse a las necesidades concretas de cada empresa
- Escalable: con capacidad de crecimiento

Ejemplos de ERP:

- SAP (http://www.sap.com/spain)
- Navision (http://www.navision.es)
- PeopleSoft. de Oracle (http://www.oracle.com/es/products/applications/peoplesoft-enterprise)
- Compiere (http://www.compiere.com)
- OpenBravo (http://www.openbravo.com/es)

8.3.1. SAP

SAP (*Systeme, Anwendungen und Produkte* – Sistemas, Aplicaciones y Productos) es el nombre de la empresa alemana que desarrolla el ERP comercial del mismo nombre más usado en la actualidad. La empresa SAP es el desarrollador de software más potente en Europa y de los mayores del mundo.

El nombre actual del ERP es **ECC** (**E**nterprise **C**ore **C**omponent – Componente Central de la Empresa) y su versión la 6.0. Anteriormente recibió otros nombres como R/1, después R/2 y el más conocido R/3 (sistema en tiempo real de 3 capas).

Está compuesto de multitud de módulos que cubren la mayoría de los procesos de negocio que puede necesitar cualquier empresa. Los módulos más destacados (los cuales tienen a su vez múltiples sub-módulos) son:

- **FI** (Financial) Finanzas.
- **CO** (Controlling) Costos y control.
- **LO** (Logistics) Logística.
- **SD** (Sales and Distribution) Ventas y distribución.
- **MM** (Materials Management) Gestión de materiales.
- **LE** (Logistics Execution) Ejecución logística.
- **PP** (Production Planning) Planificación de la producción.
- **HR** (Human Resources) Recursos humanos.
- **BC** (Basis Components) Componentes básicos.
- **IS** (Industrial Solutions) Soluciones industriales.
- **CRM** (Customer Resource Management) Gestión de clientes.
- **QM** (Quality Management) Gestión de calidad.

Como ya se ha comentado anteriormente, los ERPs se pueden adaptar a las necesidades específicas de una empresa. Esto se puede conseguir mediante la codificación de programas que realicen determinadas tareas necesarias para la empresa. SAP integra un lenguaje de programación de cuarta generación propio, **ABAP** (**A**dvanced **B**usiness **A**pplication

Programming – Programación de Aplicaciones Avanzadas de Negocio), que apareció con la versión R/2. Permite trabajar con distintos tipos de datos, acceder a diversas bases de datos, tratar archivos. Permite, así mismo, realizar llamadas a procesos remotos para comunicares con otros sistemas.

La última versión de SAP permite también codificar módulos en Java.

8.3.2. OpenBravo

OpenBravo es una alternativa a SAP en software libre. Se ejecuta a través de un cliente web y dispone de una interfaz **RIA** (**R**ich **I**nternet **A**pplication – Aplicación Rica de Internet) de fácil manejo. Dispone igualmente de una serie de módulos que se integran entre sí:

- Gestión de datos maestros.
- Gestión de aprovisionamientos.
- Gestión de almacenes.
- Gestión de proyectos y servicios.
- Gestión de la producción.
- Gestión comercial y CRM.
- Finanzas y contabilidad.
- Inteligencia de negocio.
- Retail **POS** – (**P**oint **O**f **S**ale – Punto de venta).

Figura 8.1: Módulos de OpenBravo

Actualmente se encuentra en la versión 3 (en adelante se usarán las siglas OB3 para referenciar el producto).

Se autodenomina el ERP ágil, porque se supone que su aprendizaje y posterior manejo son rápidos y fáciles.

Especificaciones técnicas

Los requisitos técnicos para la instalación de OB3 son:

- Servidor:
    - Sistema operativo: Microsoft Windows Server, Linux y otros SOs que soporten Java.
    - Base de datos: PostgreSQL u Oracle.
    - Lenguaje de programación: Java 2 SE
        - Framework: Hibernate y Weld, ambos de la comunidad de desarrolladores JBoss (Red Hat).
    - Servidor web: Apache.
        - Contenedor de servlets: Apache Tomcat.
        - Apache Ant (http://wiki.openbravo.com/wiki/ERP_2.50:Openbravo_ERP_installation/es)
    - Generación de informes: JasperReports, de Jaspersoft.
- Cliente:
    - Completamente web sin instalaciones.
    - JavaScript: librería Smartclient, de Isoporphic Software.
    - Ajax.

Trabajando con OpenBravo

Para experimentar el manejo de OB3, se usará la versión de demostración existente en el sitio web. Aunque solicita registrarse, se puede omitir el paso e ir directamente a la propia demostración.

Aparecerá una primera pantalla en la que se solicita el país de ubicación y el idioma a emplear. Los datos aparecen rellenos por defecto.

