



Lenguajes de marcas y sistemas de gestión de información

Elvira Mifsud Talón



Lenguajes de Marca y Sistemas de Gestión de Información

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito de los titulares del *copyright*.

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

Nota: Este libro se atiene al artículo 32 del derecho de cita de la Ley de Propiedad Intelectual de 1996 (R. D. Leg. 1/1996 de 12 de abril).

Derechos reservados © 2022, respecto a la primera edición en español, por:

McGraw Hill/Interamericana de España, S. L.

Basauri, 17 Edificio A, 1ª planta

28023 Aravaca (Madrid)

ISBN digital: 9788448626631

© Elvira Mifsud Talón

Corrección técnica: Avelina Beatriz Pérez Onate

Equipo editorial: Cristina García Sánchez y Silvia Garía Olaya

Corrección de estilo y ortotipografía: Àlex López Fructuoso

Ilustración: Pablo Vázquez Rodríguez, Archivo McGraw-Hill, 123rf

Fotografías: Archivo McGraw-Hill, 123rf



Unidad 1.

Características de los lenguajes de marcas



1. Lenguajes de marcas

No hay que confundir los lenguajes de marcas con los lenguajes de programación. En un lenguaje de programación se puede trabajar con variables de diferentes tipos o hacer operaciones lógicas o matemáticas. En un lenguaje de marcas no. El lenguaje de marcas lo que hace es añadir información acerca de la estructura del texto o su presentación.

En general, se utilizan marcas con muchas finalidades. Por ejemplo, se marca un texto en negrita si se considera importante una definición en un libro; el estudiante, al tomar apuntes en clase, establece su propio código de marcas; se marca de alguna manera la presencia o ausencia de los alumnos en clase utilizando algún software específico o simplemente en un cuaderno de notas, etcétera.

El lenguaje de marcas establece unas normas de marcado en aquellos textos que van a ser difundidos a través de la red o van a relacionarse entre ellos. Es decir, establece una codificación añadida al texto.

El siguiente ejemplo muestra un texto entre marcas con el objetivo de darle un significado concreto o una forma de representación:

```
<tema1>
  <fecha>22/05/2021 </fecha>
  <titulo>Lenguajes de marcas</titulo>
  <contenidos>Los lenguajes de marcas son ...</contenidos>
  <profesor>Profesor DAM titular</profesor>
</tema1>
```

A. Tipos de lenguajes de marcas

En función de su propósito, los lenguajes de marcas se agrupan en lenguajes de marcas de presentación, de procedimiento, descriptivos o semánticos y ligeros (LML). En la siguiente tabla se resumen sus características principales y algunos ejemplos.

Tipo de lenguaje de marcas	Características	Ejemplos
<i>De presentación</i>	Establecen el formato del texto, es decir, cómo lo verá quien lo vea o lea.	Ejemplo de este tipo es el utilizado desde cualquier procesador de texto, como LibreOffice, OnlyOffice o Word, que añade al texto marcas ocultas que determinan el aspecto que tendrá.
<i>De procedimiento</i>	También está dedicado a la presentación del texto, pero en este caso se indica al software que lo muestra las acciones que debe realizar.	Ejemplos de este tipo son los lenguajes HTML o Hipertext Markup Language y LaTeX.

	• El software que muestra el texto debe interpretar el código en el mismo orden en que aparece. • Cuando se edita el texto se muestran las marcas al usuario.	
<i>Descriptivos o semánticos</i>	• Utilizan marcas para describir los fragmentos de texto pero no determinan cómo será presentado el texto ni el orden en que aparecerá.	Ejemplo de este tipo son los lenguajes SGML (Standard Generalized Markup Language) y XML (Extensible Markup Language).
<i>Ligeros (LML)</i>	• Utilizan un tipo de formateo de texto bastante estandarizado, y su principal característica es que los archivos generados ocupan poco espacio y se editan con un editor de texto. • La reducción en el tamaño de los archivos permite optimizar el almacenamiento y supone un menor consumo de ancho de banda en su transmisión.	Ejemplos de lenguajes de marcas de este tipo son Markdown, ReStructured Text y Textile.

Tabla 1.1. Tipos de lenguajes de marcas.

En la sección **Material de trabajo > Descarga de documentos** está disponible el archivo [UD1_Clasificación.pdf](#), que contiene otros criterios de clasificación de los lenguajes de marcas.



IMPORTANTE

El lenguaje Markdown se utiliza para documentar proyectos de software. Lo veréis con detalle en el módulo de Entornos de desarrollo.

B. Características de los lenguajes de marcas

Las características generales de los lenguajes de marcas son las siguientes:

- Los documentos, al ser en texto plano, son independientes del software utilizado, pudiendo hacer uso de cualquier editor de texto.
- Las marcas añadidas están insertadas en el propio contenido.
- Sus componentes son elementos sencillos e intuitivos, y permiten la creación de lenguajes específicos dependiendo del tipo de documento.
- Son muy versátiles y con un ámbito de utilización muy extenso: páginas web, libros, gráficos, fórmulas matemáticas, vídeos, sindicación de contenidos, etcétera.

2. Conceptos básicos

Veamos cada uno de estos componentes.

A. Etiquetas o marcas

Las marcas son un conjunto de códigos que se añaden al texto para establecer su formato de visualización, la forma como se ha de imprimir o la estructura que tiene. Se añaden al texto, pero no forman parte de este.

Cada lenguaje tiene su propia colección de marcas con su clasificación y significado.

Hay tres tipos de etiquetas:

1. **Apertura:** carácter de inicio '<' y carácter de finalización '>'. Ejemplo: <p>.
2. **Cierre:** carácter de inicio '</' y carácter de finalización '>'. Ejemplo: </p>.
3. **Vacía:** carácter de inicio '<' y carácter de finalización '/>'. Ejemplo: <linea/>.

Ejemplo:

```
<p>...contenido_párrafo...</p>
```

donde <p> y </p> son las marcas que delimitan el párrafo.

En la práctica se pueden combinar marcas pertenecientes a diferentes lenguajes de marcas. No siempre es requerida la marca de cierre o final, depende del lenguaje concreto.

Las marcas pueden ser de dos tipos: etiquetas o referencias a entidades.

En esta unidad, más adelante, se explican las referencias.

En un documento, todo lo que no son marcas es contenido. Las etiquetas se pueden escribir tanto en mayúsculas como en minúsculas, el navegador las interpreta igual. La etiqueta <HTML> se interpreta igual que <html>. En la actualidad se recomienda utilizar siempre minúsculas.

B. Elementos

Un elemento representa una estructura lógica completa o un comportamiento concreto.

Por ejemplo, para HTML un elemento es un párrafo, una lista, una tabla, una imagen, etcétera.

La declaración de un elemento tiene una etiqueta inicial, un contenido y una etiqueta final.

Ejemplo:

```
<ul>
  <li><p>...contenido 1 de la lista...</p>
  <li><p>...contenido 2 de la lista...</p>
</ul>
```

La porción de código anterior representa una estructura de lista y, dentro de ella, dos estructuras de párrafo, cada estructura con su etiqueta inicial o de apertura y su etiqueta final o de cierre. Cada una de estas estructuras es un elemento.

Observar que la etiqueta `<li>` no tiene su correspondiente etiqueta de cierre. No es necesaria, y para `<p>` tampoco es necesaria. En estos casos se considera como cierre la aparición de otro elemento con su etiqueta, ya sea de apertura o de cierre.

Hay otros elementos que permiten omitir también la etiqueta inicial, es el caso de `<head>` y `<body>`. No hace falta escribirlas de forma explícita, porque siempre están. En general, y para HTML es su DTD (declaración de tipo de documento) el que determina si la etiqueta inicial y final son imprescindibles o no.

Un elemento es vacío si no tiene contenido. Por ejemplo, `<br>` es un elemento de salto de línea y no tiene contenido ni marca final, pero indica el comportamiento que se debe tener, es decir, cuando se encuentra este elemento se produce un salto de línea.

En HTML son elementos vacíos: `<area>`, `<base>`, `<br>`, `<col>`, `<embed>`, `<hr>`, `<img>`, `<input>`, `<link>`, `<meta>`, `<param>`, `<source>`, `<track>` y `<wbr>`.

No hay que confundir elemento con etiqueta. Los elementos son estructuras completas que contienen etiquetas o marcas.

C. Atributos

Los elementos pueden tener atributos, con unos determinados valores (entre comillas dobles " o entre comillas simples '), bien por defecto o bien asignados por el programador.

Los atributos deben ir incluidos dentro de '`<`' '`>`', y si se incluyen varias parejas atributo/valor deben ir separadas por un espacio en blanco. El orden en el que aparecen los atributos es indiferente.

Puede haber atributos en las etiquetas de apertura y en las vacías, pero no en las de cierre.

En el ejemplo siguiente, el elemento `<html>` indica el comienzo o la raíz del documento, y está siempre presente aunque no se incluya. En este caso se ha añadido el atributo `lang` (lengua) con el valor `es` (español).

```
<html lang="es">
```

En este otro ejemplo, el elemento `<meta>`, que es vacío, añade el atributo `charset`, que indica que el juego de caracteres a utilizar es utf-8.

```
<meta charset="utf-8">
```

Si se incluyen dos atributos quedaría:

```
<meta name="author" content="Nombre del autor" />
```



IMPORTANTE

Para ampliar información sobre los sistemas de codificación ASCII, UTF-8 y los atributos HTML ordenados alfabéticamente, consulta los [Enlaces web](#) en la sección [Material de trabajo](#).

D. Metadatos

Los metadatos, en general, proporcionan información sobre los propios datos.

En una página HTML, los metadatos dan información relativa al contenido de la página como, por ejemplo, el tema, la descripción de la misma, la codificación utilizada, las palabras que describen la página, etcétera.

Los metadatos son utilizados por los buscadores para definir la información principal de la página web (temática, descripción), y es muy importante que estén bien definidos si se quiere que en la web los buscadores indexen correctamente la página.

El elemento HTML asociado es `meta`.

Ejemplo:

```
<meta name="author" content="Nombre del autor" />
```

El metadato `meta` contiene el atributo `name`, que indica que el atributo `content` será el nombre del autor. En la unidad dedicada a HTML se verá con más detalle este elemento.

3. Evolución de los lenguajes de marcas

A continuación se da una breve introducción cronológica de los lenguajes de marcas más importantes y una comparativa entre ellos.

A. SGML

El lenguaje GML fue mejorado y ampliado, y en 1986, siguiendo la norma ISO 8879, aparece SGML o *Standard Generalized Markup Language*, estándar de lenguaje de marcado generalizado, que, en realidad, no es un lenguaje de marcas, sino un metalenguaje. Esto quiere decir que, utilizando SGML, se pueden definir otros lenguajes de marcas específicos, pero todos ellos con el objetivo común de facilitar la gestión, la publicación y el intercambio de documentos digitales con independencia del hardware y el software utilizados.

SGML es un estándar internacional, no propietario, y proporciona un sistema de codificación que permite la transferencia de documentación utilizando diferentes sistemas operativos, diferente software de edición, presentación y diferentes soportes hardware para su almacenamiento e impresión sin pérdida de información.

Pero un documento SGML no incorpora nada que indique cómo va a verse en la pantalla o impreso. Para ello habrá que adjuntar al documento todo lo relativo al estilo con el que se verá.

SGML es un marco general muy abierto; de hecho, no obliga a utilizar ningún conjunto de etiquetas concretas, y eso le añade complejidad. Por ello ha quedado en desuso a nivel general muy rápidamente, pero debido a su potencia y flexibilidad ha servido de base para la creación de los lenguajes de marcas más utilizados en este momento, como son HTML, XML y HTML5.

Como resumen, las tres propiedades de SGML son:

1. Utilización de codificación descriptiva.
2. Introducción del concepto «tipo de documento».
3. Independencia del sistema utilizado para representar el documento.

Los objetos SGML pueden ser:

- **Elementos:** insertados como etiquetas y sus atributos locales. La etiqueta `<!ELEMENT` se utiliza para la declaración de un elemento.
- **Referencias a entidades:** en general son unidades de contenido. En concreto, la norma ISO 8879 define entidad como «un conjunto de caracteres que pueden ser referidos como una unidad». La etiqueta `<!ENTITY` se utiliza para la declaración de una entidad.

Un documento SGML consta de tres partes o componentes (archivos):

1. **Declaración SGML.**
2. **Declaración de tipo de documento DTD.**
3. **Instancia del documento.**

Veamos cada uno de estos componentes.

El componente **declaración SGML** indica una serie de características del documento; por ejemplo, qué conjunto de caracteres se utilizan, los delimitadores, la sintaxis del documento y, en general, la normativa aplicada a él.

Este marco formal determinado por la declaración SGML va a delimitar el alcance de **la DTD**, es decir,

la DTD debe respetar la normalización impuesta por la declaración SGML. Y si no lo hace se desencadenará una cascada de errores en el documento.

Es obligatoria su existencia, y si no aparece se toma la sintaxis por defecto, como son el conjunto de caracteres y otras características.

Un ejemplo de declaración SGML sería la siguiente:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD Docbook V3.1//EN" []>
```

Observar cómo se indica en la declaración el tipo de documento. En este caso es `book`, y en ella ya se está indicando qué DTD se debe utilizar, que es `DTD Docbook`.

Por lo tanto, con el tipo de documento se está indicando la gramática que se va a utilizar y cómo se debe utilizar. SGML no está obligando a utilizar ningún tipo de marcado específico, y es la DTD asociada al tipo de documento la que establece la gramática.

El componente declaración de tipo de documento DTD e **instancia del documento** contiene la estructura del documento, es decir, todo lo necesario para crearlo. Se trata de las también llamadas «declaraciones de documentos», que sirven para poner nombre a las marcas y especificar el contenido de cada elemento.

Lo más usual es que la DTD se almacene en un archivo aparte con extensión `.dtd` y desde el documento SGML se referencie. En cualquier caso, es imprescindible que exista la DTD; de lo contrario, la herramienta que comprueba si el documento SGML está bien conformado, el **parser**, daría error.

El **parser** es el programa que se encarga de comprobar que se cumplen todas las reglas. Interpreta las etiquetas, resuelve las referencias a otros archivos incluyéndolos, ejecuta las instrucciones de procesamiento, etcétera.

Por ejemplo, un documento sin título no será validado por el *parser*, ya que no cumple las especificaciones de la DTD que obliga a que siempre exista.

Ejemplo de DTD para el tipo de documento DocBook:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.1//EN">
  <book lang="es">
    <bookinfo>
      <date>Abril 2021</date>
      <title>Lenguaje de marcas</title>
      <author>
        <firstname>Nombre</firstname>
        <surname>Apellido1 Apellido2</surname>
      </author>
    </bookinfo>
    <chapter id="capitulo1">
      <title>Evolución de los lenguajes de marcas</title>
      <para>
        Texto...
      </para>
      <sect1 id="seccion1">
        <title>SGML</title>
        <para>Introducción</para>
      </sect1>
    </chapter>
  </book>
```


Si el libro completo no estuviera en un solo archivo sino en varios habría que añadir la marca `ENTITY` de la siguiente forma, y en la que indicamos el nombre del archivo `.sgml` a incluir.

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.1//EN" [
    <!ENTITY capitulo2.sgml SYSTEM "capitulo2.sgml"> ]>
```

La instancia o muestra del documento es el propio documento y contiene el marcado y el texto con todos los objetos SGML necesarios siguiendo la estructura jerárquica establecida por la DTD.

Para escribir el código se pueden utilizar herramientas clásicas como Vim, Emacs, Bloc de notas, Notepad++, Gedit o también Visual Studio Code (Microsoft), que dispone de versiones para Windows, GNU/Linux y MacOS.



IMPORTANTE

Aconsejamos utilizar Visual Studio Code. En los [Enlaces web](#) está la URL de descarga, y hay información sobre la instalación y el uso básico de la herramienta.

El esquema siguiente muestra la relación entre SGML y los lenguajes de marcas que tienen su origen en él, como son HTML y XML.

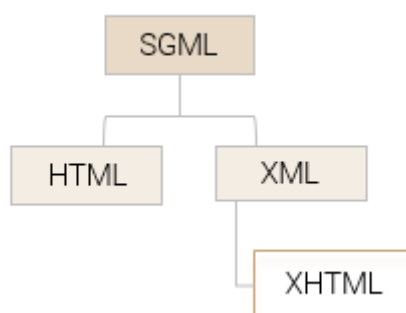


Fig. 1.1. Jerarquía de lenguajes de marcas.

B. HTML

HTML o *HiperText Markup Language*, lenguaje de marcas hipertextual, es un lenguaje de marcas que sigue el estándar SGML.

La *World Wide Web* (WWW) fue un proyecto desarrollado por Tim Berners-Lee a principios de los años noventa. El objetivo de dicho proyecto era enlazar grandes volúmenes de información ubicados en sistemas diferentes para que, en entornos científicos, se pudieran compartir datos, noticias y documentación. Para ello creó un nuevo lenguaje de marcas, HTML, que permitía la descripción de estos documentos y se basó en dos estándares que ya existían:

- **ASCII**, que representa en binario cada carácter del alfabeto latino, más una serie de caracteres de control y símbolos. Este formato de representación es entendido por los procesadores y editores de texto y, por lo tanto, permite la transferencia de información entre ordenadores.
- **SGML**, que proporciona estructura al documento, tal y como se ha explicado.

Pero HTML es mucho más sencillo que SGML, ya que solo incorpora al documento los elementos imprescindibles; eso hizo que su uso se expandiera y pasó a ser un estándar en la confección de páginas web. Pero no tiene la potencia del SGML, y además presenta algunos inconvenientes:

- Un conjunto de etiquetas limitado.
- El documento contiene tanto la estructura como el diseño.
- No permite mostrar contenido dinámico. En las últimas versiones comienza a incorporarlo.
- El mantenimiento de las páginas es costoso.
- Es muy sencillo para la presentación de información, pero no para su procesamiento o almacenamiento.

En la unidad 2 se estudia con detalle este lenguaje de marcas HTML con todos sus elementos.

C. XML

XML, o *eXtensible Markup Language*, lenguaje de marcas extensible, aparece en 1996 y sigue el estándar SGML, y al igual que él no es realmente un lenguaje de marcas sino un metalenguaje, es decir, permite la creación a partir de él de otros lenguajes de marcas que se adaptan a los usos.

Tiene ventajas o potencia similares a SGML, pero es menos complejo. En realidad, XML contiene un subconjunto de la funcionalidad de SGML.

Es de tipo descriptivo o semántico, y su objetivo es estructurar la información en documentos utilizando etiquetas propias y una gramática que determina la jerarquía mediante la DTD. Así, XML se puede utilizar para validar documentos que contienen fórmulas matemáticas (MathML), hojas de estilo (XSL), descripción de recursos (RDF), etc.

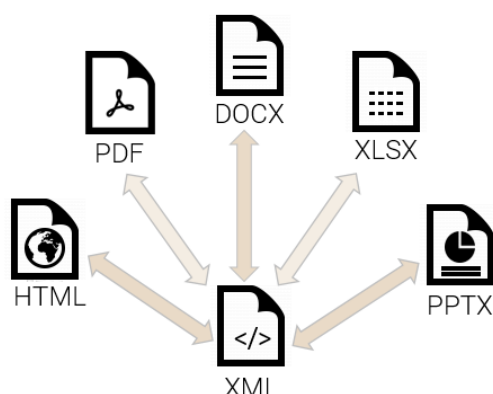


Fig. 1.2. Opciones de exportación de XML.

La figura 1.2 muestra diferentes opciones que ofrece XML para exportar documentos a otros formatos.

Las principales características de XML son:

1. Permite la creación de etiquetas propias y asignarles atributos.
2. Al incluir los metadatos en el propio documento, los motores de búsqueda de la web pueden indexarlos mejor y las búsquedas serán más efectivas.
3. La estructura del documento y su diseño están separados.
4. Permite exportar los documentos a otros formatos, como HTML, PDF y RTF, entre otros.
5. Permite compartir documentos entre diferentes sistemas operativos y plataformas, como por ejemplo páginas web y bases de datos.
6. Permite la internacionalización, ya que puede utilizar diferentes juegos de caracteres.
7. XML es un estándar abierto, es decir, se permite su uso libre sin problemas de licencias.

En [el sitio web del W3C](#) están disponibles las recomendaciones de XML publicadas por el W3C, *World Wide Web Consortium*.

En la sección **Material de trabajo > Descarga de documentos** está disponible el archivo [UD1_Organizaciones_Internacionales.pdf](#), con información sobre las organizaciones internacionales desarrolladoras, entre ellas el W3C.

D. Comparativa: XML y HTML

La tabla siguiente muestra las principales características de XML y HTML de forma comparativa:

XML	HTML
Es un metalenguaje.	Es un lenguaje de marcas.

Subconjunto funcional de SGML.	Derivado de SGML.
No es un lenguaje de presentación.	Es un lenguaje de presentación.
Describe información.	Da formato y muestra información.
XML es dinámico y se usa para transferir datos.	HTML es estático, se usa para mostrar datos.
Permite la definición de nuevas marcas, es decir, puede extenderse.	Conjunto limitado de marcas predefinidas en su estándar.
Obliga a cerrar las etiquetas.	Las etiquetas pueden funcionar bien sin necesidad de cerrarlas.
XML imprime los espacios en blanco.	HTML no conserva los espacios en blanco.
Utiliza el navegador como plataforma para el desarrollo de aplicaciones.	Utiliza el navegador solo para mostrar información.
Obliga a ser ordenados.	Se puede ser desordenado.
Proporciona interoperabilidad con SGML y HTML.	Facilita la transferencia de información basada en la web.
Objetivo principal: almacenar y transferir datos.	Objetivo principal: presentar datos.

Tabla 1.1. Comparativa entre XML y HTML.

E. Comparativa: XML y SGML

Como se ha mencionado anteriormente, XML sigue el estándar SGML y, al igual que él, no es realmente un lenguaje de marcas sino un metalenguaje, es decir, permite la creación a partir de él de otros lenguajes de marcas que se adaptan a los usos que se vayan a hacer.

En la siguiente tabla se resumen las principales diferencias entre ambos lenguajes:

XML	SGML
Subconjunto de la funcionalidad de SGML pero más sencillo.	Muy complejo.
Introduce el concepto de documento "bien formado" sin requerir validación.	Requiere la validación del documento.
Muy utilizado en diferentes áreas.	Uso muy restringido por su complejidad.
Compatible con HTML.	No hay compatibilidad específica con HTML.
Formateo y estilo relativamente fáciles de aplicar.	Formateo y estilo relativamente difíciles de aplicar.

Tabla 1.2 Comparativa entre XML y SGML.

Reseñar que a lo largo de las unidades del módulo Lenguajes de Marcas se podrán utilizar indistintamente los términos «marca» y «etiqueta», teniendo ambos el mismo significado.

4. Iniciación a XML

Comenzamos describiendo el origen del lenguaje XML.

A. Origen del XML

En 1989, Tim Berners-Lee utilizó como base SGML para crear HTML, con el objetivo de ser utilizado en el laboratorio europeo CERN, pero rápidamente se extendió su uso para el intercambio de información a nivel mundial. El mismo Berners-Lee, en 1994, creó el organismo internacional W3C para hacer un seguimiento, una normalización y una ampliación del nuevo lenguaje HTML.

Pero llegó un momento en que su uso obligaba a una ampliación de aplicaciones tan dispares que el W3C decidió establecer unas reglas más generales para que se pudieran crear diferentes lenguajes de marcas en función de las necesidades específicas de los usuarios. Y esas reglas deberían permitir la compatibilidad con HTML a nivel de estructuras, sintaxis, etc. Y así fue como apareció XML en 1998, como una plataforma base para, sobre ella, generar diferentes lenguajes de marcas.

El W3C continuó el desarrollo de XML generando o incorporando todas aquellas tecnologías necesarias para procesar los documentos XML y que pudieran ser utilizadas desde cualquier lenguaje de marcas derivado de XML.

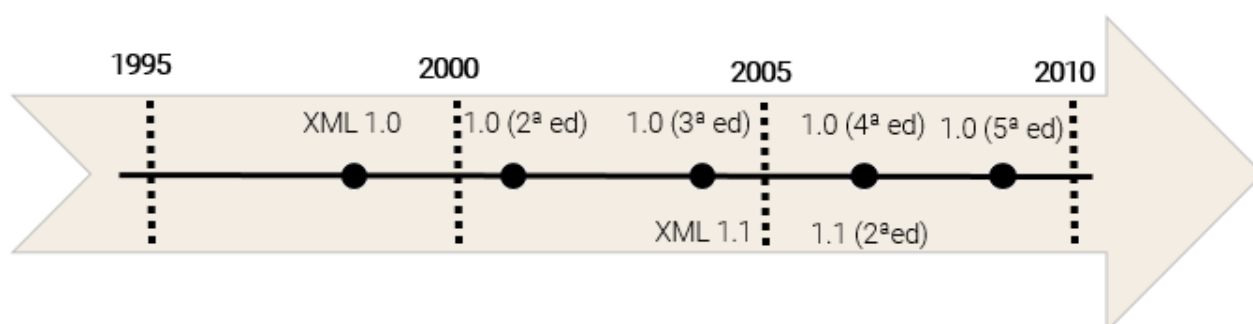


Fig. 1.1. Línea de tiempos para XML.

En el año 2000, y dado que HTML no seguía el estándar XML, el W3C creó el XHTML, que era un HTML acorde a XML. Entonces el desarrollo de HTML terminó.

Otra iniciativa del W3C fue impulsar la llamada web semántica. La semántica es la parte de la lingüística que estudia cómo los seres humanos dan sentido a las expresiones lingüísticas a través de las estructuras léxicas y los procesos mentales (Ver [Enlaces web](#) en la sección [Material de Trabajo al final de la unidad](#)).

Su objetivo es que la búsqueda de información por parte de los usuarios sea más sencilla e intuitiva, utilizando como base XML. Esta iniciativa del W3C no tuvo éxito, ya que la web no adoptó ninguna de las recomendaciones de la web semántica.

El recorrido del nuevo lenguaje XHTML fue muy corto por diferentes motivos, y en 2014 el W3C retoma el desarrollo de HTML creando las recomendaciones para HTML5, una que está basada en XML y se llama XHTML5, y otra no basada en XML llamada HTML5.

Por último, decir que en 2018 el W3C abandonó el desarrollo de XML, ya que cerró todos los grupos dedicados a ello. Como principal motivo de este cierre está el hecho de que XML es altamente exigente con el tema de los errores, es un lenguaje muy estricto en ese sentido.

De cualquier forma, XML sigue siendo un lenguaje muy utilizado en todo lo relativo al almacenamiento y la transmisión de la información, y es muy posible que el W3C lo retome en un futuro.

B. Características del metalenguaje XML

XML es un metalenguaje de tipo descriptivo o semántico, y su objetivo es estructurar la información en documentos utilizando etiquetas propias y una gramática que determina la jerarquía mediante la DTD o *Document Type Definition*.

Su potencia radica en su interoperabilidad entre diferentes sistemas operativos y aplicaciones, además de ser un estándar para el almacenamiento de información, creado y optimizado para ser utilizado en la web.

Un documento XML se guarda en un archivo .xml y es un archivo de texto cuyo contenido son parejas de etiquetas con estructura de árbol que encierran la información propiamente. Por lo tanto, este archivo .xml puede ser editado con cualquier editor de texto incluso en la nube utilizando, por ejemplo, [XML Viewer](#) o [XMLGrid](#), que además son gratuitos.

Hay que tener en cuenta que, aunque el archivo .xml es un archivo de texto, no es exactamente un archivo .txt. Para la conversión de un archivo XML a otros formatos, y si la propia herramienta con la que se generó no lo permite, hay que hacerlo utilizando algún tipo de software específico o servicio en línea.

En [Material de trabajo > Descarga de documentos](#) tienes disponibles enlaces a diferentes conversores en el archivo [UD1_Conversores_XML.pdf](#).

Además de las características básicas de XML ya enumeradas, ahora se van a completar con otras pero enfocadas a la web:

- Un documento XML tiene que estar «bien formado», es decir, tiene que cumplir unas reglas, y si no las cumple el *parser* lo rechazará.
- Un documento XML, además de las reglas del lenguaje, debe cumplir las reglas del esquema. El esquema define qué elementos se pueden utilizar, cómo se pueden anidar y tipos de atributos que se pueden incluir. Solo así será un documento válido.
- Al ser un lenguaje extensible se puede adaptar a diferentes requerimientos. Por lo tanto, su lista de etiquetas no es ni fija ni cerrada.
- Su objetivo es la descripción semántica del texto, no la presentación del mismo.
- Al incluir los metadatos en el propio documento, los motores de búsqueda de la web pueden indexarlos mejor y las búsquedas serán más efectivas.

C. Marcado del documento

El marcado de un documento se hace para estructurar su contenido.

Para colocar las marcas hay que tomar una parte del texto que tenga un significado concreto. La marca asociada se encierra entre los caracteres "<" y ">", que siempre se interpretan como código.

Una marca es, simplemente, una secuencia lineal de caracteres con una estructura concreta.



EJEMPLO 1

Revisión de etiquetas

Volvemos al código XML del caso práctico 3 y se propone revisar las diferentes etiquetas y la estructura del código.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE libro>
<libro>
  <titulo>Lenguaje de marcas: XML</titulo>
  <autor>Nombre autor</autor>
  <editorial>McGraw-Hill</editorial>
  <isbn>978-2-7360-4965-1</isbn>
  <depositolegal>M-8672-2021</depositolegal>
  <paginas>270</paginas>
</libro>
```