Figura 8.2: Pantalla de login de OB3

Actividad 8.1:

Conectarse al sitio de OpenBravo y acceder a la versión de demostración, seleccionando el país España y el idioma español. Una vez hecho esto, inspeccionar la interfaz.

Para empezar

Al empezar a ejecutar OB3 nos aparecerá uno de los *widgets* (complementos) llamado *Para empezar*, donde se pueden visualizar varios video-tutoriales y manuales para aprender diversos aspectos de la aplicación.

En el primero, *Take a look around*, se explica cómo llevar a cabo algunas funcionalidades comunes, como crear una factura, examinar los datos de un cliente o hacer un pago.

En el segundo, *Navigate OpenBravo*, se describe el manejo de la interfaz y sus elementos: espacio de trabajo, pestañas, tablas, formularios…

Figura 8.3: Sección *Para empezar* de OB3

La tercera opción, *Learn what OpenBravo can do for you*, es un manual en línea (por defecto en inglés pero con versión española en http://www.openbravo.com/es/product/erp/professional), donde se explican las funcionalidades que aparecen en las diferentes versiones del producto, la *Profesional* (software comercial) o la *Básica* (software libre con pago de subscripción).

Por último aparece una guía rápida, *Adopt OpenBravo for your organization*, donde se puede aprender sobre los requisitos de la aplicación, la instalación, la configuración de los distintos módulos…

Interfaz

La interfaz de OB3 es muy compacta y permite trabajar simultáneamente con múltiples ventanas.

Se divide en diferentes zonas:

- Menús (en la parte superior), donde se pueden crear informes (icono ✱), seguir atajos para acceder rápidamente a funcionalidades (icono ⏮ ·), gestionar alertas… También se sale de la aplicación (icono ⎆).
- En el lateral izquierdo aparece una zona de vistas y documentos abiertos recientemente. En la misma zona un poco más abajo se puede gestionar el *Espacio de trabajo*, que se comentará a continuación.
- La zona central la ocupa el *Espacio de Trabajo*, y las distintas pestañas que se hayan abierto. El *Espacio de Trabajo* es un área que podría simular una mesa de trabajo, en la que se pueden tener abiertos distintos *widgets*, que son pequeñas pantallas que nos ofrecen diferentes informaciones o funcionalidades.

Por otra parte, en muchas pestañas donde aparecen listados de diferentes conceptos (facturas, clientes, alertas…) existen una serie de funcionalidades que se repiten:

Icono	Acción
	Crear un registro
	Insertar una fila
	Guardar los cambios
	Cerrar registro actual y volver a vista en modo grid
	Cancelar los cambios
	Eliminar un registro
	Refrescar datos
	Exportar a hoja de cálculo
	Subir un archivo adjunto
	Obtener un enlace directo a la vista o registro
	Personalizar la disposición de los campos
	Guardar la vista

Tabla 8.2: Funcionalidades comunes en OB3

8.4. CRM

Un **CRM** (**C**ustomer **R**elationship **M**anagement – Gestión de Relaciones con Clientes) es una aplicación informática, en ocasiones integrada en un ERP, orientada al registro de información de clientes, ventas y marketing. Su objetivo fundamental es ganar y mantener clientes.

Un ejemplo de CRM podría ser la aplicación que se usase en un concesionario de automóviles donde se haga un seguimiento de clientes, ofertas, campañas publicitarias…

Existen una serie de tareas que suelen cubrir los CRMs:

- Almacenado de datos de clientes.
- Ofertas comerciales.
- Informes.
- Campañas publicitarias y de marketing.

Algunos de los CRM existentes actualmente en el mercado son:

- **Salesforce**, de Salesforce.com Inc, con licencia propietaria.
- **CiviCRM**, alternativa de software libre.
- **HiperGate**, software libre.
- **SugarCRM**, es otra alternativa de software libre.

8.4.1. SugarCRM

Es un CRM de software libre (http://www.sugarcrm.com). Actualmente se encuentra en su versión 6.5.

Al igual que los ERPs, dispone de una serie de módulos. Los más importantes son:

- Ventas.
- Marketing.
- Servicio al cliente.

Las características de la arquitectura del servidor son:

- Desarrollado inicialmente para LAMP (Linux, Apache, MySQL y PHP).
- Toda la lógica de negocio está implementada en PHP, por lo que no usa procedimientos almacenados ni triggers de la base de datos.
- Actualmente tiene soporte para diferentes SOs: Windows, Unix, Mac OS.
- Los sistemas gestores de bases de datos para los que hay versiones son MS SQL Server y Oracle.
- Modelo-Vista-Controlador.