Solución

La línea de código `<titulo>Lenguaje de marcas: XML</titulo>` incorpora dos etiquetas, una de apertura (`< >`) y otra de cierre (`</ >`), igual que en HTML. Son etiquetas que se incluyen para delimitar una porción de texto del documento que identifica el título. Como alternativa se puede utilizar también `<` para representar `'<'` y `>` para `'>'`.

En el ejemplo se observa que el documento XML tiene un nodo raíz y de él cuelgan todos los elementos, pudiendo presentar cuantas anidaciones se requieran de estos elementos y creando una estructura en árbol. En el ejemplo solo existe un anidamiento, siendo el nodo raíz el elemento `<libro>`.

Elementos, atributos y valores

El elemento es la unidad de información semántica y consta de una etiqueta o marca de apertura `'<'` y otra de cierre `'>'`.

Un elemento consta de:

1. El nombre del elemento o identificador.
2. Uno o varios atributos con sus valores respectivos.

Los atributos permiten añadir propiedades a los elementos del documento. Pero no añaden nada a la estructura lógica del documento XML y no pueden contener otros elementos y/o atributos.

```
<elemento atributo="valor">contenido</elemento>
```

que, en el ejemplo concreto, añadiendo el atributo `'id'`, será:

```
<titulo id="XML">Lenguaje de marcas: XML</titulo>
```

El nombre del atributo debe ir dentro de la etiqueta de apertura, separado por un espacio del nombre del elemento y seguido de `'='`. El valor asignado irá entre comillas dobles.

El orden de los atributos dentro de la etiqueta es indiferente, pero no se deben repetir.

Cuando un elemento no tiene contenido se llama elemento vacío y se representa `<elemento_vacio/>`.

A continuación, y como resumen, se indican unas reglas mínimas para la formación de elementos. De esta forma, el procesador XML no dará error.

Respecto al nombre del elemento:

1. No puede empezar por un dígito, signo o la cadena 'xml'.
2. Solo puede contener letras y dígitos, pero no acentos o espacios.
3. Puede contener algunos caracteres como el guion '-', el guion bajo '_' o el punto '.'.
4. Se diferencia entre mayúsculas y minúsculas.

Respecto a la estructura:

1. Un documento XML debe tener un único elemento raíz.
2. Se permite la anidación de elementos siempre que estén correctamente abiertos y cerrados. Es decir, no se puede cerrar un elemento si tiene anidado otro elemento abierto.
3. El texto contenido en un elemento no puede contener los caracteres '<', '>', '&', '"', '''.

La tabla siguiente muestra las cadenas que sustituyen a estos caracteres en el caso de que sea necesaria su inclusión en el texto.

Carácter	Cadena	Carácter	Cadena	Carácter	Cadena
>	>	&	&	'	'
<	<	"	"		

Tabla 1.1. Cadenas que sustituyen a caracteres.



EJEMPLO 2

Identificar atributos

El siguiente código muestra un documento XML en el que algunos de sus elementos contienen atributos. Identificar cuáles son esos elementos. ¿Qué se verá en el navegador?

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE biblioteca >
<biblioteca>
    <ejemplar tipo_ejem="libro" titulo="Lenguaje de marcas: XML"
editorial="McGraw-Hill">
        <tipo> <libro isbn="978-2-7360-4965-1" edicion="1"
paginas="270"></libro> </tipo>
        <autor nombre="Nombre autor"></autor>
        <autor nombre="Nombre Apellido1 Apellido2" funcion="traductor"></autor>
        <prestado lector="Fulanito">
            <fecha_pres dia="1" mes="ene" año="2021"></fecha_pres>
            <fecha_devol dia="31" mes="ene" año="2021"></fecha_devol>
        </prestado>
    </ejemplar>
</biblioteca>
```

Solución

Crear un archivo .xml con el código dado y comprobar que en el navegador lo que se verá es el propio código XML. Vemos que los elementos aparecen coloreados en ciruela, los nombres de los atributos en rojo y sus valores en azul.

Como se observa en el código, los atributos se definen y dan valor dentro de una etiqueta de inicio o de elemento vacío, a continuación del nombre del elemento o de la definición de otro atributo y siempre separado de ellos por un espacio. Los valores del atributo van precedidos de un igual que sigue al nombre del mismo y tienen que definirse entre comillas simples o dobles.

Los nombres de los atributos han de cumplir las mismas reglas que los de los elementos, y no pueden contener el carácter menor que, '<'.



IMPORTANTE

Hay que tener en cuenta que XML diferencia entre mayúsculas y minúsculas en los identificadores y los atributos.

Se pueden (y deben) añadir comentarios a los documentos XML. El intérprete XML no los tiene en cuenta pero añaden información que facilitará la posterior lectura y comprensión de dicho documento. Estos se incluyen entre las cadenas "<!--" y "-->".

D. Partes de un documento XML

Un documento XML contiene las siguientes partes:

1. Prólogo

No es obligatorio, pero si se incluye debe ir al comienzo del documento. Consta de la declaración XML y la declaración de tipo de documento.

Declaración XML (instrucciones de procesamiento).

Un documento XML comienza con la declaración XML:

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

Antes del elemento declaración XML no se puede añadir otro elemento, comentarios o espacios en blanco.

Al ser opcional, los documentos SGML y HTML pueden ser procesados como si fueran XML.

Las funciones de la declaración XML son tres:

1. **Declara que es un documento XML y la versión de XML utilizada.** Para ello se incluye:

```
<?xml version="1.0"?>
```

donde el carácter '<' seguido del interrogante de cierre (?) se interpreta como código y le indica que es una declaración o una instrucción de procesamiento, no de un elemento normal.

Es importante conocer la versión del lenguaje utilizada por si, en el futuro, con nuevas versiones, hubiese algún tipo de incompatibilidad.

2. **Declara el tipo de codificación de caracteres utilizada.**

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

que, en el ejemplo, es ISO-8859-1, que permite el uso de acentos o la 'ñ' española, aunque por defecto utiliza UTF-8 (conjunto de caracteres universal).

También se puede incluir la hoja de estilo utilizada para la visualización del documento. Por ejemplo:

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

3. **Declara la autonomía del documento.** Utilizando el atributo `standalone` se indica si el documento tiene (no) o no tiene (yes) dependencia de un esquema externo. Si no depende

(standalone="no"), el documento sigue las reglas de XML. Si depende (standalone="yes"), el documento sigue un DTD externo.

Se incluye en la declaración de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
```

Declaración de tipo de documento

Con la declaración de tipo de documento se está indicando qué tipo de documento se está creando. Siempre comienza por la cadena:

```
<!DOCTYPE Nombre_tipo ...>
```

Si la definición de tipo de documento DTD se incluye en el propio documento, la sintaxis es:

```
<!DOCTYPE Nombre_tipo [... declaraciones ...] >
```

Ejemplo:

```
<!DOCTYPE libro SYSTEM "http://localhost/libro.dtd">
```

En otra unidad se volverá sobre el DTD ampliando y completando todos los detalles relativos a dicho elemento.

2. Cuerpo del documento o ejemplar

El cuerpo del documento contiene los datos o el texto del documento XML. Su estructura es una serie de elementos anidados que son los que contienen los bloques de texto. Como ya se ha dicho, cada elemento lleva una marca de apertura y una de cierre del elemento, y cada elemento puede llevar atributos.

En un documento XML no puede haber ningún texto, dato o información que no esté contenido en un elemento.

En general, se busca que los nombres asignados a las marcas o etiquetas sean significativos, es decir, que tengan una relación con el contenido descrito.

Por último, y como resumen, se incluyen las reglas básicas que debe seguir la estructura de un documento XML.

En un documento XML:

1. Los elementos que lo componen deben tener estructura de árbol, es decir, jerárquica.
2. Solo puede haber un elemento raíz.
3. Los elementos deben estar correctamente anidados.
4. Los elementos no se deben superponer entre ellos.

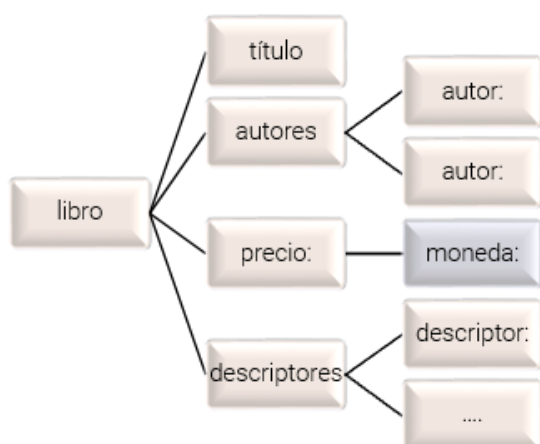


Fig. 1.2. Estructura jerárquica de un libro.

El esquema de la figura 1.4 muestra un ejemplo de dicha estructura jerárquica para el caso de un libro con varios autores, un precio y varios descriptores:

Es muy importante conocer la estructura jerárquica de la información antes de la creación del documento XML. Ayudará a poder manejarla más tarde.

E. Documento XML bien formado

Se dice que un documento XML está bien formado cuando cumple las siguientes reglas o normas sintácticas definidas en la recomendación del W3C:

- Debe tener definido un prólogo con la declaración XML completa.
- Solo debe existir un único elemento raíz, que anide al resto de elementos del documento.
- Las instrucciones de procesamiento son opcionales.
- Un elemento debe comenzar y terminar con la misma etiqueta.
- Los atributos de los elementos son opcionales.
- Los valores de los atributos deben ir entre comillas dobles ("") o comillas simples (').
- Se diferencia entre mayúsculas y minúsculas. En XML no es lo mismo <etiqueta> que <ETIQUETA>.



IMPORTANTE

No hay que confundir el concepto de «documento bien formado» con el de «documento válido». Un documento decimos que está bien formado cuando cumple las reglas antes descritas, y un documento válido, además de estar bien formado, tiene que cumplir las reglas que establece su DTD o esquema.

El concepto de esquema se estudia en otra unidad, en la que se verán todos los mecanismos de validación.

5. Espacio de nombres en XML

Si se quiere eliminar problemas derivados de una posible repetición de nombre o atributo, la solución pasa por asignar a cada vocabulario XML un espacio de nombres, es decir, un espacio semántico único. De esta forma desaparecen posibles ambigüedades al trabajar conjuntamente con documentos XML que han utilizado los mismos nombres de elementos o atributos para representar cosas diferentes.

Por ejemplo:

```
<capital>
  <pais>Francia</pais>
  <nombre>París</nombre>
</capital>
<capital>
  <bonos>
    <cantidad>10000 €</cantidad>
  </bonos>
  <letras>
    <precio>1000 €</precio>
  </letras>
</capital>
```

Si en un documento XML se incluyen los elementos <capital> hay un conflicto de nombres.

Por lo tanto, los espacios de nombres en XML se utilizan para:

- Diferenciar entre elementos y atributos de vocabularios diferentes con diferentes significados pero que comparten nombre.
- Agrupar todos los elementos y atributos relacionados de un documento XML para que el software pueda reconocerlos con facilidad.

La utilización de espacios de nombres XML es una recomendación del W3C.

Cuando en un documento XML hay conflicto de nombres de etiquetas a estas etiquetas, para referenciarlas se les añade un prefijo que especifica a qué contexto pertenece esa etiqueta.

Esto es:

```
<prefijo:etiqueta></prefijo:etiqueta>
```

Pero previamente hay que declarar el espacio de nombres para cada una de esas etiquetas conflictivas.

Declarar un espacio de nombres consiste en asociar un índice con el URI asignado al espacio de nombres. Para ello se utiliza el atributo reservado **xmlns**, y se inserta entre el prólogo y el ejemplar del documento XML de la forma siguiente:

```
<elemento xmlns:nombre_prefijo="URL">
  <capital xmlns:capital="http://localhost/UD1/capital">
  <contab xmlns:capital="http://localhost/UD1/contab">
```

Donde:

- El espacio de nombres se inicia con la palabra clave **xmlns**.
- La palabra nombre_prefijo es el prefijo del espacio de nombres. En el ejemplo, 'capital'.
- La URI es el identificador del espacio de nombres. Actúa como un identificador único.

El prefijo actúa como un alias, de forma que todos los nombres de elementos que lleven ese prefijo pertenecen a ese espacio de nombres.

De esta forma, cuando nos referimos en el documento a capital habrá que indicarlo con su prefijo correspondiente:

```
<capital:capital>
```

```

    <pais>Francia</pais>
    <nombre>París</nombre>
</capital:capital>
<contab:capital>
    <bonos>
    <cantidad>10000 €</cantidad>
    </bonos>
    <letras>
    <precio>1000 €</precio>
    </letras>
</contab:capital>

```

Así no hay confusión sobre a qué `capital` nos referimos en cada caso y a qué espacio de nombres corresponde.

El URI, *Uniform Resource Identifier*, o identificador uniforme de recursos, es realmente el nombre del espacio de nombres y no se interpreta como una dirección web sino como una cadena de texto que identifica el espacio de nombres (el *parser* XML lo trata como una cadena de texto).

Es importante diferenciar entre URI y URL.

URI (Uniform Resource Identifier): es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

URL (Uniform Resource Locator): es un identificador de recursos uniforme URI cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo.

El concepto de espacio de nombres XML se retomará en otra unidad de forma más ampliada, así como la definición de vocabularios XML en la DTD o en el Schema.



Unidad 2.

Utilización de lenguajes de marcas en entornos web



1. Introducción. Estándares web. Versiones

En 1997, el W3C comienza la normalización para conseguir la accesibilidad web (WAI, *Web Accessibility Initiative*). Este fue el punto de partida para la aparición de nuevos estándares que consiguieron que los contenidos web se puedan visualizar de la misma forma en diferentes navegadores. Además, con la ayuda de los estándares se consiguió que la web fuera un sistema abierto, gratuito y accesible para cualquier usuario.



Fig. 2.1. Estándares web del W3C.

¿Qué se entiende por estándares web?

Los estándares web se pueden definir como un conjunto de tecnologías específicas que se utilizan para crear sitios web.

Y ¿qué es un estándar en general?

Un estándar es un documento que contiene las especificaciones técnicas de la tecnología referida y que tienen que ser implementadas por los desarrolladores.

Por ejemplo, para implementar código HTML5, el estándar a seguir está disponible en el siguiente enlace de [WHATWG](https://whatwg.org/).

Además del W3C existen otros organismos dedicados a estandarización. Entre ellos está WHATWG (<https://whatwg.org/>, *Web Hypertext Application Technology Working Group*, agrupación de empresas y personas implicadas en el desarrollo del HTML entre las cuales están Apple, Google, Mozilla, etcétera), el [ECMA](#) (que publica el estándar que cumple JavaScript), entre otros.

Entre las ventajas más significativas del uso de estándares están las de:

- Mantener la web gratis y abierta para todos.
- Ayudar a simplificar el código fuente.
- Reducir del tiempo de desarrollo y mantenimiento de los contenidos web.
- Permitir la compatibilidad entre navegadores.
- Ayudar en la optimización de motores de búsqueda.

Entre los estándares web más conocidos están:

- HTML, *HyperText Markup Language*, lenguaje de marcas de hipertexto.
- XML, *eXtensible Markup Language*, lenguaje de marcas extensible.
- CSS, *Cascading Style Sheets*, hojas de estilo en cascada.
- JavaScript.

Respecto a las versiones de los diferentes estándares web, en la actualidad están vigentes:

- HTML. La versión 5. En el apartado siguiente se repasa su evolución brevemente.
- XML. La versión actual es la XML 1.0 5.ª edición (2008).
- CSS. La versión actual es la CSS3, publicada en 2011.
- Javascript. La versión actual estable es ES12 o ECMAScript 2021.

A. Evolución del HTML

HTML (*HyperText Markup Language*, lenguaje de marcas de hipertexto) es un lenguaje de marcas estándar y utilizado en la creación de páginas web. El código generado por este lenguaje de marcas establece la estructura del contenido de la página web como son el texto, las imágenes, los vídeos, el sonido, etcétera.

HTML se caracteriza por ser reconocido por todos los navegadores y muestra prácticamente el mismo aspecto de la página web, con independencia del sistema operativo de base.

Este lenguaje ha ido evolucionando desde las primeras versiones hasta el HTML5. Con el detalle de que, hasta la versión anterior del HTML5, las recomendaciones eran creadas por el W3C. Pero a partir de los años noventa, W3C deja el desarrollo del HTML y comienza con unas recomendaciones nuevas que siguen el estándar XML y que llamó XHTML. La versión HTML5 fue desarrollada en 2004 por el grupo WHATWG.

A partir de esta fecha se producen una serie de acercamientos y alejamientos entre el W3C y WHATWG hasta 2011, cuando separan sus trabajos definitivamente. Ambas organizaciones siguen sus desarrollos del HTML5 dando lugar a diferentes versiones (HTML 5.1 en 2016, HTML 5.2 en 2017) hasta 2018, fecha en la que el W3C abandona definitivamente el HTML 5.3 en favor de WHATWG con el compromiso de publicar una versión de HTML anualmente.

En la actualidad, HTML5 es el lenguaje de marcas para el web más utilizado y se tiene como estándar, aunque sigue evolucionando y adaptándose a las nuevas necesidades que plantean las nuevas tecnologías que van apareciendo. Es, por lo tanto, un lenguaje dinámico y vivo. Además, HTML5 funciona correctamente en dispositivos móviles.

Los desarrollos de esta unidad se basan en HTML5.

La siguiente tabla muestra la evolución de las diferentes versiones del HTML:

HTML	Creado por Tim Berners-Lee al comienzo de los años noventa. Fue el primer sistema de hipertexto para compartir documentos.
HTML 2.0	En 1995, el organismo IETF, Internet Engineering Task Force, organiza un grupo de trabajo de HTML y publica el primer estándar oficial HTML 2.0.

HTML 3.2	En 1997 se publica la versión 3.2, y es la primera recomendación de HTML publicada por el W3C. Esta versión incorpora la utilización de los applets de Java.
HTML 4.0	En 1998, W3C publica la versión 4.0, que incorpora la utilización de las hojas de estilo CSS, añade mejoras en la accesibilidad, la opción de incluir guiones o scripts en las páginas web, optimiza las tablas y los formularios, etcétera.
HTML 4.01	En 1999, el W3C publica la última especificación oficial de HTML, la 4.01, que es una actualización de la versión HTML 4.0 y no incluye novedades significativas. En este punto, W3C paraliza el proceso de estandarización del HTML.
HTML 5	En 2014, WHATWG publica la versión 5 de HTML. En dicha versión se añaden varias etiquetas semánticas nuevas que indican el significado del contenido, como <article>, <header> y <footer>, entre otras.

Tabla 2.1. Evolución del HTML.

Respecto a XHTML, la primera versión (1.0) se publicó en el año 2000 y es, básicamente, una combinación de la sintaxis de la versión HTML 4.01 y XML añadiendo algunas características específicas de XML. Este lenguaje de marcas XHTML resuelve el problema de la incompatibilidad del HTML 4.01 con las herramientas basadas en XML.

La versión actual es XHTML 2.0 y presenta varios problemas, entre ellos: no se añadió en el estándar nuevas funcionalidades demandadas por los programadores, no ofrece compatibilidad hacia atrás con las páginas web ya existentes, su sintaxis es muy estricta y presenta altas exigencias a los navegadores.

Todo ello, unido a que ya había aparecido el estándar HTML5, hizo que el W3C abandonase el proyecto. Esto no afectó mucho en general, ya que HTML5 acepta todo lo que es admitido en XHTML, aunque no al contrario.

2. Estructura de un documento HTML5

Una página web básica se compone de tres tipos de código:

- HTML5, que aporta los elementos estructurales.
- CSS, que aplica estilo a estos elementos.
- JavaScript, que posibilita la interacción dinámica de esos elementos.

Página web = Estructura HTML5 + Estilo CSS + Interacción JS

En esta unidad se estudia HTML5, así como las hojas de estilo CSS3. JavaScript y la integración en documentos HTML se ven en una unidad posterior.

Toda página HTML5 debe tener un prólogo y un ejemplar, y de esa forma es compatible con cualquier documento XML.

2.1. Prólogo

Un documento HTML5 debe tener una declaración de tipo de documento, en la que se indica que es de tipo HTML. Con esta declaración se activa el modo estándar del navegador web. El navegador sabe el tipo de documento que va a mostrar y las incorporaciones de HTML5 son interpretadas siempre que sea posible, o ignoradas si no las soporta.

```
<!DOCTYPE html>
```

2.2. Ejemplar

El ejemplar es el propio documento HTML5 y está delimitado por las marcas `<html>` y `</html>`.

El elemento raíz o nodo raíz es siempre `<html>`. Hay que añadir el atributo `lang`, que define el idioma del contenido del documento que se está creando, en este caso 'es', por español. Para HTML5 no es necesario.

```
<html lang="es">
```

El ejemplar consta de dos secciones: la cabecera y el cuerpo.

CABECERA

El elemento `head` (cabecera) contiene los metadatos del documento que incluyen información sobre la página, como son el título, la descripción, el autor, etcétera. Se pueden incluir enlaces a otros contenidos para completar la presentación del contenido del documento, como una hoja de estilo asociada o algunos scripts.

El contenido de la cabecera no se muestra en la página.

Por ejemplo:

```
<head>
  <meta charset=utf-8" />
  <title>TEXTO</title>
  <link rel="stylesheet" type="text/css" href="style-
original.css" />
</head>
```

En la cabecera se pueden incluir los siguientes elementos:

- `<meta>` permite incluir metadatos y posicionar la web en los buscadores. Un ejemplo de utilización de la etiqueta `<meta>` es para indicar la codificación, y de esa forma las palabras con acentos no mostrarán símbolos extraños en el navegador.

```
<meta charset="utf-8" />
```

- `<style>` para la definición de estilos en línea. No es necesaria si se va a utilizar una hoja de estilo externa mediante la etiqueta `<link>`.

```
<style type="text/css">
```

.....

```
</style>
```

- `<title>` descripción que identifica el documento y que se muestra en la pestaña de título del navegador.

```
<title>Mi página de prueba</title>
```

- `<link>` vincula las hojas de estilo como recurso externo. También permite incorporar código Javascript, imágenes o iconos desde archivos externos.

```
<link rel="stylesheet" href="estilos.css" />
```

En este ejemplo, el atributo `rel` tiene el valor `stylesheet`, que le dice al navegador que el archivo `estilos.css` es un archivo CSS con los estilos necesarios para presentar la página.

- `<script>` para incluir un script en la página web. Hay dos opciones para indicar el código del script:

a) código en un archivo:

```
<script src="archivo.js" type="text/javascript"></script>
```

b) código insertado aquí:

```
<script type="text/javascript">Código de un script integrado en la
página</script>
```

El atributo `type` indica el tipo MIME, y que no es necesario en los elementos `style` y `script` en HTML5.

**IMPORTANTE**

El tipo MIME, *Multipurpose Internet Mail Extensions* o extensiones multipropósito de correo de Internet, especifica el tipo de dato del contenido del archivo. Por ejemplo, texto, imagen, audio, etcétera. Cada tipo de dato tiene un sufijo concreto. En el ejemplo anterior para JavaScript es `type="text/javascript"`.

CUERPO

Contiene la información de la página que se verá en el navegador. Está delimitado entre las etiquetas `<body>` y `</body>`.

En el apartado siguiente se estudian los diferentes elementos a incluir en el cuerpo del documento HTML.

**IMPORTANTE**

Para realizar los ejemplos, casos prácticos y, en general, toda la parte práctica del módulo se recomienda instalar al menos otro navegador web diferente del utilizado normalmente.

3. Identificación de etiquetas y atributos de HTML5

Las etiquetas siguen las mismas reglas que en XML, es decir, HTML tiene etiquetas de apertura (<etiqueta>) y de cierre (</etiqueta>), pero a diferencia de XML, que puede crear tantas etiquetas como se quiera o necesite, en HTML las etiquetas están predefinidas por el lenguaje.

Por este motivo, para aquellas páginas que, por la complejidad de alguno de sus componentes (elementos multimedia, enlaces, etcétera), necesitan añadir información para su correcta visualización, es preciso complementar las etiquetas predefinidas con atributos.



Fig. 2.2. Esquema de un elemento HTML.

Los atributos son pares **nombre=valor** escritos dentro de una etiqueta de apertura y detrás del nombre de dicha etiqueta. El *valor* puede estar entre “comillas dobles” o ‘simples’.

Cuando el valor de un atributo no es válido, el navegador lo ignora.

El formato o sintaxis general de una línea de código HTML es:

<etiqueta atributo1="valor1" atributo2="valor2">contenido</etiqueta>

Para un caso concreto, por ejemplo un enlace:

```
<a href="http://www.lmsgi.com" target="_blank">Ejemplo de enlace</a>
```

Donde:

- <a> es la etiqueta de apertura y la etiqueta de cierre. En este caso, el elemento a define un enlace a una página web, pero podría ser un archivo o una dirección de correo.
- href y target son los atributos. href indica la URL.
- http://www.lmsgi.com y _blank son los valores asignados a los atributos. En este caso concreto se está diciendo con target="_blank" que la página de destino hacia la que apunta se abra en otra ventana del navegador.
- El texto Ejemplo de enlace es el contenido.



IMPORTANTE

En HTML, las etiquetas y los atributos están definidos previamente, mientras que en XML los define el programador.

A. Clasificación de los atributos comunes según su funcionalidad

Cada etiqueta HTML5 puede tener sus propios atributos, pero hay algunos que son comunes a varias etiquetas. En general, los atributos se pueden agrupar según su funcionalidad en:

- Atributos básicos.
- Atributos de internacionalización.
- Atributos de eventos.
- Atributos de foco.

A.1 Atributos básicos

Los atributos básicos se pueden utilizar en la mayoría de etiquetas HTML.

Atributo	Descripción
title = "texto"	Establece el título del elemento HTML. El título es mostrado por los navegadores cuando el usuario pasa el ratón por encima del elemento.
id = "texto"	Establece un indicador único a cada elemento.
style = "texto"	Aplica al elemento HTML el estilo "texto" directamente.
class = "texto"	Establece la clase CSS que se aplica a los estilos del elemento.

Tabla 2.2. Atributos básicos de HTML.

A.2 Atributos de internacionalización o *i18n*

Estos atributos se utilizan en páginas que muestran sus contenidos en varios idiomas o en las que se quiera indicar de forma explícita el idioma de sus contenidos.

Atributo	Descripción
lang="codigo"	Indica el idioma del elemento. Ejemplo: lang="es".

Tabla 2.3. Atributos de internacionalización.

La recomendación RFC 1766 se refiere a códigos de países para Internet. El documento original está disponible en este [enlace a la web del IETF](#).

A.3 Atributos de eventos y de foco

Los atributos de eventos se utilizan en páginas web que incluyen código JavaScript para realizar acciones dinámicas sobre los elementos de la página.

Se llama foco (*focus*) cuando un control o elemento del documento ha sido seleccionado.



IMPORTANTE

La tabla completa de atributos de eventos y de foco la puedes encontrar en el archivo **UD2_Atributos_eventos_HTML.pdf** disponible en la sección **Material de trabajo > Descarga de documentos**, situada al final de la unidad.

B. Elementos HTML5

En HTML5 (al igual que XML), además de etiquetas y atributos, se utiliza el término elemento para referirse a las partes que componen un documento HTML.

En general, un elemento HTML consta de:

1. Una etiqueta de apertura.
2. Cero o más atributos.
3. Texto encerrado por la etiqueta. Es variable, ya que no todas las etiquetas pueden encerrar texto.
4. Una etiqueta de cierre. No siempre es necesaria.

En HTML existen dos tipos de elementos contenedores: los elementos en línea y los elementos en bloque.

- Elementos en línea o *Inline elements*. Ocupan el espacio necesario para mostrar sus contenidos. Cada elemento en línea se colocará a continuación del elemento anterior, sin introducir ningún salto de línea. Su contenido puede ser texto u otros elementos en línea. Por ejemplo, los enlaces, el salto de línea y el texto resaltado son elementos en línea. En este [enlace](#) tienes disponible información adicional sobre ellos.
- Elementos en bloque o *Block-level elements*. Definen bloques de contenido coherente, ya sea texto o imágenes. Un elemento en bloque incorpora un salto de línea antes y después del mismo. Por ejemplo, encabezados, párrafos, listas, divisiones o imágenes son elementos en bloque. En este [enlace](#) tienes más información sobre ellos.

Hay elementos que se pueden comportar como elementos en línea o en bloque, en función de las circunstancias.

En general se utiliza:

- `<span>` para elementos en línea.
- `<div>` para elementos en bloque.

Por ejemplo, `<div>`, al poder contener otros elementos HTML, se suele utilizar para diseñar la página web, y para añadir estilo al bloque se utilizan las hojas de estilo.

En el caso de `<span>` se suele utilizar para aplicar estilo al texto o agrupar elementos en línea.

En este [enlace a la web para desarrolladores de Mozilla](#) está disponible la lista de todos los elementos HTML.

B.1 Elementos de la estructura básica del documento

La estructura básica de un documento HTML contiene las siguientes etiquetas:

Elemento	Descripción
html	Las etiquetas <html> y </html> indican la apertura y el cierre de un documento HTML, y no debe añadirse nada antes o después de ellas excepto el DOCTYPE.
head	Las etiquetas <head> y </head> indican la apertura y el cierre de la cabecera del documento y la información que contiene no se muestra en el navegador, a excepción del contenido de la etiqueta <title>.
body	Las etiquetas <body> y </body> delimitan el cuerpo del documento HTML, y su contenido se muestra en el navegador.

Tabla 2.4. Estructura básica documento HTML.

EJEMPLO 1

En la sección **Ejemplos para trabajar y analizar**, situada al principio de la unidad, puedes encontrar el archivo **UN02_AP03_ejemplo_1.html**, con el código correspondiente a algunos de estos elementos básicos de todo documento HTML.

B.2 Elementos que dan formato al texto

La siguiente tabla incluye los principales elementos para dar formato al texto.

Elemento	Descripción
p	Delimita párrafos.
h <i>i</i>	Indica el nivel del encabezado. “ <i>i</i> ” es un valor entre 1 y 6. El tamaño de la letra del encabezado será mayor cuanto menor sea el valor de “ <i>i</i> ”. Solo se utilizan en títulos de párrafos.
br	Salto de línea.
b	Texto en negrita.
i / em	Texto en itálica o cursiva. Mejor utilizar .
u	Texto subrayado.

sup	Superíndice.
sub	Subíndice.
center	Texto centrado.
strong	Texto resaltado. Normalmente será en negrita, pero es posible que algún navegador lo muestre en cursiva y naranja.
blockquote	Es una cita textual y se representa como un párrafo indexado con respecto al margen.
hr	Es una línea horizontal. Se escribe </hr>.
comentario	El texto del comentario no es visible en el navegador. Se escribe <!-- comentario -->.

Tabla 2.5. Elementos que dan formato al texto.

EJEMPLO 2

En el archivo UN02_AP03_ejemplo_2.html puedes encontrar el código correspondiente a algunos de estos elementos de formato del texto. En el navegador Firefox se vería así:

Ejemplo 2: elementos de formato

Muestra un texto en **negrita**, *cursiva* y subrayado.

Un ejemplo de nota, al cual se le añade un sangrado o margen por la izquierda.

Otro encabezado

Ejemplo con superíndices ($E=mc^2$) y subíndices (CO_2).

Fig. 2.2. Ejemplo_2.html, formato de un texto.

HTML5 incluye la etiqueta `<hgroup>`, que se utiliza para agrupar un conjunto de uno o más elementos del tipo (h1-h6).

```
<header>
  <hgroup>
    <h2>Título de la página web</h2>
    <h2>Subtítulo</h2>
  </hgroup>
</header>
```

Observa que ahora se utiliza el elemento `header` y no `head`. La diferencia entre ellos es que `head`

contiene información no visible, tal como metadatos o links, y `header` puede contener logotipos, encabezados `h1`, menús de navegación `nav`, secciones, artículos..., y se incluye en el `<body>`.

B.3 Elementos de listas

En HTML5 se pueden utilizar tres tipos de listas:

- Ordenadas.
- Desordenadas.
- Listas de descripción.

La siguiente tabla describe los elementos asociados a cada una de ellas:

Elemento	Descripción
<code>ul</code>	Delimita los elementos que forman una lista desordenada. Anida elementos <code>"li"</code> .
<code>ol</code>	Delimita los elementos que forman una lista numérica u ordenada. Anida elementos <code>"li"</code> .
<code>li</code>	Indica cada uno de los elementos de una lista.
<code>dl</code>	Delimita los elementos que forman una lista de descripción.
<code>dt</code>	Cada uno de los términos que se definen de una lista de descripción.
<code>dd</code>	Cada una de las definiciones de una lista de descripción.

Tabla 2.6. Elementos de listas.

La lista de descripción sirve para crear un conjunto de elementos con pares concepto-descripción. En ella se especifican varios términos por su nombre y se escribirá una descripción o valor para cada uno de ellos.

En las listas ordenadas, por defecto, se utiliza el orden numérico, pero con el atributo `type` se puede indicar otro estilo de numeración.

Con el atributo `start` se indica el primer número de la lista. Por defecto es 1.

EJEMPLO 3

En el archivo UN02_AP03_ejemplo_3.html puedes encontrar el código correspondiente a los elementos de listas. En el navegador Firefox se vería así:

Ejemplo de lista desordenada: Tareas pendientes

- Hacer la compra
- Ordenar papeles
- Contestar correos

Ejemplo de lista ordenada: Prioridades

1. Contestar correos
2. Hacer la compra
3. Ordenar papeles

Ejemplo de lista de descripción: Detalles

Contestar correos
 Importante el correo pendiente al tutor
 Hacer la compra
 No queda leche, huevos y pan
 Ordenar papeles
 Los apuntes de Lenguaje de marcas

Fig. 2.3. Elementos de listas.

B.4. Elementos de tablas

Los elementos para definir una tabla en HTML5 son los siguientes:

Elemento	Descripción
table	Delimita el contenido de una tabla.
tr	Delimita cada una de las líneas de la tabla. Debe estar dentro de las etiquetas de la tabla.
td	Delimita el contenido de cada celda de la tabla. Debe estar dentro de las etiquetas de una fila.
colgroup	Permite agrupar columnas.
tbody	Permite agrupar líneas de la tabla.
thead	Define la línea cabecera de la tabla.
th	Delimita cada una de las celdas de la cabecera.
tfoot	Define la fila pie de la tabla.

Tabla 2.7. Elementos de tablas.

En el siguiente [enlace puedes consultar más información](#) sobre los atributos de tablas.

EJEMPLO 4

En el archivo **UN02_AP03_ejemplo_4.html** encontrarás el código correspondiente a los elementos de tablas. En el navegador Firefox se vería así:

Menú del comedor

Plato	Opciones
Primero	1. Ensalada 2. Lentejas 3. Macarrones
Segundo	1. Pescado 2. Pollo 3. Arroz
Postre	1. Yogur 2. Fruta

Fig. 2.4. Elementos de tablas.

B.5. Elementos de formularios

El formulario es un mecanismo de petición de información al usuario. Consta de un conjunto de campos, de tipos diferentes, que el usuario rellena desde una página web y envía a la URL indicada. Los elementos asociados a la definición y la gestión de formularios son los siguientes:

Elemento	Descripción
form	Encierra o delimita la definición de todos los campos del formulario.
input	Permite añadir en el formulario cajas de texto para textos cortos.
textarea	Caja de texto para texto largo.
select	Menú desplegable que permite elegir de una lista de opciones que se muestra. Para campos de selección.
option	Delimita las opciones de un menú desplegable que contiene este elemento.
button	Define un botón. Se diferencia de los botones hechos con input en que este elemento permite introducir en el botón otros elementos de HTML, como imágenes.
label	Etiqueta de un campo del formulario.
progress	Representa un avance o progreso en la ejecución de una tarea. Por ejemplo, en la descarga de un archivo.
meter	Es similar a <progress> y se utiliza para representar escalas de medidas conocidas, como la longitud, la masa, el peso, el uso de disco, etc.

Tabla 2.8. Elementos de formularios.

En la etiqueta de apertura <form> se han de indicar los atributos básicos:

- **action=""** Se indica la acción a realizar al enviar el formulario. En general, se indica el nombre de un archivo alojado en el servidor que se encargará de procesar la información. **method=""** (post o get) determina el método de transferencia de las variables. El método "post" envía los datos de forma no visible. El método "get" adjunta los datos a la URL a la que se redirige.
- **enctype=""** Indica el tipo de codificación de la información enviada. En el método "get" no es necesario utilizar este atributo, ya que no se realiza codificación, solo se cambian caracteres especiales como el espacio en blanco. En el método "post" se pueden utilizar los valores:
 - **application/x-www-form-urlencoded**: valor predeterminado y codifica todos los caracteres antes de enviarlos.
 - **multipart/form-data**: requerido al enviar archivos mediante un formulario y no

codifica la información.

- text/plain: no codifica la información y cambia los espacios por el "+".

siguiente código muestra un ejemplo de cabecera de un formulario:

```
<form action="http://www.lmsgi.com/UD2/formulario.php"
method="POST">
<!-- Utiliza codificación por defecto -->
<!-- Campo1 del formulario -->
<!-- Campo2 del formulario -->
<!-- Campo3 del formulario -->
</form>
```

B.6. Otras etiquetas para formularios

Etiqueta	Atributos	Descripción
<input>	name=""	Nombre que se le da al campo. Este nombre no es visible en el navegador, solo se utiliza para diferenciar los campos al enviar la información al servidor. Hace la función de nombre de variable a la que se asigna el valor del campo.
	type=""	Tipo de campo a utilizar. Puede ser de tipo text, password, checkbox, radio, file, hidden, submit, reset.
<label>	for=""	Las etiquetas se utilizan para poner un texto o descripción de los campos de un formulario. Se utiliza la etiqueta <label> con el atributo for para indicar el nombre (atributo name) del campo asociado. Por ejemplo: <pre><label for="campo1">Etiqueta</label> <input name="campo1" type="text"/></pre>
<select>		Se utiliza para seleccionar una o varias opciones de una lista. Las opciones de la lista se indican con la etiqueta <option></option>. El nombre que se verá debe indicarse entre esas etiquetas. En cada option, el atributo value="" permite indicar el valor que se envía con el formulario, y el atributo selected permite indicar la opción seleccionada por defecto. Si no se indica, aparece como seleccionado el primer elemento de la lista.
	name=""	Nombre del campo.
	size=""	Especifica el número de opciones visibles a la vez. Si no se indica nada o se le da valor uno, se presentará como un menú

		desplegable. Para valores mayores de uno aparece como una lista con barra de desplazamiento.
	multiple	Permite seleccionar más de una opción a la vez para el campo.

Tabla 2.9. Otras etiquetas para formularios.

Puedes consultar la lista completa de atributos de `<input>` en el archivo **UD2_Atributos_input_HTML.pdf** que puedes encontrar en la sección **Material de trabajo > Descarga de documentos**.

B.7 El elemento indicador de progreso

El elemento `progress` tiene los atributos siguientes relativos a valores que no se identifican con ningún tipo de unidad:

- `max`: define el trabajo total a realizar por la tarea, la duración de un vídeo, etcétera. El valor por defecto es 1.0.
- `value`: estado del progreso. El valor debe ser ≥ 0.0 y ≤ 1.0 , o el valor especificado en `max`.
- `position`: atributo de solo lectura que representa la posición actual del elemento `progress`. El valor de `position` es igual a `value/max`, o `-1` si no se puede determinar la posición.

Las unidades son arbitrarias, y no se especifican.

```
<progress value="5" max="20">5</progress>
```

EJEMPLO 5

En el archivo **UN02_AP03_ejemplo_5.html** puedes encontrar el código correspondiente a los elementos para la utilización de formularios. En el navegador Firefox se vería así:

Ejemplo de utilizacion de formulario sencillo

Introducción de contraseña

Nombre:

Contraseña:

Fig. 2.5. Utilización de formularios.

En `action` no se ha indicado el programa que procesará la información enviada desde el formulario. Se verá más adelante.