La aplicación puede ejecutarse:

- En la nube: de la compañía, pública o en la de un socio.
- En local.
- En teléfonos inteligentes y tabletas.
- Flexible e integrable con otras aplicaciones.

Las funciones principales son:

- Mantenimiento de cuentas de empresas.
- Mantenimiento de contactos de personas.
- Mantenimiento de oportunidades de negocio.

- Creación de informes.
    - Tabular.
    - Sumarizado.
    - Sumarizado detallado.
    - Matricial.
- Mantenimiento de documentos.
- Mantenimiento de casos.
- Registro de llamadas.
- Enviar correo.
- Planificar reunión.
- Mantenimiento de tareas.
- Mantenimiento de notas o adjuntos.
- Crear caso.
- Crear campaña publicitaria:
    - Vía correo electrónico.
    - Vía correo ordinario.
- Creación de artículos para la base de conocimiento.
- Mantenimiento de empleados.

Interfaz

Para poder probar la versión de evaluación hay que registrarse de manera garutita. Una vez hecho esto, se podrá elegier entre 5 perfiles con los que accederse: gestor de ventas, director de ventas, director de marketing, representante para clientes y administrador de SugarCRM.

Se puede elegir el idioma de la demostración con las flechas que aparecen en la esquina inferior derecha de la pantalla de bienvenida.

Actividad 8.2:

Conecta con el sitio de SugarCRM y accede a la versión de demostración, registrándote de manera gratuita y accediendo con un par de usuarios diferentes. Comprueba que las funcionalidades ofrecidas a cada uno de ellos difieren, aunque existen algunas en común.