EJEMPLO 6

En el archivo `UN02_AP03_ejemplo_6.html` encontrarás el código correspondiente a un ejemplo de uso de diferentes atributos `type` para `input`. Es un ejemplo muy sencillo y que no es procesado. Simplemente es una muestra de código validado.

EJEMPLO 7

En el archivo `UN02_AP03_ejemplo_7.html` puedes encontrar el código correspondiente a un ejemplo de utilización de formularios. En este ejemplo se ha utilizado una tabla para organizar los elementos dentro del formulario.

Formulario con diferentes tipos de campos

Nombre:

Apellidos:

e-mail:

contraseña:

Sexo ☐ Hombre ☐ Mujer	Profesión ☒ Profesor ☐ Programador ☐ Diseñador gráfico	Aficiones: ☐ Ciencia ☐ Libros ☐ Deportes	Elija la herramienta de diseño web que prefiera: Herramienta web:
--	---	---	--

Indique sus comentarios:

Comentario:

Fig. 2.6. Utilización de formularios con tablas.

En este [enlace del W3C](#) puedes encontrar más ejemplos de formularios.

B.8 Otros elementos

Otros elementos de HTML 5 que podemos utilizar son:

Elemento	Descripción
img	Inserta una imagen en una página web. Es obligatorio utilizar el atributo src para especificar el path o ubicación del archivo de imagen que se quiere insertar. Se puede escribir un texto alternativo con alt y dar dimensiones con los atributos height y width. <pre></pre>
blockquote	Contiene un bloque de texto con sangrado.
q	Contiene una cita y el navegador añade las marcas de citación.
br	Inserta una línea en blanco. No tiene etiqueta de cierre.
div	Permite agrupar varios elementos de bloque (párrafos, encabezados, listas, tablas, otras divisiones, etcétera). Si no se le da estilo a la división, el navegador no muestra nada. No hace falta definir divisiones cuando el grupo es de un solo elemento.
video	Permite incrustar vídeos en la página web. Puede reproducirse con autoplay, en bucle con loop, controlando la reproducción con controls, etc. <pre><video src="video.mp4" width="640" height="480"controls></video></pre>
audio	Permite insertar archivos de audio en diferentes formatos, como mp3 y ogg. También dispone de una interfaz de control sobre la reproducción de este con una API en JavaScript sin necesidad de plugins. <pre><audio id="player" src="archivo.ogg" controls></audio></pre>
canvas	Es un nuevo elemento de HTML5 y proporciona una API para dibujar formas, imágenes, textos, crear gradientes, aplicar transformaciones, etcétera. El llamado lienzo (canvas) es también como un rectángulo en la página web en la que, con scripts de JavaScript, por ejemplo, se puede dibujar.
mark	Permite representar un texto resaltado porque tiene una importancia destacada. <pre><p>Algunos elementos de HTML5 como <mark>aside</mark> tienen una importancia especial por la funcionalidad que introducen.</p></pre>
embed	Facilita un punto de integración en la página para una aplicación externa o un contenido interactivo, como un plug-in.

Tabla 2.10. Otros elementos de HTML 5.



IMPORTANTE

Puedes comprobar la compatibilidad HTML de tu navegador visitando el sitio web <http://www.html5test.com/>. La herramienta devuelve, con todo detalle, el grado de compatibilidad de tu navegador con el estándar HTML5.

En este enlace tienes disponible [una lista más completa de símbolos](#).

4. Etiquetas semánticas

En HTML5 se introduce el concepto de esquema (outline). Dicho esquema consiste en estructurar el contenido de la página en secciones y bloques de contenido.

Hasta HTML4, el contenido se jerarquizaba utilizando los títulos “h1”, y se siguen utilizando en HTML5, pero con algunas de las nuevas etiquetas semánticas se definen de forma explícita nuevas secciones y los títulos “h1” son relativos a cada una de esas secciones. De esta forma desaparece el posible problema de no respetar la jerarquía de títulos “h1”. En concreto, las etiquetas <article>, <section>, <nav> y <aside> definen secciones y reinician los títulos “h1” que contienen.

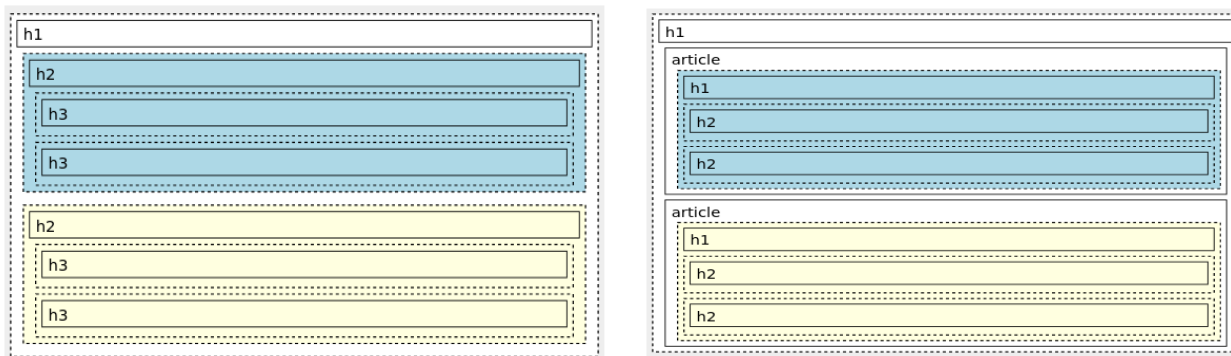


Tabla 2.11. Diferencias jerarquización entre HTML4 y HTML5.

En general, las etiquetas semánticas de HTML5 se utilizan para estructurar los documentos HTML y evitar la necesidad de usar etiquetas <div> (versiones anteriores de HTML) para identificar cada bloque de la página. Utilizando etiquetas semánticas, el navegador al leer la página web, conoce cuál es la estructura de la página y la procesa correctamente.

Estas etiquetas estructurales son: **header, section, article, aside, nav, footer y main.**

Un ejemplo de estructura de una página web en HTML5 utilizando las etiquetas semánticas sería:

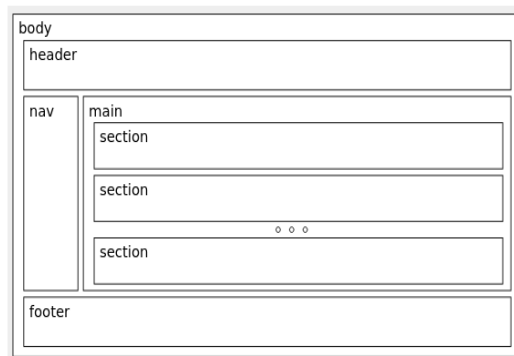


Fig. 2.9. Estructura de página web en HTML5.

La figura siguiente muestra la comparativa entre la estructura de una página en versiones anteriores a HTML5 y con HTML5:

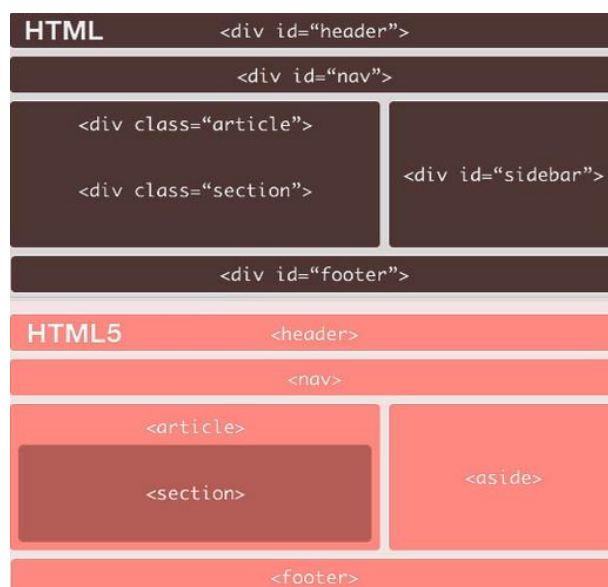


Fig. 2.10. Diferencias de estructura web entre HTML5 y versiones previas.

A continuación, se detalla cada una de estas nuevas etiquetas semánticas.

<header></header>	Contiene un grupo de elementos introductorios (encabezados, logos, menús globales) o de navegación específicos de una sección de la página web o de su totalidad. Por lo tanto, el <code>header</code> no tiene por qué ser único en un documento, se puede incluir un <code>header</code> por cada sección. No hay que confundir con el elemento <code>head</code>, que proporciona los metadatos para el documento. Puede ir dentro de <code>body</code> si es encabezado de toda la página y en <code>section</code>, <code>article</code>, <code>nav</code> o <code>aside</code> si el encabezado es solo para esos elementos.
<section></section>	Identifica las partes o zonas temáticas de los contenidos dentro de un documento o página web. Es decir, <code>section</code>, aunque es genérico, tiene significado semántico y la información que contiene debe estar relacionada de alguna forma. Una sección puede tener subsecciones, y utilizando elementos “hi” se sigue estructurando la página para jerarquizar el contenido, aunque la recomendación de HTML5 es utilizar el esquema (<code>outline</code>). Hasta HTML4 se utilizaba para esta función el elemento <code>div</code>, pero existe una sutil diferencia entre <code>section</code> y <code>div</code>. Cuando se crean bloques cuyo contenido está relacionado se usa <code><section></code>, y cuando estos bloques responden a diseño o necesidades de estructura se usa <code><div></code>.
<article></article>	Define un contenido de un tema específico dentro de una página web. Su característica principal es que puede distribuirse de forma independiente. Se utiliza en mensajes de foro, comentarios enviados por los usuarios, artículos de periódicos, revistas y blogs. Es decir,

	contenidos que pueden publicarse o imprimirse de forma independiente. Una página web puede contener uno o varios artículos de temas relacionados o no. También se pueden anidar. Un elemento <code>article</code> puede contener secciones, y viceversa.
<code><aside></aside></code>	Agrupar contenido separado del contenido que acompaña. Suele aparecer al lado del contenido. Ejemplo típico son los anuncios que aparecen en los laterales, contenidos en barras laterales, citas, enlaces externos, etcétera. Puede estar dentro de <code>body</code>, en cuyo caso es contenido secundario en relación con la página web completa. Si va dentro de <code>section</code> o <code>article</code> es contenido secundario en relación con el elemento en el que esté incluido. Es el contenido propiamente, y su relación con otros elementos, lo que lo define.
<code><nav></nav></code>	Agrupar enlaces de navegación a las diferentes secciones de la página web o a otras páginas. Una página web puede contener varios <code>nav</code>, y su significado dependerá del elemento <code>header</code>, <code>body</code> o <code>aside</code> que lo contenga. Sus usos suelen ser: en una barra de navegación principal de la página web, para definir un conjunto de enlaces que apuntan a diferentes partes de la página actual o para una zona de búsqueda utilizada como mecanismo de navegación dentro de la página. Si aparecen enlaces a pie de página web no se utilizará el elemento <code>nav</code>.
<code><footer></footer></code>	Agrupar enlaces hacia otros sitios, secciones de la página o información general de la página, como son el autor, las licencias y el copyright, entre otras. Puede ir dentro de <code>body</code> si es pie de toda la página web y en <code>section</code>, <code>article</code>, <code>nav</code> o <code>aside</code> si es pie solo para esos elementos.
<code><main></main></code>	Establece el contenido principal del documento y que es único. Todo aquello que se repite en todas las páginas del sitio web (menús, pie de página, barras laterales, etcétera) no va en <code>main</code>. No tiene que haber más de un <code>main</code> por página y no puede ir dentro de <code>header</code>, <code>article</code>, <code>aside</code>, <code>footer</code> y <code>nav</code>.

Tabla 2.12. Etiquetas semánticas.

Es importante tener en cuenta que, utilizando las etiquetas semánticas, cada sección tiene su propio esquema de documento y no se necesitan los `div`, aunque se pueden seguir utilizando. Por otra parte, la jerarquía de títulos es relativa a la sección o artículo. De esa forma, además, se puede mover una sección completa a un documento diferente manteniendo el mismo esquema.

En el siguiente enlace tienes [información referente a las etiquetas semánticas](#).



EJEMPLO 1

En el archivo `UN02_AP04_ejemplo_1.html` encontrarás el código correspondiente a un ejemplo de utilización de etiquetas semánticas.

Ejemplo 8 Uso de etiquetas semánticas

Este es el header de la sección

Esto es un título h1 dentro del header del article 1

Esto es un texto en un párrafo 1

Esto es el pie del article 1

Esto es un título h1 dentro del header del article 2

Esto es un texto en un párrafo 2

Esto es el pie del article 2

Fig. 2.11. Uso de etiquetas semánticas.



IMPORTANTE

Las etiquetas semánticas no definen un estilo predeterminado.

Las etiquetas semánticas no tienen un estilo predeterminado que el navegador vaya a asignar directamente. Esto quiere decir que, aunque header significa 'cabecera' no por eso el navegador va a colocar el elemento en la parte superior del documento. O en el caso de footer, no lo colocará en la parte inferior.

Es el desarrollador quien tiene que diseñar o establecer el estilo que se quiera aplicar a los elementos del documento HTML. Y esto lo hace mediante el uso de CSS.

5. Hojas de estilo CSS

CSS puede ser utilizado tanto en HTML en general (HTML5 en particular) y en XML.

El W3C (*World Wide Web Consortium*) es el organismo encargado de formular las recomendaciones para las hojas de estilo y que son un estándar para los navegadores.

En resumen, las ventajas de utilizar CSS son:

- Control centralizado del aspecto de todas las páginas del sitio web al mismo tiempo. Facilita la actualización.
- Los usuarios pueden configurar su propia hoja de estilo, ya que el navegador lo permite y así se mejora la accesibilidad.
- Una misma página puede vincularse a hojas de estilo diferentes que puede elegir el usuario en función de sus necesidades.
- El uso de hojas de estilo simplifica y reduce el tamaño del código HTML.
- Una misma hoja de estilo puede ser utilizada en diferentes sitios web.

La versión actual de CSS es la 3, CSS3 que, entre otras opciones, incluye soporte para selectores adicionales, sombras, esquinas redondeadas, múltiples fondos, animaciones y transparencias.

A. Soporte de CSS en los navegadores

No todos los navegadores soportan por igual el uso de hojas de estilo CSS. Por este motivo es muy importante para el diseñador de la página conocer si el navegador que utiliza el cliente soporta dichas hojas de estilo. Cada navegador utiliza su motor para interpretar el código HTML y CSS y mostrar las páginas correctamente. La versión de este motor es la que le dirá al diseñador si su diseño se verá como se espera o no.

La siguiente tabla muestra el soporte de CSS3 de los cinco navegadores más populares:

Navegador	Motor	CSS3
Google Chrome	WebKit	Todos los selectores, pseudoclases y muchas propiedades.
Microsoft Edge	WebView2	Todos los selectores, pseudoclases y muchas propiedades.
Firefox	Gecko	Todos los selectores, pseudoclases y muchas propiedades.
Safari	WebKit	Todos los selectores, pseudoclases y muchas propiedades.
Opera	Presto	Todos los selectores, pseudoclases y muchas propiedades.

Tabla 2.13. Soporte de CSS en los navegadores.

Para conocer el soporte del navegador a cada uno de los selectores de CSS se puede utilizar la herramienta online llamada [caniuse](https://caniuse.com/). Este software es de código abierto.

Tiene una herramienta de búsqueda que, al introducir una propiedad, parámetro o característica dice exactamente qué navegadores la soportan y qué versiones. Además, esa compatibilidad la expresa también en porcentaje, ya que podría ser parcialmente compatible.

B. Cómo incluir CSS en un documento HTML5

Las hojas de estilo se pueden añadir al código HTML de tres formas:

1. In line (estilo en línea). Utilizando el atributo `style` en los diferentes elementos HTML.
2. Estilo interno. Utilizando la etiqueta `<style>` en la sección de cabecera `<head>`.
3. Estilo externo. Utilizando un archivo CSS.

ESTILO EN LÍNEA

El estilo en línea ya se ha utilizado en apartados anteriores y solo permite aplicar un estilo a un solo elemento HTML. Ejemplo:

```
<p style="color:blue">Esta es mi página</p>
```

Mostrará el texto “Esta es mi página” en azul. En este caso, `style` se define como atributo del párrafo. Esta forma de aplicar estilo se utiliza cuando hay que definir pocos estilos o se necesita incluir estilos concretos que complementen el estilo global de la página.



EJEMPLO 1

En el archivo `UN02_AP05_ejemplo_1.html` puedes ver un ejemplo de utilización del estilo en línea.

ESTILO CSS INTERNO

El estilo interno permite aplicar un solo estilo a toda la página web. Se define en la etiqueta `<head>` de la página web utilizando la etiqueta `<style>`.



EJEMPLO 2

En el archivo `UN02_AP05_ejemplo_2.html` puedes ver un ejemplo de utilización del estilo CSS interno.

ESTILO CSS EXTERNO

En este caso, el archivo de estilo creado puede ser utilizado en diferentes páginas HTML. El archivo puede crearse con cualquier editor de texto y debe tener extensión `.css`.

Se puede decir que ahora sí que existe una separación real entre el código HTML y el estilo CSS.

La vinculación a una hoja de estilo externa se incluye en la sección <head>. Por ejemplo:

```
<head>
  <meta charset=utf-8" />
  <title>Mi pagina de prueba</title>
  <link rel="stylesheet" href="estilos.css" />
</head>
```

Para el ejemplo anterior, el contenido del archivo de estilos.css sería simplemente:

```
p {color: blue;}
```

Los tributos de link son:

- `rel` Indica el tipo de relación existente entre el archivo que se referencia con la etiqueta `link` y el documento HTML donde se carga. En el ejemplo se indica `stylesheet` porque la relación es de hoja de estilos, es decir, el archivo que se referencia es una hoja de estilos y se aplica al documento HTML.
- `href` Indica la URL absoluta o relativa del archivo externo referenciado.
- `type` indica el tipo MIME del archivo referenciado. En este caso, y al ser una hoja de estilos, será `type="text/css"`, pero HTML5 no lo necesita ya que, por defecto, una hoja de estilo es `text/css`.

C. Sintaxis CSS

Una hoja de estilo consta de uno o más estilos, y cada uno de ellos se llama regla.

Existen dos tipos de reglas:

- reglas @
- reglas, propiamente

Las reglas @ siempre comienzan con el símbolo de arroba @ junto a un identificador y acaban con el símbolo punto y coma (;). El esquema es el siguiente:

regla-arroba identificador punto y coma

```
@ import url("estilo.css");
```

Fig. 2.12. Regla CSS arroba @.

En este ejemplo, el identificador incluido `import` obliga a que esta regla esté siempre al comienzo del archivo CSS, y permite cargar uno o varios archivos CSS dentro de otro. Se pueden utilizar otros identificadores como `name` o `media`, entre otros.

El archivo de estilos puede contener muchas reglas @, pero hay que tener en cuenta que cada vez que se encuentra una regla @ se ejecuta una petición al servidor y se puede ralentizar el proceso de carga de la página, así como de trabajo al servidor.

Cada regla normal puede contener varios selectores y declaraciones, y cada declaración puede contener otras declaraciones. El archivo de estilo puede contener muchas reglas de este tipo.

Resumiendo, cada regla consta de:

- **Selector**, para indicar a qué elemento/s HTML se aplica dicha regla.
- Una **llave de apertura** {.
- **Declaración**, para establecer los estilos concretos que se aplican a los elementos y que consta de:
 - **Propiedad**, para modificar el aspecto o la característica de un atributo del elemento.
 - **Valor**, para establecer el nuevo valor del atributo modificado o propiedad.
- Una **llave de cierre** }.

En el ejemplo anterior:

- `p {color: blue;}` `p` es el selector
- `color: blue` es la declaración; en concreto, `color` es la propiedad y `blue` es el valor.

El esquema es el siguiente:

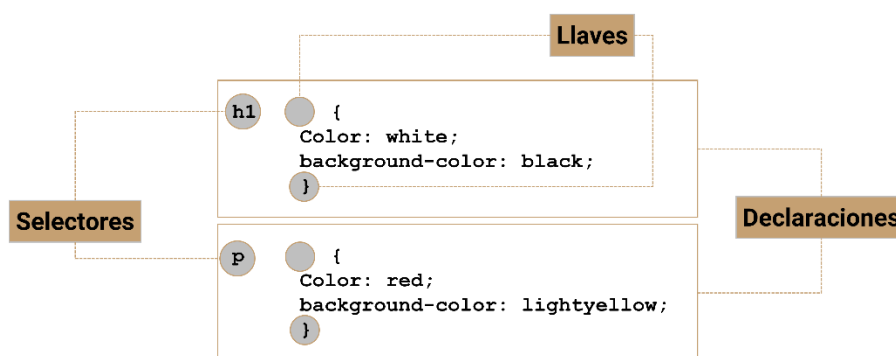


Fig. 2.13. Esquema de una regla CSS.

No se deben repetir los nombres de selectores, para evitar confusión.

Una declaración puede contener varias propiedades, que van separadas entre sí por un punto y coma (;).

Entre la propiedad y su valor se deben poner dos puntos (:).

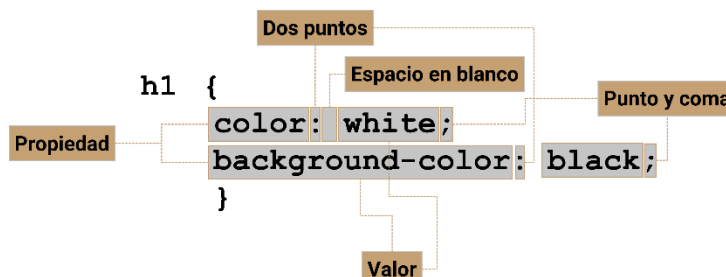


Fig. 2.24. Esquema de la declaración.



IMPORTANTE

Los signos de llaves { }, dos puntos (:) y el punto y coma (;) pertenecen a la sintaxis CSS, y se deben escribir.

Las unidades de medida que se pueden utilizar en CSS son de dos tipos:

Absolutas	- milímetros (mm). - centímetros (cm), un cm son 10 mm. - puntos (pt), un punto son 1/72 in - pulgadas (in), una pulgada son 2,54 cm.
Relativas	- element (em) hace referencia al tamaño en puntos de la letra que se está utilizando. Si se utiliza una tipografía de 12 puntos, 1 em equivale a 12 puntos. - pixels (px) es relativo a la resolución del monitor.

Tabla 2.14. Unidades de medida.

D. Propiedades de CSS

Se clasifican en los siguientes grupos:

D.1 Propiedades de color y fondo

La siguiente tabla enumera las principales propiedades de color y fondo usados en CSS.

Propiedad	Descripción y valores
color	Define el color del texto. Lo admiten casi todas las etiquetas de HTML. Valor: es un nombre de color o su valor RGB.
background-color	Define el color de fondo de un elemento. Valor: es un nombre de color o su valor RGB, o la palabra "transparent".
background-image	Coloca una o más imágenes de fondo para un elemento. El valor es un nombre y se referencia con path absoluto o relativo.
background-repeat	Define si ha de repetirse el fondo del documento, y si es así, cómo lo ha de hacer, en horizontal o en vertical. Valores posibles: repeat-x, repeat-y, no-repeat. Si aparecen dos valores, el primero se aplica al eje X (repetición horizontal) y el segundo al eje Y (repetición vertical).
background-position	Define la posición inicial de la imagen de fondo especificada. Puede venir en porcentaje, una medida o valores preestablecidos. Valores posibles: porcentaje, tamaño, o [top, center, bottom] [left, center, right].
background-attachment	Indica si la posición de la imagen de fondo será fija dentro de la pantalla o se desplazará con su bloque contenedor. Valores posibles: scroll o fixed.
background	Define los valores individuales del fondo en una única regla CSS. Valores posibles: background-color, background-image, background-repeat, background-attachment, background-position.

Tabla 2.15. Propiedades de color y fondo.



IMPORTANTE

Aunque es más sencillo utilizar el nombre del color, dado que no todos los nombres están estandarizados, se aconseja utilizar el valor RGB. En el siguiente enlace puedes consultar [información adicional sobre RGB y los valores de los colores más utilizados](#).

D.2 Propiedades de fuente



EJEMPLO 3

En el archivo `UN02_AP05_ejemplo_3.html` puedes ver un ejemplo de utilización de las propiedades CSS de color y fondo usando una hoja de estilos interna.

La siguiente tabla enumera las principales propiedades de formato de fuente en CSS.

Propiedad	Descripción y valores
<code>font-size</code>	Indica el tamaño del tipo de letra (fuente) del elemento. El valor puede ser el tamaño absoluto, relativo, distancia o porcentaje.
<code>font-family</code>	Indica el tipo de letra o fuente del elemento. El valor es el nombre de la familia fuente.
<code>font-weight</code>	Define el grosor del trazo. Valores posibles: normal, bold, bolder, lighter y valores numéricos 100, 200, 300, 400, 500, 600, 700, 800 o 900, que van del más fino al más grueso.
<code>font-style</code>	Permite elegir la inclinación de la fuente. Valores posibles: normal, italic, oblique. La diferencia entre italic y oblique es que la letra oblique no cambia de forma, solo se inclina, y en la letra italic algunas de ellas cambian de forma y se inclinan.
<code>font-variant</code>	Indica si la fuente es normal o mayúsculas pequeñas. Valores posibles: normal o small-caps.
<code>line-height</code>	Establece el espaciado entre líneas de un bloque de texto. Valores posibles: en unidad de longitud, porcentaje o em.
<code>font</code>	Permite definir simultáneamente las propiedades relacionadas con el tipo de

	letra que figuran en la tabla. El orden es importante, aunque no es necesario que aparezcan todas ellas: font-style, font-variant, font-weight, font-size/line-height, font family. Deben ir separadas por espacios. Obligatorias son font-size y font family.
--	--

Tabla 2.16. Propiedades de fuente.



EJEMPLO 4

En el archivo [UN02_AP05_ejemplo_4.html](#) puedes ver un ejemplo de utilización de las propiedades de fuente con CSS interno.

D.3 Propiedades de texto

La tabla siguiente enumera las principales propiedades que se pueden aplicar a un texto.

Propiedad	Descripción y valores
text-decoration	Indica si el texto es normal, subrayado, sobrrayado o tachado. Valores posibles: none, underline, overline y line-through.
text-align	Establece alineación horizontal en el texto. Valores posibles: left, right, center y justify (justificado a derecha e izquierda).
vertical-align	Establece alineación vertical en el texto. Valores posibles: baseline, sub, super, top, text-top, middle, bottom, text-bottom o un porcentaje.
text-indent	Establece la tabulación o sangría del texto. Valores posibles: una longitud o porcentaje.
text-transform	Permite cambiar el texto a mayúsculas o minúsculas. Valores posibles: none (ninguno), capitalize (la primera letra de cada palabra en mayúsculas), uppercase (todas las letras en mayúsculas) y lowercase (todas las letras en minúsculas).
word-spacing	Permite establecer un espaciado entre las palabras de un texto. Valores posibles: tamaño positivo o negativo, se establecen como una longitud y se añaden al espaciado normal.
letter-spacing	Permite establecer un espaciado entre las letras de un texto. Valores posibles: tamaño positivo o negativo, se establecen como una longitud y se añaden al espaciado normal.

Tabla 2.17. Propiedades de texto.



EJEMPLO 4-BIS

Utilizaremos el mismo archivo del ejemplo anterior `UN02_AP05_ejemplo_4.html` , modificándolo para utilizar propiedades de texto con CSS externo.

Si en lugar de definir el estilo en el propio HTML lo hacemos utilizando un archivo externo CSS, dicho archivo (al que podríamos nombrar como `estilo_4.css`) contendría el siguiente código:

```
h1{  
color:blue;  
text-decoration:underline;}  
p{  
color:black;  
font-size:12px;  
text-align:center;  
text-transform:uppercase;  
word-spacing:20px;}  
ul{  
text-align:left;  
text-indent:20px;  
font-weight:bold;  
font-style:italic;  
font-size:10px;  
color:red;}
```

Y en el archivo HTML del ejemplo 4 tendremos que eliminar esta parte del código e incluir la siguiente línea:

```
<link rel="stylesheet" href="estilo_4.css"/>
```

Tanto el archivo HTML como la hoja de estilo deberán estar en el mismo directorio. Si no es así hay que indicar el path o camino hasta su ubicación.

D.4 Propiedades de caja

El diseño de las hojas CSS se basan fundamentalmente en el modelo de caja. Cada caja ocupa un espacio en la página web, y todo elemento de la página web está contenido en una caja. La figura siguiente muestra la estructura de la caja:

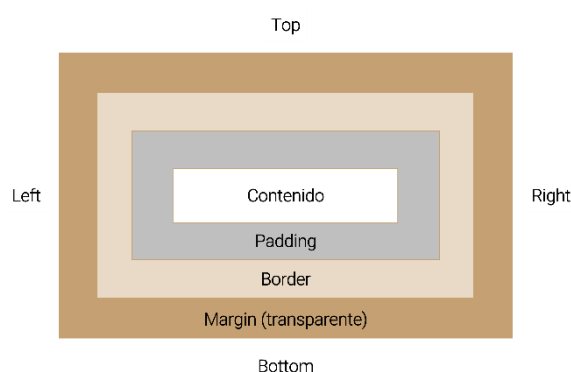


Fig. 2.15. Estructura de una caja.

El detalle de todas las propiedades de las cajas y sus valores está disponible en el archivo **UD2_Propiedades_cajas_CSS.pdf**, en la sección **Material de trabajo>Descarga de documentos**.

En el siguiente enlace se explican en detalle [los márgenes de la caja de texto](#).



IMPORTANTE

El margen exterior (margin) y el margen interior (padding) son transparentes por defecto.



EJEMPLO 5

En el archivo **UN02_AP05_ejemplo_5.html** puedes ver un ejemplo de creación de una caja con las siguientes características: ancho de 450 px, altura de 180 px, color de fondo rojo, borde de 8 px con puntos, color verde, margen de 50 px, relleno de 40 px y que contenga una frase en medio que diga: "Mi primera CAJA" con el tipo de letra sans serif, tamaño 28 px, color azul y en mayúsculas. La hoja de estilo está en el archivo **UN02_AP05_ejemplo_5.css**.

D.5 Propiedades de clasificación

Las propiedades de clasificación influyen en la forma de representar elementos de clasificación HTML, como son las listas.

La siguiente tabla describe las principales propiedades de clasificación y sus valores posibles.

Propiedad	Descripción y valores
display	Indica si el elemento es de bloque, línea, lista o ninguno de ellos. Valores posibles: block, inline, list-item o none.
white-space	Especifica el modo en que se tratarán los espacios en blanco. Valores posibles: pre (se representan todos los espacios), nowrap (la línea no se cortará al llegar al extremo de la página.), normal (si hay varios espacios seguidos, se representarán como uno solo).
list-style-type	Selecciona la marca o viñeta que acompaña a un elemento de lista. Se aplica a los elementos que tengan definido su atributo display como list-item. Valores posibles: disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none.

list-style-image	Selecciona la imagen que se usará como marca de elemento de lista. Se aplica a los elementos que tengan definido su atributo display como list-item. El valor que toma es el path del archivo imagen .
list-style-position	Indica si la marca de lista aparece dentro del cuadro al que pertenece o aparece fuera del cuadro. Se aplica a los elementos que tengan definido su atributo display como list-item. Valores posibles: outside o inside.
list-style	Establece de una única vez todas las características de una lista. Hay que seguir el orden siguiente: list-style-type, list-style-position y list-style-image.

Tabla 18. Propiedades de clasificación.



EJEMPLO 6

En el archivo **UN02_AP05_ejemplo_6.zip** encontrarás un ejemplo de propiedades de clasificación utilizando CSS externo. La hoja de estilo está en el archivo **UN02_AP05_ejemplo_6.css**

E. CSS de posicionamiento

Los navegadores crean una caja para representar cada elemento de la página HTML y la posicionan de forma automática en ella, pero desde CSS se puede modificar dicha posición.

Existen tres formas de posicionar una caja:

- Posicionamiento normal o estático. Es el utilizado por defecto en los navegadores. Los elementos se muestran en la pantalla en el mismo orden en que se encuentran en el código fuente. Hay que conocer si el elemento es de bloque o en línea, sus propiedades `width`, `height` y su contenido.
- Posicionamiento flotante. Los elementos se muestran en la pantalla en el mismo orden en que se encuentran en el código fuente, pero pueden estar desplazados a derecha o izquierda de la posición en la que inicialmente estaba. Utiliza la propiedad `float`.
- Posicionamiento absoluto. Se utiliza para establecer de forma absoluta o exacta la posición en la que se muestra la caja de un elemento. Los elementos se pueden mostrar en pantalla en orden diferente al del código fuente. Utiliza la propiedad `position`.

E.1 Posicionamiento absoluto

El posicionamiento absoluto de un elemento en la caja se controla con la propiedad `position`, que puede tomar los valores:

- **relative**: posicionamiento absoluto relativo. Es como el posicionamiento normal, pero luego la caja se desplaza y se controla con las propiedades `top`, `right`, `bottom` y `left`.
- **fixed**: el posicionamiento absoluto es fijo cuando la caja es posicionada de forma fija, es decir, inamovible dentro de la ventana del navegador. En ese caso, las cajas no modifican su posición, aunque el usuario mueva la página que muestra el navegador.
- **absolute**: el posicionamiento absoluto es absoluto cuando la caja se sitúa en la posición exacta que se le da. En ese caso, las cajas sí que pueden modificar su posición si el usuario mueve la página que muestra el navegador.

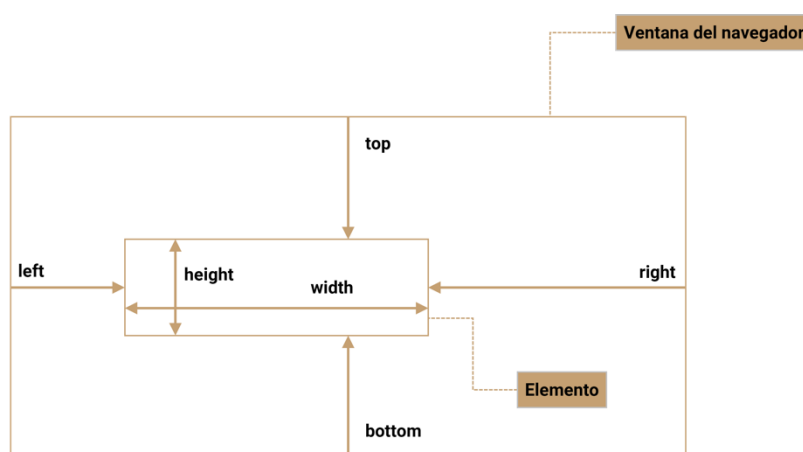


Fig. 2.36. Análisis de una caja.

En el posicionamiento absoluto, en cualquiera de las tres modalidades, las cajas se pueden superponer. Con la propiedad `z-index` se puede controlar qué caja va encima o debajo.

El valor de la propiedad `z-index` es un número entero (>0 , <0 , 0). Valor cero equivale a no tener `z-index`. Cajas con valores mayores de `z-index` se ponen encima de las que tengan valores menores.

E.2 Otras propiedades

La siguiente tabla describe un conjunto de propiedades de las cajas.

Propiedad	Descripción y valores
height	Permite establecer la altura de un elemento. Los valores que puede tomar son: auto o un tamaño o porcentaje.
width	Permite establecer la anchura de un elemento. Los valores que puede tomar son: auto o un tamaño o porcentaje.
visibility	Indica si el elemento sobre el que actúa será visible o no. Valores posibles: - hidden (esconde un elemento, pero deja vacío el espacio donde debería aparecer) - collapse (esconde filas o columnas de una tabla).
left	Indica la posición del lado izquierdo del elemento. Los valores que puede tomar son: auto o un tamaño o porcentaje.
top	Indica la posición del lado superior del elemento. Los valores que puede tomar son: auto o un tamaño o porcentaje.
overflow	Indica si el elemento será visible o no en caso de superar los límites del contenedor. Los valores que pueden tomar son: - visible (se verá aunque sobrepase). - hidden (se recorta si no cabe). - scroll (se recorta pero muestra barras de desplazamiento). - auto (depende del navegador que aparezcan barras o no).

Tabla 2.19. Otras propiedades de las cajas.



EJEMPLO 7

En el archivo [UN02_AP05_ejemplo_7.html](#) encontrarás un ejemplo de propiedades de posicionamiento utilizando CSS interno.

Ejemplo de propiedades CSS de posicionamiento



Fig. 2.17. Ejecución del ejemplo 7.

F. Definición y uso de clases

Cuando se quiere aplicar estilos a un solo elemento de la página HTML se puede utilizar el atributo `class` de HTML sobre ese elemento y así enlazar directamente con la regla CSS que se le debe aplicar.

La clase define un estilo que, en principio, no está asociado a ninguna etiqueta HTML, pero que es susceptible de asociar a etiquetas concretas.

El atributo `class` se define mediante una lista de elementos separados por espacio, y uno de esos elementos debe coincidir exactamente con el nombre de clase dado en el selector.

En la declaración del selector CSS se incluye el nombre de clase precedido de un punto (.). Así, el selector de identificador aplica el estilo al elemento HTML cuyo identificador sea ese nombre.

Recordar que los selectores se utilizan para seleccionar los elementos a los que se van a aplicar las propiedades declaradas en el código CSS.

Por ejemplo, se define así una clase para el color rojo:

```
.clase_rojo{color:red}
```

Si se quiere que el título "h1" se muestre en rojo, se asocia en el archivo HTML el elemento `h1` a dicha clase de la forma:

```
<h1 class="clase_rojo">Cabecera h1 en rojo</h1>
```

Si solo se quiere utilizar esta clase con el elemento HTML de párrafo, en la declaración de la clase hay que añadir el selector `p`, y queda:

```
p.clase_rojo{color:red}
```



IMPORTANTE

Las clases se pueden utilizar varias veces dentro del mismo documento. Si solo se utiliza una vez se puede emplear el atributo `id`.



EJEMPLO 8

En el archivo `UN02_AP05_ejemplo_8.html` encontrarás un ejemplo del uso de clases utilizando CSS interno.

G. Frameworks para CSS

Los frameworks CSS son herramientas de diseño web que facilitan la tarea de crear y editar hojas de estilo en cascada. Facilitan el trabajo del desarrollador/diseñador y le proporcionan una colección de hojas de estilo que se pueden adaptar a sus necesidades.

De ellas, las más importantes o conocidas son las siguientes:

- [Bootstrap](#).
- [Bulma](#).
- [Foundation](#).
- [Materialize CSS](#).

Muchas de ellas utilizan un diseño responsive, es decir, adaptan el diseño de la página al dispositivo que la visualiza.

6. Validación de código HTML y CSS. Herramientas

Existe software descargable o servicios online que facilitan la tarea de validación haciéndola más eficiente, rápida y fiable.

Un servicio de validación online para HTML es el que proporciona el [W3C](#). La página dispone de tres pestañas en función de si se da la URL de la página a validar, se sube el archivo o, si la página aun no es pública, se copia y se pega el código.

El resultado puede ser de dos tipos: anotaciones (verde) y advertencias (amarillo) o errores propiamente (rojo).

Una vez se han realizado los ajustes necesarios hay que comprobar de nuevo.

Para validar hojas de estilo, estando en el [sitio de W3C](#) hay que seleccionar la opción para hojas CSS.

Otra opción en castellano es el enlace de [W3C siguiente](#). El aspecto de la página es el siguiente:



Fig. 2.18. Validación de CSS del W3C.

Desde Visual Studio Code también se puede realizar la validación mediante la extensión HTMLHint. Desde la pestaña “Extensions” de Visual Studio Code hay que buscar esta extensión e instalarla.

Una vez instalada, para ver los posibles errores detectados ir al menú: Ver → Problemas. De cualquier forma, la validación que hace Visual Studio Code no es completa, pero sí que puede ayudar al desarrollador a detectar algunos problemas.

Existen muchos otros validadores que de forma gratuita y online permiten validar el código HTML y/o CSS, como por ejemplo:

- <https://www.freeformatter.com/html-validator.html>
- <https://html5.validator.nu/>
- <https://www.htmlhelp.com/tools/csscheck/>

Otros tipos de validadores

- Validador HTML del W3C para móviles: <http://validator.w3.org/mobile/>

- Existe también un validador o comprobador de enlaces del W3C. Esta herramienta comprueba todos los enlaces que contiene la/s página/s, indicando aquellos que no funcionan. El enlace es: <http://validator.w3.org/checklink/>.
- Por último, existe también un validador de feeds del W3C que permite validar archivos de fuentes RSS. Disponible en <http://validator.w3.org/feed/>. Se verá en otra unidad.

Herramientas

Existe una serie de herramientas que facilitan al desarrollador la tarea de generar y optimizar código HTML y estilo CSS. Algunas de estas herramientas vienen incluidas en el propio navegador y ayudan a depurar el código. Es el caso de las devtools, que para el navegador Firefox están disponibles desde el menú de la aplicación



Fig. 2.21

Desplegado este menú de la aplicación, ir a Desarrolladores web → Herramientas para desarrolladores. En la parte inferior de la ventana de Firefox se muestra un menú con una serie de opciones, como el inspector de estilos y DOM, una consola, un depurador, un editor de estilos, medidor de rendimientos, ocupación de memoria, etcétera. También se puede acceder con el atajo de teclado CTRL + May + I.

En otros navegadores, el procedimiento de acceso a estas herramientas puede ser otro. La imagen siguiente muestra su aspecto para el código de la página visitada:

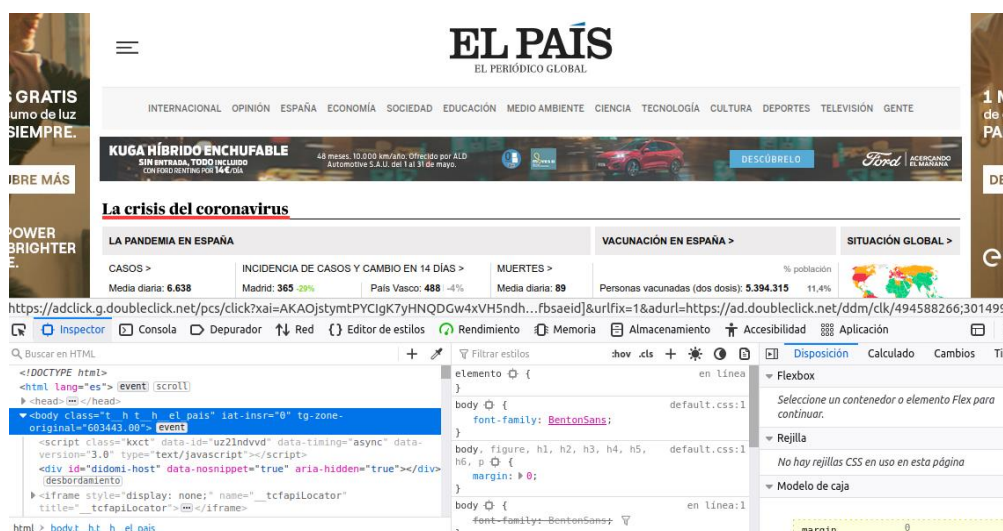


Fig. 2.19. Consola del navegador Firefox.

Existe otro tipo de herramientas más especializadas para ejecutar el código y comprobar si funciona como se espera en diferentes navegadores. Más información sobre estas herramientas en el siguiente [enlace](#).

Este punto de la unidad 2 es también válido para la unidad en la que se describe JavaScript.



Unidad 3.

Aplicación de los lenguajes de marcas a la sindicación de contenidos



1. Sindicación de contenidos

Se define como canal de contenidos o feed el conjunto formado por los servicios ofertados por el sitio original y sus metadatos asociados.

El contenido se distribuye a través de los canales, y los usuarios pueden leer esos canales con un software específico llamado agregador o lector de canales (en inglés, FeedReader).

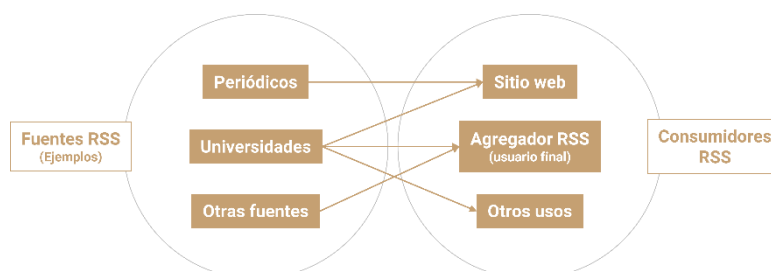


Fig. 3.1. Elementos de la sindicación.

Además de los productores de contenidos o fuentes y los consumidores existe también la figura del prosumidor o prosumer. Este término aparece con la web 2.0 y hace referencia al consumidor que también produce contenidos.

La figura muestra los elementos que forman parte de la sindicación: la fuente u origen de los contenidos, el usuario consumidor de los contenidos y el protocolo de transferencia, y aparece otro elemento básico en la sindicación que es el agregador. El agregador es el que permite llevar a cabo la suscripción a un canal de contenidos o feed y su lectura.

El lenguaje más utilizado en el proceso de sindicación es el XML, y HTTP (Hypertext Transfer Protocol) es el protocolo web utilizado para la transferencia por la red.

A. Características

Una de las muchas ventajas de Internet es la facilidad con la que un usuario puede publicar y compartir contenido. En ese sentido, un sitio web puede ser visto como un flujo de información. Para ello, en la cabecera del sitio ha de incluir, debajo del elemento `title`, el link al canal de contenidos que ofrece. La sintaxis cambia en función de la tecnología utilizada, RSS o Atom, y que se explica más adelante.

```
<link rel="alternate" type="application/rss+xml"
title="título_del_enlace" href="http://localhost/UD3/archivo.rss"
/>

<link rel="alternate" type="application/atom+xml"
title="título_del_enlace" href="http://localhost/UD3/archivo.atom"
/>
```

Esos contenidos a compartir y almacenados en repositorios deben sufrir una transformación para poder ser o bien visualizados (HTML) o bien transferidos al usuario (RSS/Atom).

Los archivos .rss y .atom son los que contienen toda la información necesaria para la transmisión, y están en formato XML.



Fig. 3.2. Logo de RSS.

Pero en la actualidad la sindicación de contenidos también se aplica, bien o mal, al hecho de republicar el contenido, por ejemplo, de un blog, en sitios web de amigos, conocidos o terceros en general.