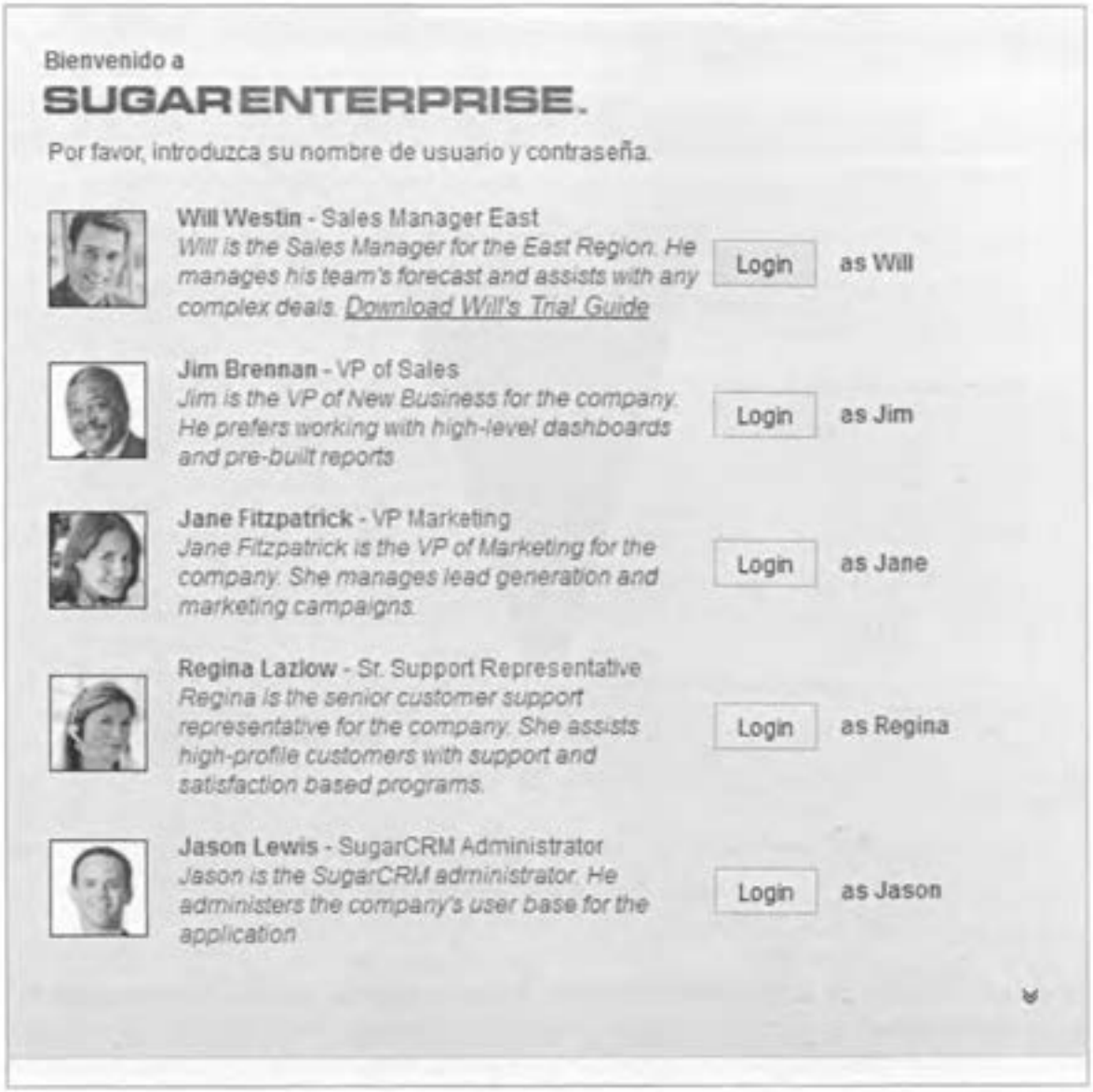


Figura 8.5: Pantalla de login de SugarCRM

La interfaz web se parece de alguna manera a la de OpenBravo 3. Aparece una barra de menús en la parte superior y una serie de pestañas en la parte central.

En los menús, como se observa en la *Figura 8.6*, se agrupan las funcionalidades por conceptos: Cuentas, Contactos, Oportunidades, Informes, etc. Cada uno de estos conceptos contiene, habitualmente, opciones de alta, baja, modificación y listado.

Figura 8.6: Interfaz principal de SugarCRM

En el lado derecho de la página de inicio (a la que se accede en todo momento con el icono del cubo existente en la esquina superior derecha), aparecen una serie de paneles que son parecidos también a los *widgets* de OpenBravo, es decir, una serie de pequeñas aplicaciones de apoyo. Ejemplos de esto son, por ejemplo, un acceso a Twitter y una lista de reuniones.

En la esquina superior derecha aparece un signo más que permite la creación rápida de los distintos elementos: cuentas, contactos, oportunidades…

más Buscar will

Agregar Sugar Dashlets Cambiar Diseño

My Twitter

(0 - 0 de 0)

Sin Datos

My Activity Stream

Will Enviar

(0 - 0 de 0)

Sin Datos

Mis Tareas Abiertas

(1 - 5 de 35)

Cerrar	Asunto	Relacionado con	Prioridad	Estado	Fecha de inicio	Fecha vencimiento
	call customer	aldo 10000	Alta	En Progreso	08-25-2011 13:00	08-25-2011
	Review Large Opportunity	Halliburton - 1000 Pro	Alta	No Iniciada	09-16-2011 10:45	09-25-2011
	Review Large Opportunity	500 Enterprise	Alta	No Iniciada	09-06-2011 15:15	09-08-2011

Figura 8.7: Paneles complementarios de SugarCRM

Ejercicios propuestos

1. Accede al sitio web de OpenBravo y ahí a la versión de demostración de OB3. En la aplicación, realizar una creación de algún concepto (icono ☆), modifícalo después y elimínalo finalmente.

   Explora las distintas funcionalidades existentes mediante consultas y listados.

2. Accede al sitio web de SugarCRM y crea un nuevo *Contacto*, accediendo después para su modificación y, por último, elimínalo.

   Explora las distintas funcionalidades existentes mediante consultas y listados.

Técnico Superior en Informática ASIR - DAM - DAW

Lenguajes de marcas y sistemas de gestión de información

Este texto tiene una orientación fundamentalmente práctica; para ello, el enfoque ha sido intercalar un buen número de ejemplos resueltos y actividades cortas entre los contenidos teóricos, para asimilar las ideas y dinamizar el desarrollo de las clases. Al final de cada capítulo hay ejercicios propuestos de mayor duración. Además, se aporta un buen número de enlaces a páginas web con múltiples ejemplos y contenidos teóricos para profundizar.

En cuanto a los contenidos, comienza con los lenguajes básicos para el desarrollo Web como son HTML/XHTML y CSS avanzado para una presentación adecuada. Sigue con el estándar XML para almacenamiento de datos. Se estudian lenguajes de expresiones para la recuperación de información, como XPath y XQuery. Igualmente se citan sistemas gestores XML nativos y no nativos que trabajan con XML. Además, se verá algún ejemplo de tratamiento de XML con Java. Continúa con las hojas de estilo XSLT y la transformación de documentos XML a diferentes formatos como HTML o PDF.

Para terminar, los últimos temas tratan sobre la sindicación o redifusión de contenidos mediante RSS y los ERP o sistemas de gestión de información, que sirven para gestionar de manera integral los procesos de producción, inventario, contabilidad, facturación, envío, etc. de una empresa.

Formación profesional

ISBN 978-84-1545-217-1

9 788415 452171

Garceta grupo editorial

www.garceta.es