B. Ventajas y desventajas de la sindicación de contenidos

Las ventajas de utilizar los canales de contenidos son las siguientes:

Ventajas de la sindicación
• Disponer de la información actualizada. • Ahorro de tiempo, al no tener que acceder a todos los sitios web. • El usuario evita correos electrónicos y con ello elimina posible spam o malware. • Facilita la fidelización de los usuarios. • Fácil suscripción y confidencialidad. El uso de este servicio no solicita datos personales. • La cancelación de la suscripción a un canal determinado no requiere el envío de correo de anulación. • Ocupa poco espacio, y se puede configurar la caducidad de la información. • Aumenta el tráfico del sitio web que oferta los contenidos, y por lo tanto su visibilidad. • Ayuda a mejorar el posicionamiento del sitio web en los buscadores. • Si se dispone de un agregador o lector y el usuario lo abre, los contenidos se descargan automáticamente, y puede acceder a ellos cuando lo decida. • Ayuda en la actualización profesional de los suscriptores.

Tabla 3.1. Ventajas de la sindicación.

Hacemos especial mención a dos ventajas que tienen mucha importancia en la actualidad, relacionadas con beneficios comerciales, y son las siguientes:

Obtener tráfico de referencia
Si el sitio web que publica un contenido dice a sus “lectores” cuál es la fuente original de este y la enlazan, el usuario puede acceder a ella directamente, y se obtiene mayor publicidad, mayor visibilidad y, posiblemente, un beneficio económico.

Construir una marca

Si un usuario está expuesto a una marca con mucha frecuencia es muy probable que acabe comprando sus productos. Si un sitio web de prestigio muestra esta marca se produce un efecto de expansión. Todo ello ayuda a construir la marca.

Tabla 3.2. Ventajas comerciales.

Pero la sindicación también tiene desventajas, puesto que puede favorecer:

- El plagio de contenidos.
- Descuidar la comprobación de la veracidad de la información y la fiabilidad del autor.
- Respecto al autor, perder el control y los usos de sus contenidos.
- Respecto al suscriptor, sobrecarga de información.

C. Ámbitos de aplicación

La sindicación de contenidos apareció como un pilar fundamental de la web 2.0, que surgió en 2004, y es la que da, por ejemplo, soporte a los blogs como generadores de información. Pero su utilización no se limita solo a ese ámbito.

Como ya se ha estudiado en unidades anteriores, un objetivo fundamental del lenguaje de marcas XML es precisamente transferir y compartir información. La sindicación utiliza formatos XML para ofrecer contenidos propios de diferente tipo, como son texto, audio, vídeos e imágenes. De esta forma ganan todos; el que ofrece estos contenidos, porque aumenta el valor de su sitio web, tiene más visitas recurrentes y con ello está fidelizando a sus suscriptores. Y el que se suscribe, porque está permanentemente actualizado en temas de su interés.

En general, se puede decir que la sindicación se puede utilizar para recibir, entre otras:

- Novedades en sitios web de interés para el suscriptor.
- Últimas noticias relativas a medios de comunicación.
- Últimas novedades en determinadas materias o como complemento del currículo en el ámbito educativo.
- Información promocional de comercio electrónico.
- Información sobre podcasts y videocasts, incluida su descarga.
- Comentarios en un foro.

Respecto a los blogs, se utiliza una forma de sindicación de contenidos cuando el contenido de este se vuelve a publicar en un sitio web de terceros. Es algo similar al GuestBlogging (participar como invitado en un blog ajeno), es decir, como una especie de intercambio de intereses, pero no es equivalente. Es decir, el sitio web original gana más visibilidad, y el sitio web de terceros gana en tráfico con un contenido que se supone que es bueno.

Las condiciones de publicación establecerán cómo se hace esa republicación, si es completa tal y como aparece en el blog o si solo se publica una parte, pero siempre con un enlace al sitio original, en este caso al blog.

Por otro lado, no hay que confundir la sindicación de contenidos con el contenido duplicado.

Desde el punto de vista de los buscadores, contenido duplicado es “cualquier texto que se repite total o parcialmente en más de un dominio (contenido duplicado externo), y cuando se detecta se puede penalizar el posicionamiento SEO”. Y, a diferencia del contenido duplicado, la sindicación correcta puede ayudar a aumentar el alcance del sitio web y mejorar su posicionamiento SEO (Search Engine Optimization, optimización para motores de búsqueda).

También puede aparecer contenido duplicado en el propio dominio, y se dice que es interno. En este caso, el problema suele estar provocado por un mal mantenimiento del sitio web o una mala gestión de la creación de categorías en el sitio y etiquetas; por ejemplo, en blogs muy extensos y con muchos artículos.

2. Tecnologías de creación de canales de contenidos

RSS (Really Simple Syndication) es un formato de estructuración de datos basado en XML que facilita el acceso automatizado a la información contenida en un sitio web.

RSS es una de las tecnologías más utilizadas en la creación de canales de contenidos. Las siglas RSS hacen referencia a tres estándares diferentes:

RSS 0.91 Rich Site Summary	Fue creado por Netscape en 1999 (empresa que creó el navegador web gratuito Netscape en 1994) y utiliza la especificación RDF de metadatos. Su objetivo era que el proyecto My Netscape tuviera titulares de otras webs.
RSS 1.0 RDF Site Summary	Desarrollado por el grupo de trabajo RSS-DEV a partir del estándar RSS 0.90. Es más estable y permite definir una cantidad mayor de datos que el resto de versiones de RSS.
RSS 2.0 Really Simple Syndication	Desarrollado por la empresa UserLand Software, que consideró que la versión RSS 1.0 era compleja y retomó la versión 0.91 para continuar el desarrollo. Publicó versiones 0.92, 0.93 y 0.94, pero la sintaxis de estas está incompleta y no cumplen todas las normas de XML. UserLand vendió este desarrollo a Harvard Law, quien publicó el estándar RSS 2.0 y solucionó esos problemas.

Tabla 3.3. Estándares de RSS.

Estos estándares han sido desarrollados por tres organizaciones diferentes. De ellos solo el formato RDF (RSS 1.0) ha sido creado por la W3C, y es el único organismo que valida sus normas.

Pero de ellos el más utilizado en la actualidad es el RSS 2.0. Esta versión incorpora el uso de los espacios de nombres XML.

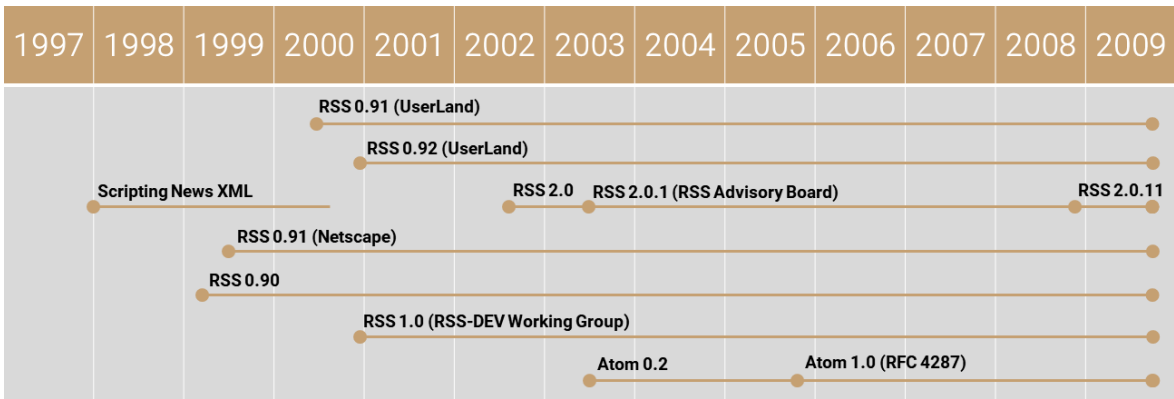


Fig. 3.3. Evolución de RSS.

El archivo RSS simplemente contiene un conjunto de etiquetas definidas que siguen las reglas XML, y en él están los datos de las novedades del sitio, como el título, la fecha de publicación o la descripción. El programa que lee el RSS es el encargado de darle estilo o apariencia a los datos que se incluyen en el archivo y de presentarlos de una manera atractiva al usuario y de fácil lectura.

Se puede utilizar el formato RSS en los dos sentidos:

- Para recibir información desde otros sitios web.
- Para ofrecer información desde la propia web.

Según el usuario sea consumidor o creador de contenidos, elegirá una u otra opción, o también puede utilizar ambas a la vez.

El otro formato o tecnología utilizado en la sindicación es Atom (Atom Publishing Format and Protocol, formato y protocolo de publicación Atom), publicado en el [RFC4287](#) por el grupo de trabajo [Atompub](#) de la [IETF](#) (Internet Engineering Task Force, Grupo de Trabajo de Ingeniería de Internet).

La versión actual de Atom es la 1.0. Atom, y se convirtió en estándar IETF en diciembre de 2005.

Atom también tiene en cuenta los espacios de nombres XML.



Fig. 3.4. Tecnología Atom.

Atom aparece, en principio, como solución a los diferentes estándares RSS existentes y que no son del todo compatibles entre sí. Es un formato que reúne las características comunes de los RSS, está basado también en XML y además permite controlar la cantidad de información que se va a representar en los agregadores.

Pero en la actualidad Atom no ha sustituido a RSS, sino que compite con él y su uso aún es minoritario.

Una característica de Atom es que se puede, además de recibir información, añadirla o enviarla. Para ello, Atom se compone de:

- [Atom Syndication Format](#) para recibir.
- [Atom Publishing Protocol](#) para enviar.

Al final del siguiente apartado se detallan las diferencias entre RSS y Atom.

3. Estructura de los canales de contenido

Es importante no confundir el concepto de fuente web o canal con RSS. Fuente web se refiere al medio de redifusión del contenido web, y RSS se refiere al formato de dichas fuentes web o canales. Mientras solo existía RSS como formato disponible no había problema, pero ahora, al haber más formatos, se presta a confusión.

Estructura del canal

Un canal de contenido o feed consta de un archivo con formato RSS o Atom, ambos basados en XML. El sitio web que ofrece contenidos debe publicar este archivo en un directorio del propio sitio web.

El archivo .rss o .atom debe contener:

- La declaración de documento XML, y especificar el sistema de codificación utilizado. Normalmente es utf-8.
- El elemento canal, que es el que indica cuál es el sitio web asociado a la fuente web, y que estará formado por secciones.
- Las secciones, que son las diferentes referencias a los sitios web que contienen los contenidos o servicios ofrecidos. Un canal puede tener varias secciones, es decir, varios enlaces.

A. ¿Cómo crear un canal o feed?

Se va a suponer que el usuario tiene un sitio web y quiere difundir alguna parte de él.

En general, los pasos que se deben seguir son los siguientes:

1. **Crear un archivo XML** con formato RSS/Atom.
2. En este archivo se **debe especificar**:
 - a. La **versión de RSS/Atom**.
 - b. **Información de nuestro canal**.
 - c. Las **secciones con los enlaces a las páginas web** que se quieren difundir.
3. **Guardar el documento XML** con extensión .rss/.atom .
4. **Validar** el documento.
5. **Colocar el indicador en la página donde se ofrece el canal**; puede ser el icono naranja "RSS" o "XML" o el azul, o bien las siglas, etc.
6. **Subir el documento al sitio web** (junto con el icono), en el servidor web que lo aloje.
7. **Colocar un enlace** (en el icono o en las siglas) que vincula el archivo RSS/Atom.

Por ejemplo, ponemos el icono naranja y enlazamos con el canal RSS:

```
<a href="caso_práctico_3.rss"></a>
```

8. **Enviar el feed RSS/Atom** creado a los directorios de contenidos RSS/Atom.

Recordemos que aunque aparece el archivo con extensión .rss/.atom, en realidad es un archivo XML.

B. RSS

Veamos un ejemplo concreto.

Para crear un canal de contenidos con el formato RSS, los pasos son los siguientes:

1. **Crear un archivo .rss** utilizando un editor de texto.
2. **Añadir la declaración de archivo XML** y la codificación.

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

3. **Indicar la versión de RSS utilizada** mediante el elemento RSS, y que constituye el elemento raíz del documento.

```
<rss version="2.0">
```

4. **Incluir el elemento channel** para crear el canal propiamente, y añadir las secciones que correspondan.

El canal debe contener los elementos:

Elemento	Descripción
title	Título que se da al sitio web. Puede ser cualquier nombre.
link	URL de la página asociada al archivo RSS y que ofrece los contenidos.
description	Descripción de la finalidad del sitio.
language	Idioma utilizado en el sitio web.
item	Son las diferentes secciones del canal.

Tabla 3.4. Elementos del canal.

Cuando el contenido sindicado está dividido en partes que son independientes, cada una de ellas se enlaza con una descripción diferente y en un ítem diferente.

Cada uno de los ítems debe contener los siguientes elementos:

Elemento	Descripción
title	Nombre cualquiera con el que se identifica el título del contenido.
link	Contiene el enlace al contenido.
guid	Contiene un Globally Unique Identifier, que permite identificar una entrada. Los lectores de feeds suelen utilizar esta etiqueta para diferenciar entre elementos

	nuevos y elementos leídos.
description	Descripción del contenido ofrecido en este enlace.
category	Clasifica el artículo en una categoría y se puede usar como elemento único varias veces en una misma etiqueta <code>item</code> .

Tabla 3.5. Elementos del ítem.

El siguiente código muestra las partes o el esqueleto de un archivo con formato RSS.

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel> <!-- Propiedades del canal -->
    <item> <!-- Contenido publicado 1 -->
    </item>
    <item> <!-- Contenido publicado 2 -->
  </item> ...
</channel>
</rss>
```



EJEMPLO 1

Este sería un código RSS concreto:

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
  <channel>
    <title>Ejemplo de canal</title>
    <link>http://localhost</link>
    <description>Este es un archivo RSS</description>
    <item>
      <title>Qué es el RSS</title>
      <link>http://localhost/UD3/define_RSS.html</link>
      <description>Define el estándar RSS</description>
      <guid>http://localhost/UD3/01</guid>
    </item>
    <item>
      <title>Para qué sirve</title>
      <link>http://localhost/UD3/usos_RSS.html</link>
      <description>Explica cuáles son sus usos</description>
      <guid>http://localhost/UD3/02</guid>
    </item>
  </channel>
</rss>
```

Existen otros elementos que pueden aparecer en el canal, aunque de menor utilidad que los anteriores. Por ejemplo, podemos agregar una imagen identificativa enlazada al sitio web fuente. El elemento `image` debe incluirse dentro de `channel`, pero fuera de cualquier `item`.

```
<image>
```

```
<url>http://localhost/UD3/imagenes/rss.png</url>
<link>http://localhost</link>
<title>Ejemplo de canal</title>
</image>
```

Hay que tener en cuenta que el `title` de la imagen debe coincidir con el `title` del `channel`, y lo mismo para el elemento `link`.

También se pueden añadir en el propio RSS posibles derechos de autor utilizando el elemento `<copyright>` y escribiendo el texto que corresponda.

C. Atom

Para construir un canal con el estándar Atom, la secuencia de pasos es similar:

1. Crear un archivo Atom utilizando un editor de texto.
2. Añadir la declaración de archivo XML y la codificación.
3. Definir el canal con el elemento `feed`, e indicar el estándar utilizado y el idioma.

En concreto, en el `feed` hay que incluir los elementos:

Elemento	Descripción
title	Título que se da al sitio web. Puede ser cualquier nombre.
id	Identificador del canal. Generalmente será la URL.
link	Enlaces necesarios que definen el canal: 1. Enlace al propio archivo <code>.atom</code>. En el atributo <code>rel</code> del elemento <code>link</code> hay que indicar "self". 2. Enlace al archivo web que oferta ese canal. En el atributo <code>rel</code> del elemento <code>link</code> hay que indicar "alternate".
updated	Fecha y hora de actualización, en formato YYYY-MM-DDTHH:MM:SSZ, donde T es el separador entre la fecha y la hora y Z indica que la hora es la relativa al meridiano de Greenwich.
author	Indica el autor del canal.
entry	Son las diferentes secciones del canal.

Tabla 3.6. Elementos del `feed` de Atom.

Cada una de las secciones o `entry` debe contener los siguientes elementos:

Elemento	Descripción
title	Nombre cualquiera con el que se identifica el título del contenido.
id	Identificador de la sección. Ha de ser único en el archivo <code>.atom</code> .
link	Enlace a la fuente de contenidos de esa sección. En el atributo <code>rel</code> hay que indicar "alternate".

updated	Fecha y hora de actualización. El mismo formato que para el canal.
author	Indica el autor de la sección o enlace.
summary	Resumen del contenido del enlace.

Tabla 3.7. Elementos de cada entrada.

EJEMPLO 2

El siguiente código muestra las partes de un archivo con formato Atom.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Ejemplo archivo Atom</title>
  <link href="http://localhost"/>
  <updated>2021-10-13T18:30:02Z</updated>
  <author>
    <name>Profesor</name>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
  <entry>
    <title>Ejemplo archivo Atom</title>
    <link href="http://localhost/2021/10/13/atom01"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2021-10-13T18:30:02Z</updated>
    <summary>Simplemente un ejemplo de prueba.</summary>
  </entry>
</feed>
```

Una vez se ha creado el archivo con formato RSS o Atom hay que comprobar si es válido o no. Para ello se utilizan los validadores que se verán en el próximo apartado.

D. Diferencias entre RSS y Atom

Aunque el objetivo de ambas tecnologías es el mismo, existen algunas diferencias entre ellas:

Definición	- RSS es una familia de formatos de fuentes web que se utilizan para publicar trabajos que se actualizan con frecuencia. - Atom se basa en dos estándares para las fuentes web.
Campos requeridos en un feed	- RSS es más flexible y no necesita mucha información. Requiere un título, un enlace a su contenido y una descripción. - Atom es más restrictivo y necesita más datos. Requiere la fecha de la actualización, un título y un identificador único.
Tipo de contenido	- RSS: puede contener texto plano sin formato y HTML. - Atom puede contener tipos como texto plano, HTML, XHTML y XML, además de referencias a contenido externo del tipo audio, vídeo y documentos.

Formato de la fecha	Ambos basados en diferentes RFC (Request for Comments). - RSS muestra la fecha y hora de los datos cuando se creó y actualizó la fuente por última vez. Se basa en RFC 822, que es un formato estándar de Internet para cabeceras de mensajes de correo electrónico. - Atom muestra la fecha y hora en que se actualizó por última vez el sitio web. Se basa en RFC 3339, que es un subconjunto de la norma ISO 8601 sobre el formato de fechas.
Internacionalización	Por la forma de especificar el lenguaje: - RSS no permite especificar el idioma para cada elemento o ítem. - Atom utiliza la norma xml: lang para que se pueda identificar el idioma y su contenido sea legible.
Modularidad	- RSS no permite la reutilización de los elementos en otros vocabularios XML. - Atom permite la reutilización de los elementos fuera del contexto de un feed Atom. En los Documentos de trabajo y estudio está disponible el archivo Equivalencia_RSS_Atom.pdf, que contiene más información acerca de los elementos de ambas tecnologías de forma comparativa.

Tabla 3.8. Diferencias entre RSS y Atom.

4. Utilización de herramientas de creación, edición y validación

Para la creación y edición de canales de contenidos, aunque se puede hacer con cualquier editor de textos, tipo notepad++ o gedit , existen herramientas que facilitan la tarea.

Algunas de estas herramientas son:

Atom	Es un editor de código fuente de software libre disponible para Windows, GNU/Linux y MacOS. Desarrollado por GitHub, apareció en 2015 y se basa en una combinación del navegador Chromium, Node.js (entorno de ejecución para JavaScript) y lleva un sistema de control de versiones integrado.
Aptana Studio	Es un entorno de desarrollo integrado basado en Eclipse y desarrollado por Aptana. Es software libre y disponible para Windows, GNU/Linux y MacOS. Con soporte nativo para muchos lenguajes de programación y complementos que lo extienden a otros lenguajes.
PSPad	Es un editor de texto con licencia freeware y orientado a la programación. Con soporte para la mayoría de los lenguajes de programación, entre ellos XML, CSS y RSS. Para sistemas Windows.
Visual Studio Code	Es un editor de código de Microsoft disponible para Windows, GNU/Linux y MacOS. Es gratuito y de código abierto.

Tabla 3.9. Herramientas de edición de canales de contenidos.

En nuestro caso vamos a recomendar Visual Studio Code, que ya se ha utilizado en la unidad 2 y que cubre completamente las necesidades.

Referido a este tema que nos ocupa, Visual Studio Code lleva embebidas extensiones relativas a RSS, como son:

- RSS: un lector de RSS.
- RSS-reader: muestra las fuentes de hasta 10 RSS en un widget de barra lateral.
- HTML/XML/RSS link checker: busca enlaces rotos en archivos HTML, XML, RSS o Markdown.

A. Validación

El proceso de validación del código es un paso muy importante para garantizar que el código es correcto. Funcionará o no, pero lo que es seguro es que sintácticamente es correcto y cumple las normativas del estándar correspondiente.

El procedimiento de validación es muy sencillo. Se puede hacer online, en alguno de los sitios disponibles para ello. Hay que dar la URL donde está ubicado el archivo y la herramienta se encarga de comprobar que la dirección es correcta y que el archivo no contiene errores. También se puede copiar y pegar el archivo en la propia web validadora y ejecutar el chequeo.

Hay validadores que ofrecen un icono para incluirlo en la página principal del sitio web, y el usuario, al pulsar sobre él, accederá al archivo actual y navegará por las opciones dadas.

Validadores conocidos disponibles en Internet:

- W3C Feed Validation Service mediante URI.
- W3C Feed Validation Service mediante código.
- RSS Advisory Board.

5. Directorios de canales de contenidos. Agregadores

Para incluir un canal en un directorio hay que registrar el archivo RSS en él.

Al syndicar el canal es importante tener en cuenta el problema que pueda presentar la codificación, y por ello se recomienda utilizar utf-8 y vigilar los caracteres especiales como acentos, >, <, etcétera.

Algunos directorios de contenidos o feeds son los siguientes.

	El País	Diario de noticias.
	El Mundo	Diario de noticias.
	Ibnews	Agencia de información en español.
	RSS Search Engine Roundtable	Informa sobre los hilos más interesantes que tienen lugar en los foros de SEM (marketing de motores de búsqueda).
	Moz Blog	Sugerencias, trucos y consejos para mejorar los sitios web y hacer mejores búsquedas, redes sociales, contenido y marketing de marca.
	Search Engine Journal	Noticias relacionadas con marketing y SEO.

Tabla 3.10. Feeds en español e inglés.

Además de syndicar un canal, también podemos promocionarlo en los directorios más populares. En este punto haremos una diferenciación entre syndicación pagada y syndicación gratuita.

Pagar por syndicar	Existen plataformas en las que el usuario debe pagar para poder situar sus contenidos en publicaciones importantes, como por ejemplo Outbrain o Taboola .
Syndicación gratuita	La syndicación es gratuita cuando el usuario puede publicar él mismo sus contenidos en otros sitios web.

Tabla 3.11. Tipos de syndicación.

A. Agregadores o lectores de fuentes RSS

Para leer los canales de contenidos se necesita el navegador web o algún programa especializado, que se llama agregador o lector de feeds.

Los agregadores de noticias son, por lo tanto, una ayuda al usuario, ya que le permiten disponer de varias fuentes de información en una misma página y sin necesidad de ir visitando cada uno de los diferentes sitios origen de la información.

Además, el agregador clasifica u organiza la información en base a diferentes categorías, temáticas, noticias locales, internacionales, etcétera, y muestra al usuario un titular de las noticias y una breve descripción de esta.

Los agregadores se sincronizan automáticamente al conectarse a la red, y el usuario recibe la notificación si hay noticias disponibles en aquellos canales a los cuales se ha suscrito.

¿Qué puede funcionar como agregador de contenidos?
• El propio navegador web. • Un programa instalado en el ordenador. • Un programa de correo electrónico. • Una página web específica a la que se puede acceder y ver todos los canales.
¿Qué ventajas ofrece al usuario el agregador de contenidos?
• Disponer de todos los contenidos en una única web o aplicación. • Organizar fuentes de contenidos por temática o tipo. • Descartar todo lo que se ha leído y solo ver lo que queda pendiente. • Las noticias llegan al usuario y no el usuario a ellas, y a medida que se van publicando. • Se pueden guardar favoritos o leer más tarde.
¿Cómo hay que suscribirse?
Es necesario disponer de un lector RSS, y pueden ser: • Lectores RSS que se instalan directamente en el ordenador: FeedReader. • Lectores RSS online (utilizando una página web): Netvibes. • Lectores RSS en el navegador web: Microsoft Edge (Feeder), Mozilla Firefox (Feedbro, Feedly, Easy RSS), Chrome (RSS Subscriptions), Safari (Feedly, Inoreader). • Clientes de correo electrónico: Outlook Express o Mozilla Thunderbird. En general, los navegadores web incorporan los llamados marcadores dinámicos, directamente o mediante el uso de alguna extensión.

Tabla 3.12. Características de los agregadores de contenidos.



EJEMPLO

Se puede utilizar Google para hacer la búsqueda de feeds de nuestro interés. Si se quiere estar informado sobre el tema “cacao”, en la caja de búsquedas de Google hay que escribir ‘cacao feeds’, y devuelve una lista de aquellos sitios web relacionados con el cacao que ofrecen feed.

En la figura se ha elegido el sitio web <https://dcacao.webnode.es/rss/>.

Para realizar este ejemplo hay que instalar en el navegador Firefox la extensión Feedbro.

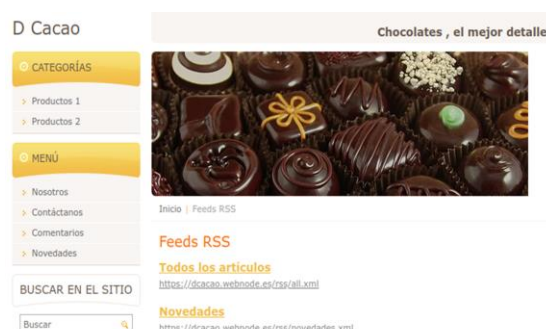


Fig. 3.8. Sindicación con Feedbro.

En la imagen aparecen dos opciones de sindicación, una a todos los artículos y otra solo a las novedades, cada una de ellas con su URL correspondiente.

Si ahora se abre, desde la barra de marcadores, la extensión Feedbro, en el campo URL escribimos la URL correspondiente al feed de Novedades, por ejemplo:

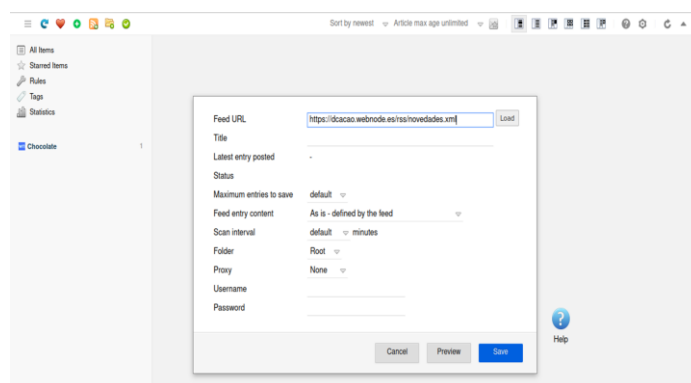


Fig. 3.9. Sindicando con Feedbro.

A partir de ahora, cuando haya novedades en la página sindicada aparecerá una señal o marca en el marcador de Feedbro.

Si el usuario ya disponía de marcadores dinámicos para poder seguir utilizándolos, lo que hay que hacer es exportarlos a un archivo OPML (Outline Processor Markup Language, que es un formato XML para esquemas) para luego importarlo a otro lector de contenidos. Hay que tener en cuenta que Firefox, por ejemplo, a partir de la versión 64, no soporta marcadores dinámicos y no se podrán crear nuevos.

Los clientes de correo, como Mozilla Thunderbird, incorporan también esta funcionalidad.

Otra opción para que los usuarios lean los canales suscritos es utilizar programas específicos instalados en su ordenador, que son los lectores de escritorio. Las noticias se podrán leer posteriormente, aunque no haya conexión, ya que son guardadas en la caché de la aplicación.

El funcionamiento de este software es sencillo. Al suscribir un canal, el programa guarda una copia del archivo .rss o .atom y luego lo compara con los disponibles en el servidor correspondiente y muestra los cambios producidos, como actualizaciones.

Nombramos otros lectores de escritorio:

Akgregator	Lector de noticias RSS y Atom para GNU/Linux (KDE). Los feeds se pueden clasificar en categorías, y el usuario puede solicitar manualmente la obtención de todos los feeds de una categoría concreta.
RSSOwl	Lector de noticias RSS multiplataforma. Con soporte para etiquetas, filtros o distintos modos de visualización y compatible con podcasts, pero requiere Java y eso lo hace menos atractivo.
Newsblur	En función de las preferencias de los usuarios, filtra y clasifica automáticamente los nuevos feeds entre interesantes, normales y poco interesantes. Dispone de versiones de pago y gratuitas.
Feedly	Lector de RSS gratuito (algunas opciones son de pago) muy popular y completo. Dispone de versiones para iOS y Android y una versión web. De esta forma, el usuario puede tener sincronizado su teléfono móvil, su tableta y su ordenador.
Inoreader	Muy similar en funcionalidad a Feedly y gratuito, excepto para algunos servicios.

Tabla 3.13. Lectores de escritorio.

Por último, comentar la opción de obtener los RSS de aquellos sitios web que no ofrecen servicio de feeds. Este es el caso de RSS Bridge, que es un servicio web desarrollado en PHP, disponible en el repositorio de GitHub (<https://github.com/rss-bridge/rss-bridge>) y que también se puede utilizar online (<https://rss.nixnet.services>). Pero no es realmente un lector de feeds, sino que los crea en diferentes formatos (Atom, HTML, JSON, etcétera).

Con RSS Bridges se puede acceder a los RSS de los sitios de Twitter, Wikipedia, YouTube, Pinterest, Instagram, Facebook, DuckDuckGo, entre otros.

B. Otros canales de distribución de contenidos

Sin llegar a la sindicación de contenidos, tal y como se ha explicado en la unidad, existen otros mecanismos de distribución de contenidos digitales muy utilizados en la actualidad, algunos de ellos más dirigidos a la publicidad o el marketing. Comentamos muy brevemente algunos de ellos.

Redes sociales	Cada red funciona de forma diferente, pero en estos momentos son las grandes distribuidoras de contenidos por su alcance y visibilidad. Entre ellas están Facebook, Instagram y Twitter, cada una con sus características propias.
----------------	--

Correos electrónicos	Es el medio más conocido por ser también el más sencillo si se dispone de una base de datos de usuarios. Debe haber una página web detrás, a la que se conducirán los potenciales usuarios consumidores de los contenidos. En concreto, Google pone a disposición del usuario un sistema de alertas para recibir avisos sobre novedades producidas en cualquier ámbito. El Caso práctico 5 muestra cómo hacerlo.
Influencers	Son personas con una gran audiencia de seguidores que confían en sus recomendaciones por el motivo que sea y en determinados ámbitos. Si el usuario consigue que un influencer hable de sus contenidos, prácticamente el éxito está asegurado.
Aplicaciones móviles	Existen apps que permiten al usuario relacionarse con sus clientes y hacerles llegar contenidos mediante las denominadas “notificaciones push”, que en realidad son estrategias de marketing.
Publicidad promocional	Se refiere a contenido patrocinado, que puede ser comercial o referencias de lectura.

Tabla 3.14. Mecanismos de distribución de contenidos.



Unidad 4.

Manipulación de documentos web



1. Lenguajes de script del lado cliente

En los lenguajes de script del lado cliente, cuando el usuario escribe la URL que quiere visitar en el navegador web, este actúa como cliente frente al servidor web donde está alojado el sitio web solicitado.

Toda la información relativa al sitio (código HTML, CSS, imágenes, videos, etcétera) está alojada en el servidor web y es transferida al navegador del usuario. El navegador muestra la/s página/s del sitio web y también puede ejecutar código en el propio navegador del usuario.

En los lenguajes de programación del lado servidor, antes de enviar la/s página/s al navegador se suelen ejecutar acciones como accesos a BBDD, conexiones de red y otras acciones requeridas para poder formar la página en el cliente web. Se suelen utilizar en aplicaciones de tipo dinámico, en las que existe mucha interactividad con el usuario.

Los lenguajes del lado cliente no se pueden considerar como un sustituto de los lenguajes del lado servidor, sino un complemento. Y una buena opción sería trabajar con lenguajes que combinen de alguna manera ambas tecnologías.

Un ejemplo de este tipo es AJAX (Asynchronous JavaScript and XML, JavaScript asíncrono y XML), que siendo del lado cliente combina varias tecnologías como son HTML, CSS, JavaScript, DOM, XML, XSLT y utiliza el objeto XMLHttpRequest para comunicarse con los servidores en segundo plano (background).

Con AJAX se realiza una transferencia asíncrona de los datos entre el cliente y el servidor, y esto le permite al servidor poder atender peticiones del cliente prácticamente en tiempo real y sin necesidad de recargar la página.

Entre los lenguajes del lado cliente, los más utilizados son JavaScript, VBScript (solo funciona con Internet Explorer o Chrome) y TypeScript. Hay lenguajes de script de consola, como son el bash o PowerShell.

Estamos hablando de lenguajes de script y lenguajes de programación. Veamos cuál es la diferencia entre ambos.

Cuando se utiliza un lenguaje de programación, el desarrollador crea un archivo cuyo contenido es el programa fuente. Este archivo fuente tiene que sufrir un proceso de transformación o traducción a lenguaje máquina (binario, ceros y unos), y para ello utiliza otro software específico llamado compilador que genera el archivo ejecutable. El proceso es algo más complejo, pero solo se quiere incidir en lo básico.

En el caso del lenguaje de script no existe proceso de compilación, no hay archivo binario, no hay ejecutable y el archivo fuente es directamente ejecutado (interpretado) en la máquina cliente. Este tipo de lenguajes facilitan la tarea al desarrollador pero cargan al procesador, y se suelen utilizar para realizar pequeñas acciones, programas pequeños.



IMPORTANTE

Se recomienda disponer al menos de dos navegadores web para ejecutar los ejemplos y ejercicios, preferiblemente que sean software libre. Por ejemplo, Mozilla Firefox o Microsoft Edge.

1.1. Javascript. Características generales

JavaScript es el lenguaje de script del lado cliente más conocido y utilizado. Fue creado por Brendan Eich (de la empresa Netscape Communications Corporation), y en un principio se llamó LiveScript.

En 1995, y después de un acuerdo (enfocado solo al marketing) entre Netscape y Sun Microsystems, pasó a llamarse JavaScript.

A partir de 1997, JavaScript sigue el estándar ECMA-262. Pero fue a partir de 2004 cuando el JavaScript despegó realmente convirtiéndose en un estándar web. La versión actual del estándar es ECMAScript 2016, y fue publicada en junio de 2016.

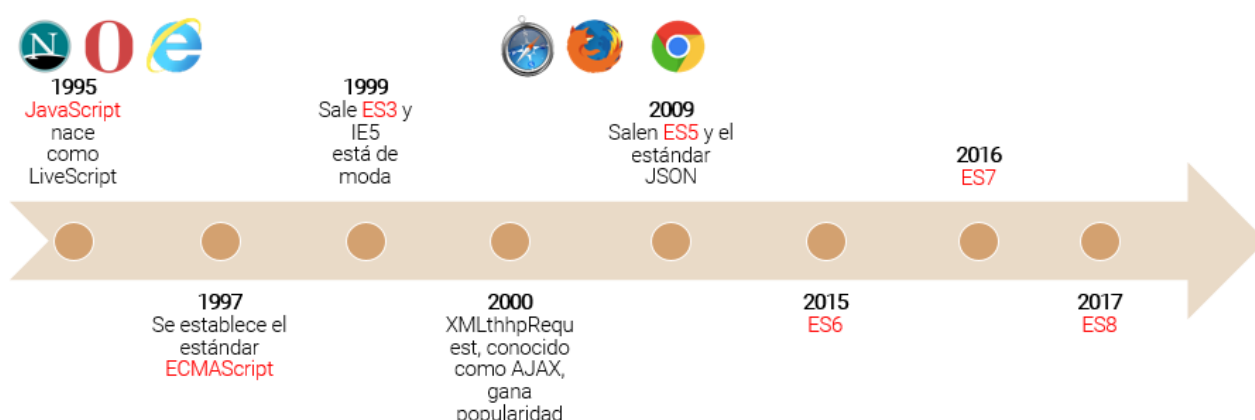


Fig. 1.1. Evolución del lenguaje JavaScript.

No hay que confundir JavaScript con el lenguaje Java (Sun Microsystems). Además de algunas sintaxis similares, solo tienen en común parte del nombre.

Con JavaScript se consigue que la página web sea más dinámica e interactiva, se le da más funcionalidad y por lo tanto resulta más atractiva para el usuario. Por ejemplo, se puede añadir movimiento a los objetos dentro de la página, crear algunos efectos especiales, etcétera.

Pero ¿qué es JavaScript?

Javascript es un lenguaje de script del lado cliente utilizado en las páginas web.

Las principales características del lenguaje JavaScript son:

- Es un lenguaje interpretado.
- No requiere especificar el tipo de dato de las variables. Esta propiedad se llama «tipado débil».
- Es un lenguaje orientado a objetos, es decir, utiliza clases y objetos.
- Por sí mismo no puede escribir archivos en el servidor. Puede preparar los datos para el envío, pero es el lenguaje de programación del lado servidor el que los procesa.
- Hay que asegurarse de que el navegador web a utilizar soporta JavaScript y sabe interpretarlo, y si no lo hace la página seguirá ejecutándose pero sin la funcionalidad añadida de JavaScript.

En este [enlace](#) tienes disponible más información sobre la programación orientada a objetos en JavaScript.

¿Para qué se utiliza JavaScript?

- Para añadir interactividad en una página web mediante la creación de menús desplegables, transformar texto en una página, añadir elementos dinámicos a una página o ayudar con los elementos de formulario (entrada de datos).
- Para preprocesar (validar) datos en el cliente antes de enviarlos al servidor.
- Para, en base a interacciones del usuario, modificar de forma dinámica estilos y contenidos.
- Solicitar archivos al servidor.
- Enviar solicitudes de lectura y/o escritura a programas del lado servidor.

En general, y aunque todos los navegadores web soportan JavaScript (Firefox, Chrome, Safari, Microsoft Edge...), es importante comprobar que los scripts creados van a ser interpretados de la misma forma en los navegadores más habituales y los resultados sean los mismos.

Pero no todo son ventajas con el uso de JavaScript. Tiene sus limitaciones y hay cosas que este lenguaje no puede hacer:

1. No todos los navegadores lo soportan, y además puede haber incompatibilidades entre ellos dependiendo de la versión de JavaScript.
2. Al ser del lado cliente, el propio usuario puede desactivar o deshabilitar el código JavaScript y, en consecuencia, la ejecución no está garantizada por defecto.
3. Respecto al tema de accesibilidad, algunos navegadores de voz no interpretan el código JavaScript.
4. Respecto a dispositivos móviles, en algunos no se ejecuta JavaScript.

1.2. Estructura de un programa en JavaScript

Un programa escrito en JavaScript consta de:

- **Variables.** Hay que declarar las variables a utilizar en el script, aunque no sea obligatorio. Es preferible adquirir el hábito de hacerlo.
- **Funciones.** Constan de una porción de código con una finalidad concreta (un cálculo, una comprobación, una acción...) y que puede devolver o no un resultado.
- **Instrucciones.** Son los comandos u órdenes a ejecutar.
- **Resultado.** Puede ser una acción concreta, un texto, un valor numérico, etcétera.
- **Comentarios.** Se deben incluir comentarios de cabecera explicando qué hace el script, comentarios de desarrollo si fuera conveniente y de resultado.

Sintaxis básica

Comenzamos con el típico programa que muestra el texto “¡Hola Mundo!”. A partir de ahí iremos introduciendo la sintaxis del lenguaje.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8"/>
    <title>Mi primera prueba con JS</title>
  </head>
  <body>
    <section>
      <span onclick="alert('¡Hola Mundo!');">Haz clic aquí</span>
    </section>
  </body>
</html>
```

El navegador Firefox muestra el texto “Haz clic aquí”, y al pulsar sobre él se ve, la siguiente imagen:

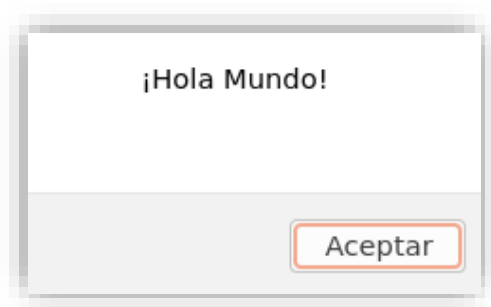


Fig. 1.2. Mensaje con alert().

En este ejemplo, el código JavaScript va entre las etiquetas `<span ></span>` y utiliza el atributo `onclick` que, al hacer clic sobre el texto, ejecuta la función `alert()`, que muestra un cuadro de diálogo con el texto dado. Dentro de la etiqueta `<span>` se puede añadir el atributo `type` de la forma `type="text/javascript"`, pero no hace falta con HTML5.

Los comandos JavaScript terminan con “;”, aunque hay situaciones en las que se puede eliminar, como por ejemplo cuando hay un salto de línea.

El código JavaScript será una secuencia de instrucciones o sentencias de la forma:

```
sentencia1;
sentencia2;
```

...

También se puede escribir una sentencia a continuación de otra: `sentencia1; sentencia2;`

El sangrado y espaciado es también opcional, y si se incluye es para mejorar la legibilidad del código.

Existen dos tipos de comentarios:

- Comentar una línea: `//comenta todo lo que haya en esa línea`
- Comentar varias líneas: `/*comenta línea1`

`comenta línea2 */`

Las funciones tienen un nombre, y los argumentos van entre paréntesis. Terminan en “;”, ya que son una sentencia. Se incidirá sobre las funciones más adelante.

1.3. Integración de JavaScript en HTML5

Al igual que CSS, existen tres técnicas para insertar código JavaScript en archivos HTML. Pero la recomendada por HTML5 siempre es la tercera.



Fig. 1.3. Integración HTML5+CSS3+JavaScript.

A. Integración en línea

El código JavaScript se integra directamente en el código HTML. Esta forma de trabajar permite utilizar atributos de los elementos HTML que serán manejadores de eventos, es decir, ejecutan código en función de la acción del usuario. Son típicos los eventos de ratón `onclick`, `onMouseOver`, etcétera. Más adelante se explican con detalle los eventos.

En el ejemplo anterior de “¡Hola Mundo!” se ha utilizado esta técnica, que no se recomienda utilizar porque presenta, por ejemplo, el problema de que insertado de esta forma no se puede reutilizar el código JavaScript. Para mostrar varias veces el mensaje habrá que escribir varias veces el código.

Como ejemplo sirve, pero no es buena técnica cuando hablamos de una página completa.

B. Código embebido

Si el código tiene ya cierta envergadura es preferible unir todas las acciones JavaScript en una zona del archivo HTML, y sus acciones se ejecutarán utilizando referencias.

Para ello se utilizan las etiquetas `<script></script>`, y HTML5 directamente “sabe” que el código incluido es JavaScript.

Se pueden incluir las etiquetas `<script></script>` entre `<head></head>` o entre `<body></body>`. Pero si se incluye el código JavaScript dentro de `head` o al principio de `body` pueden aparecer errores (no visibles) que hagan que los programas no funcionen correctamente.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <title>Mi primera prueba con JS</title>
    <script>
      function alerta() {
        alert('Hola Mundo!');
      }
    </script>
  </head>
  <body>
    <section>
      <span onclick="alerta();">Haz clic aquí</span>
    </section>
  </body>
</html>
```

El resultado es el mismo que en el caso anterior, es decir, el navegador mostrará lo mismo. Pero hay mejoras respecto a él. Se ha creado una función `alert()` que se puede invocar tantas veces como se necesite. Además, todo el código JavaScript está en la cabecera, lo cual facilita su mantenimiento. Pero es posible mejorar todavía más el código haciendo que no recaiga la ejecución sobre el atributo de HTML `onclick`. Se trata de ejecutar JavaScript al 100%, y para ello se utiliza un identificador en su lugar.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <title>Mi primera prueba con JS</title>
    <script>
      window.onload = function() {
        document.getElementById('alerta').onclick = function () {
          alert(';Hola Mundo!');
        }
      }
    </script>
  </head>
  <section>
    <span id="alerta">Haz clic aquí</span>
  </section>
</body>
</html>
```

En este caso se está utilizando el evento `onload` del objeto `window()`, y con él se está indicando que cuando el navegador termine la carga de la página (`window.onload`) ejecute una función JavaScript que incluye la llamada al evento `onclick` desde el elemento `span`. Los objetos `window()` y `document()` se estudian en esta unidad, más adelante.

C. Código en un archivo externo

En este caso se agrupa todo el código JavaScript en un archivo con extensión `.js` que se enlaza desde el código HTML. Al utilizar este método, además de ser la opción recomendada por HTML5, se le da a la página web más modularidad, se puede reutilizar el código JavaScript desde otras páginas web, se facilita la depuración de errores y le da mayor velocidad de carga, entre otras ventajas.

La referencia en el código HTML se hace de la forma:

```
<script src="./codigo.js"></script>
```

Si la página requiere de otro archivo JavaScript habría que incluir otra línea similar para referenciar el siguiente archivo `.js`. Si los archivos `.js` no están en el mismo directorio que el archivo `.html` habría que incluir la ruta (path, en inglés) o la URL donde está alojado.

```
<script src="http://localhost/UD4/codigo.js"></script>
```

Como archivo externo no hay que escribir nada entre las etiquetas `<script></script>`.

El ejemplo anterior quedaría así:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <title>Mi primera prueba con JS</title>
    <script src="./codigo.js"></script>
  </head>
  <body>
    <section>
      <span id="alerta">Haz clic aquí</span>
    </section>
  </body>
</html>
```

El archivo codigo.js contendría:

```
window.onload = function() {
  document.getElementById('alerta').onclick = function () {
    alert('Hola Mundo!'); }
}
```

1.4. Otros ambientes de ejecución del código JavaScript

En el subapartado C se han explicado las diferentes formas de integrar código Javascript en HTML. Vamos ahora a completarlo añadiendo otros contextos de ejecución del código JavaScript relacionados con tecnologías web.

A. Evento dentro del documento HTML

Este caso estaría incluido dentro de la categoría de código embebido y consiste en incluir código JavaScript en algunos eventos como son `onload`, `onmouseover` y `onclick`, y que se ejecutará cuando el evento se active.

Es preferible siempre que el código JavaScript a ejecutar no esté incluido en HTML, sino en un archivo .js.

```
<span onclick="alerta();">Haz clic aquí</span>
```

B. La consola web del navegador

Todos los navegadores disponen de un conjunto de herramientas específicas para el desarrollador a las que se accede mediante el atajo de teclado CTRL+MAY+I (en algunos casos con F12) o, desde el menú de configuración, la opción *Desarrollador web -> Herramientas*. Entre ellas está la consola web, que es un intérprete interactivo de JavaScript. Desde la consola se pueden hacer pruebas de ejecución de código en tiempo real y comprobar su funcionamiento.

La consola funciona de forma similar a una terminal del sistema operativo, pero con funcionalidad añadida. Por ejemplo, autocompleta instrucciones o guarda un historial de comandos ejecutados, pero sobre todo permite el seguimiento del programa.

En realidad es el objeto `console` el que controla esta funcionalidad de JavaScript y, en concreto, los métodos `console.log()` y `console.dir()` son los que reciben una lista de objetos separados por comas y los imprimen de diferente forma en función de su naturaleza. En concreto, `console.dir()` muestra una lista interactiva de las propiedades del objeto JavaScript especificado. Esta segunda opción no es estándar, por lo que es preferible no utilizarla.

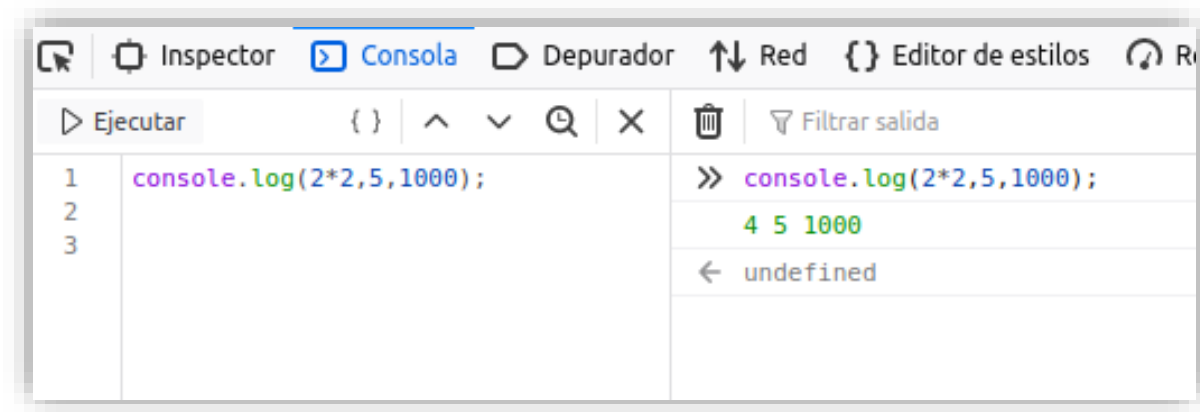


Fig. 1.4. Consola del navegador.



EJEMPLO 1

Archivo [cuadrados.js](#) disponible en [Ejemplos para trabajar y analizar](#).

Mostrar en la consola los cuadrados de los 10 primeros números naturales junto con el texto "Cuadrados de los 10 primeros números naturales".

```
console.info("Cuadrados de los 10 primeros números naturales")
for (var n = 1; n <= 10; ++n) console.log('%d^2 = %d', n, n * n)
```

El método `console.info()` permite mostrar por consola textos. Se verán más métodos cuando se estudie el objeto `console`.

La consola web del navegador en general sirve para probar instrucciones y luego trasladarlas a un archivo externo `.js`.

C. La consola web de Node.js (REPL Read-Eval-Print-Loop, intérprete interactivo)

[Node.js](#) proporciona un entorno de ejecución de código JavaScript en el propio sistema operativo, no en el navegador web. De esta forma, los programas pueden acceder a recursos del sistema que no están disponibles a través del navegador, como son bases de datos, el sistema de archivos, etcétera.



IMPORTANTE

Para instalar este software, descargar la versión para GNU/Linux 64 desde la página oficial del proyecto <https://nodejs.org/es/> o instalar directamente con el administrador de paquetes snap. Para sistemas Windows, acceder a <https://nodejs.org/es/download/> y descargar el instalador de Windows.

Una vez instalada, utiliza desde el prompt el comando `node` , que hace de intérprete de JavaScript de forma similar a la consola web del navegador. A este comando también se le conoce como REPL por la función que desempeña, ya que ejecuta un ciclo de lectura de instrucciones, las evalúa e imprime los resultados (EPL, Read-Eval-Print Loop).

Se recomienda que el script de JavaScript esté en un archivo externo `.js`, ya que, en este caso puede ser utilizado tanto desde [node.js](https://nodejs.org/es/) como desde una página HTML. Este archivo `.js` se le pasa (path absoluto hasta el archivo, si es necesario) como argumento al comando `node`.

D. Análisis estático de código (lint)

Los programas `linter` sirven, en general, para hacer un análisis estático del código y así detectar errores, código no eficiente, malas prácticas, etcétera. Va más allá de la validación.

Para JavaScript en concreto existe la herramienta [ESLint](https://eslint.org/) (ECMAScript Linter) para realizar esta tarea. Debe existir un proyecto y un archivo de configuración con las reglas que se tienen que aplicar.

El comando `eslint --init` puede ayudar en la configuración, por ejemplo para determinar cuál será el formato: JavaScript, JSON o YAML (formatos de texto para el intercambio de datos).

Otras herramientas de este tipo son [JSLint](https://jshint.com/) y [JSHint](https://jshint.com/).

1.5. Control de ejecución de scripts

JavaScript dispone de los atributos `defer` y `async` , que permiten controlar en qué momento se debe ejecutar el script.

defer	async	Comportamiento	Ejemplo
—	—	Si no se incluye ninguno de los dos, el script se ejecuta de forma inmediata. Como el código JavaScript está en un recurso externo, el navegador debe esperar a que este se haya descargado completamente antes de continuar desplegando el documento.	<code><script src="archivo .js"/></code>
defer	—	El script se ejecuta después de que el documento esté cargado por completo. Es la forma de ejecución más utilizada.	<code><script src="archivo .js" defer="defer"/></code>
—	async	De forma asíncrona, el navegador continúa cargando el documento y solicita al servidor web el	<code><script src="archivo .js"</code>

defer	async	Comportamiento	Ejemplo
		archivo .js. Cuando lo recibe lo ejecuta al tiempo que se sigue cargando el documento. Solo está disponible en HTML5.	<code>async="async"/></code>
defer	async	El navegador actúa como si solo se hubiese indicado async.	—

Tabla 1. Atributos defer y async.



IMPORTANTE

La directiva `use strict` no es código propiamente, y su inclusión indica el modo en que el navegador ha de ejecutar el código JavaScript. Supone la desactivación de la compatibilidad hacia atrás, y con ello la posibilidad de utilizar “prácticas indebidas”.

Si no se incluye JavaScript trabaja en modo normal. Por ejemplo, cuando se explique el trabajo con variables se dirá que no es necesaria su declaración (normal mode); sin embargo, trabajando en strict mode es obligatorio hacerlo.

Esta directiva se escribe al comienzo de un archivo y afecta a todo el código y a todas las funciones incluidas en él. También se puede escribir al comienzo de una función, y en ese caso afecta solo al código en el ámbito de dicha función. La sintaxis es:

```
'use strict';
```

2. Elementos del lenguaje JavaScript

Veamos cada uno de estos elementos.

2.1. Datos

JavaScript utiliza cuatro tipos de datos: numéricos, de cadena, booleanos y punteros.

- Numéricos: números enteros, decimales, con notación científica (5.0e6), con notación hexadecimal (0xFF) o notación octal (016).
- Cadena: conjunto de datos alfanuméricos encerrados entre "" ("50" es una cadena, 50 es un número).
- Booleanos: solo tienen dos posibles valores, verdadero (true) o falso (false).
- Punteros: direcciones de memoria utilizadas para asignar funciones.

En el siguiente [enlace](#) hay disponible más información complementaria sobre los datos en JavaScript.

2.2. Variables y operadores

JavaScript utiliza variables para almacenar datos.

Para la declaración de variables (es una reserva de memoria) en JavaScript se utilizan las palabras reservadas `const`, `let` y `var`.

- `const`: la variable tiene un valor constante, es decir, no cambia su valor.
- `let` / `var`: el valor de la variable puede cambiar en tiempo de ejecución.

Ejemplo:

```
var variable; //dentro de una función
let variable; //dentro del bloque
```

La diferencia entre `let` y `var` radica en su ámbito de aplicación. Con `let`, la variable solo se puede utilizar dentro del bloque donde se ha declarado. Con `var` se puede utilizar, además, desde fuera del bloque, con lo cual puede sufrir modificaciones no deseadas.

La única ventaja que tiene utilizar `var` es que se pueden definir dos variables con el mismo nombre, y que con `let` no se puede. El estándar aconseja no utilizar `var` a menos que sea dentro de una función.

Hay que tener en cuenta que se diferencia entre mayúsculas y minúsculas en el nombre de las variables (case sensitive).

Una vez declarada la variable, se le puede asignar un valor:

```
nombrevariable = 10.7;
```

Pero se puede hacer al tiempo: declarar y asignar valor, y para varias variables.

```
var variable1, variable2 = 9, variable3;
```

Se han declarado tres variables y solo a una de ellas, `variable2`, se le ha asignado un valor. El tipo de dato que contiene o va a contener la variable no afecta a la declaración.

Se puede conocer el tipo de variable utilizando la instrucción `typeof` de la forma:

```
var Falso = false;
alert (typeof Falso) // Devuelve: boolean
```

Si devuelve `undefined` es que la variable no existe o, si está declarada, no contiene nada.

Ejemplo de sintaxis: probar en la consola web.

```
var numero = 25;
numero += 10;
alert (numero) // Devuelve: 35
```

Veamos algún ejemplo con `let`: probar en la consola.

```
for(let i=0;i< 2; ++i) {
    document.writeln('Primero ', i);
    for(let i=0;i<2; ++i) {
        document.writeln('Segundo ', i);}
}
```

En esta porción de código hay dos bloques con `for` (se estudia en el siguiente subapartado), y por lo tanto se han podido definir dos variables diferentes con el mismo nombre `i`. La salida es:

```
Primero 0 Segundo 0 Segundo 1 Primero 1 Segundo 0 Segundo 1
```

Si se cambia `let` por `var` se comprueba que la salida generada es diferente.

```
Primero 0 Segundo 0 Segundo 1
```

Es más fiable utilizar `let`.

Operadores en JavaScript

La tabla siguiente detalla los tipos de operadores disponibles en JavaScript:

Tipo	Función
Comparación	Compara los valores de 2 operandos y el resultado puede ser true o false. <code>== != === !== > >= < <=</code>
Aritméticos	Resuelve la operación aritmética entre los operandos. El operador + suma números y concatena cadenas. <code>+ - * / % ++ -- +valor -valor</code>
Asignación	Asigna un valor a la variable. <code>= += -= *= /= %= <<= >= >>= >>>= &= = ^= []</code>
Concatenar	Unir cadenas. Operador + <code>var cadena3 = "Hola" + cadena2;</code>
Boolean	Realiza una operación de tipo lógico entre uno/dos operandos booleanos. <code>&& !</code>
Bit a Bit	Sobre datos binarios realiza una operación aritmética o de desplazamiento. <code>& ^ ~ << >> >>></code>

Tabla 1. Operadores en JavaScript.



IMPORTANTE

No se pueden utilizar palabras reservadas como nombres de variables.

En el siguiente [enlace](#) hay disponible más información sobre los operadores de JavaScript.

En el siguiente [enlace](#) hay disponible más información sobre las palabras reservadas de JavaScript.

2.3. Arrays

Un array es un tipo de variable que se compone de un conjunto de elementos a los que se accede a través de un índice que comienza en 0.

Para acceder a un determinado elemento, ya sea para asignar un valor o para obtenerlo, hay que indicar su posición o índice.

Ejemplo de sintaxis:

```
var semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

Para acceder a un elemento en concreto, por ejemplo el Martes, y mostrarlo al usuario se utiliza la función `alert()` de la forma:

```
alert( semana[1] );
```

Si un array se declara vacío se le puede asignar valor de la forma:

```
var vacio = [];  
.....  
vacio[0] = 123;  
.....  
alert( vacio[0] );    // muestra 123
```

2.3. Sentencias de control

Las sentencias de control permiten controlar el flujo de ejecución del programa.

Se clasifican en:

1. Condicionales o selectivas.
2. Repetitivas o iterativas.
3. De control de errores.

A. Condicionales

El control del flujo de ejecución se realiza en base a la evaluación de una condición que podrá ser `true` (verdadera) o `false` (falsa).

Los tipos de sentencias de control son: `if`, `if...else`, `switch` y los bucles.

Sentencia `if`: se evalúa la condición y si es `true` se ejecuta el código incluido.

Ejemplo de sintaxis:

```
if (1 < 2 && 8 >= 4) {
    alert("Condicion verdadera, se ve el mensaje");}
if (1 > 2 || 8 <= 4) {
    alert("Condicion falsa, NO se ve el mensaje");}
```

Sentencia `if...else`

En esta sentencia se evalúa tanto si la respuesta es verdadera (`true`) como si es falsa (`false`).

Ejemplo de sintaxis:

```
var x = 80;
if( x >= 100 ) {
    alert("Mayor o igual de 100");}
else {alert("Menor de 100");}
```

Sentencia `switch`

Cuando la elección es entre varias opciones y siempre que sea posible, es preferible utilizar `switch()`.



EJEMPLO 1

Archivo ejemplo_1.html en [Ejemplos para trabajar y analizar](#)

Dado un día de la semana, indicar si es entre semana o fin de semana.

```
var dia = 3;
switch( true ) {
    case ( dia >= 1 && dia <= 5 ):
        alert("Entre semana");
        break;
    case ( dia == 6 || dia == 7 ):
        alert("Fin de semana");
        break;
    default: alert("Valor no válido"); }
```

B. Repetitivas o iterativas. Bucles

Los bucles son sentencias repetitivas, que se ejecutan en función de una condición o un número determinado de veces.

Bucle for: se indica el valor inicial de la variable contador, la condición que debe cumplir y el incremento del valor de la variable contador que controla el bucle.

```
var contador;
for(contador=1; contador <= 5; contador++) {
    alert( "Ahora contador tiene valor [" + contador + "]" );}
```

Bucle while: mientras la variable que controla el bucle cumpla la condición, se ejecutan la/s sentencias incluidas y antes de pasar a la siguiente iteración se incrementa la variable de control.

```
var contador = 1;
while( contador <= 5 ) {
    alert( "Ahora contador tiene valor [" + contador + "]" ); contador
    = contador + 1;}
```

Bucle do...while: se ejecuta/n la/s sentencias y se incrementa la variable que controla el bucle mientras se cumpla la condición.

```
var contador = 1;
do{alert( "Ahora contador tiene valor [" + contador + "]" );
    contador = contador + 1;}
while( contador <= 5 )
```

Los ciclos del bucle se controlan mediante una variable que hace la función de contador, pero se puede manipular el funcionamiento normal utilizando las palabras reservadas `break` y `continue`.

- `break` detiene el bucle de forma brusca.
- `continue` permite saltarse algunas iteraciones del bucle.



EJEMPLO 2

Archivo ejemplo_2.html

Utilización básica del bucle `for`.

En el siguiente [enlace](#) tienes disponibles más ejemplos.



EJEMPLO 3

Archivo ejemplo_3.html

Bucle que calcula y muestra los cuadrados de los veinte primeros números naturales.

```
<body>
  <h1>Cuadrados de los 20 primeros números naturales</h1>
  <script>
    document.write('<ul>');
    for ( let n = 1; n <= 20; ++n )
      document.write('; ', n, '<sup>2</sup> = ', (n * n), '<br>');
    document.write('</ul>');
  </script>
</body>
```

En este ejemplo se está utilizando una lista no ordenada desde JavaScript `<ul></ul>` junto con un cambio de línea. También se utilizan las etiquetas `<sup></sup>` para escribir un superíndice.

C. Control de errores

La sentencia `try ... catch` permite detectar errores y procesarlos. Con `try` se comprueba si hay algún error, y si lo hay se procesa con `catch()`.

La sintaxis es:

```
try {
  // código...}
catch (err) {
  // manipulación de error}
```



EJEMPLO 4

Archivo ejemplo_4.html

En el ejemplo 4 hay un error al procesar la suma, ya que la variable 'y' no está definida y se muestra el mensaje de catch. Para mostrar un mensaje de error se suele utilizar la propiedad `message`, que es reconocida en todos los navegadores web.

Si se incluye `finally`, su contenido se ejecuta siempre, haya error o no.

```
var x = 33;
try
{ x = x + y;}
catch( e ) {
    alert( "error: " + e.message );}
finally {
    alert("En finally");}
```



IMPORTANTE

`try...catch` solo funciona para errores producidos en tiempo de ejecución. Por lo tanto, el código deberá estar previamente validado.

En este [enlace](#) hay disponible un validador de código JavaScript.

2.3. Funciones

Una función es una porción de código que permanece inactivo hasta que es llamado por un evento determinado.

Las funciones en JavaScript se definen con la palabra reservada `function`.

Hay que dar un nombre a la función (aunque no es obligatorio, es conveniente), indicar los parámetros con los que va a trabajar (entre paréntesis) y el código que ejecutará entre llaves.

Con `return`, la función devuelve el resultado.

```
function sumar( A, B )
{ return A + B; }
```

Ejemplo de sintaxis de función:

```
<script>
function popup()
{ alert("Hola mundo!!!") }
</script>
<button type="button" onclick="popup()">Mi primera función</button>
```

JavaScript trabaja con funciones definidas por el usuario y funciones predefinidas. Algunas de las funciones predefinidas son:

- **alert()**: se utiliza para mostrar un mensaje.
- **confirm()**: muestra un cuadro de diálogo para preguntar al usuario sobre alguna cuestión cuyo resultado sea `true` o `false` y que se guarda en la variable dada. Este cuadro de diálogo muestra los botones Cancelar y Aceptar.

```
var ok = confirm("&Enviar los datos?");
if( ok == true )
{
    ...
}
else {
    ...
}
```

- **prompt()**: se utiliza para pedir datos al usuario. Hay que indicar dos parámetros: el primero, el texto para el cuadro de diálogo, y el segundo, un valor por defecto (opcional).

```
var valor = prompt("Introduce la contraseña", "");
if(valor == "ABC123") {
    alert("Contraseña correcta");
} else {
    alert("Contraseña no correcta: [" + valor + "]);
}
```

- **onLoad()** / **onUnload()**: realmente no es una función, sino un evento. `onLoad()` se utiliza cuando se quiere ejecutar una función de JavaScript al cargarse la página web. Por ejemplo, pedir una contraseña. En el caso de `onUnload()`, se utiliza cuando se cierra la página o se

cambia de página. También se puede utilizar al recargar una página o cuando se sale del navegador.



EJEMPLO 5

Archivo ejemplo_5.html

Se pide una contraseña por teclado y se valida en base al valor correcto preestablecido.

Las funciones se pueden definir como anónimas, y en este caso no es obligatorio asignar un nombre. La ventaja de asignar nombre es que la función puede llamarse a sí misma, o se pasa una función como argumento a otra función o si desde una función se llama a otra función.

Ejemplo de función que devuelve el cuadrado de un número y no tiene nombre:

```
const cuadrado = function(numero) { return numero * numero }
var x = cuadrado(3) // x obtiene el valor 9
```



EJEMPLO 6

Archivo ejemplo_6.html

Escribir una función que muestre el factorial de 10.

En este ejemplo se ha utilizado el operador condicional ?: Su significado es el siguiente:

condicion ? expresión si se cumple la condición: expresión si no se cumple;



EJEMPLO 7

Archivo ejemplo_7.html

Escribir una función que compruebe si las frases dadas contienen mayúsculas y/o minúsculas.

En este ejemplo se han introducido dos nuevas funciones muy intuitivas, que son `toUpperCase()` y `toLowerCase()` y que indican la existencia de letras mayúsculas y minúsculas, respectivamente, en una cadena.

2.4. Tratamiento de fecha y hora

JavaScript no tiene un tipo de dato para fechas. El objeto `Date` y sus métodos asociados (obtener

fecha, establecer fecha, manipular fecha, entre otros) son los que permiten trabajar con fechas y horas en el código JavaScript.

Pasamos a ver algunas de las funciones de que dispone JavaScript para el tratamiento de la fecha y hora.

Método	Descripción
<code>getDate()</code>	Devuelve el día del mes (entre el 1 y el 31).
<code>getDay()</code>	Devuelve el día de la semana (entre el 0 y el 6). Domingo es el 0.
<code>getMonth()</code>	Devuelve el mes (entre el 0 y el 11).
<code>getFullYear()</code>	Devuelve el año (en formato de 4 dígitos).
<code>getHours()</code>	Devuelve la hora (entre el 0 y el 23).
<code>getMinutes()</code>	Devuelve los minutos (desde 0 a 59).
<code>getSeconds()</code>	Devuelve los segundos (desde 0 a 59).
<code>getTime()</code>	Devuelve el número de milisegundos desde el 1 de enero de 1970.
<code>getTimezoneOffset()</code>	Devuelve la diferencia de horario en minutos entre la hora local y GMT (meridiano de Greenwich).
<code>getUTCHours()</code>	Devuelve la hora UTC (tiempo universal coordinado).

Tabla 1. Métodos para el tratamiento de fechas.



EJEMPLO 8

Archivo ejemplo_8.html

Escribir el día de la semana, mes y año en curso, así como la hora y los minutos.

Utilizar el método `document.write()` para mostrar el resultado en pantalla. Más adelante se estudia el objeto `document`.

El primer paso es siempre ejecutar la función `new Date()`, que es la que devuelve la hora y la fecha actual, y la zona horaria.

Al ejecutar el código se observa que los métodos que devuelven el día de la semana y el mes lo hacen de forma numérica.

3. Utilización de objetos de JavaScript

Un objeto es una entidad cuyo estado viene definido por una serie de propiedades, y sobre estas propiedades actúan unos métodos o funciones.

Sintácticamente, el acceso a una propiedad de un objeto es: `nombreObjeto.propiedad`

Sintácticamente, el acceso a un método de un objeto es:

```
nombreObjeto.metodo([parámetros opcionales])
```

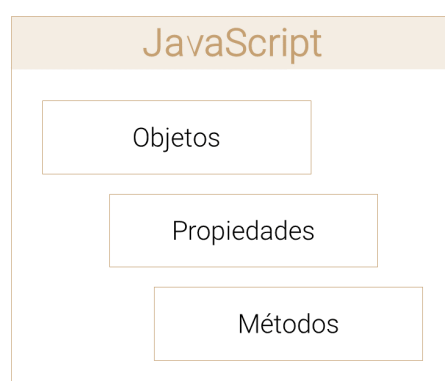


Figura 3.1. JavaScript orientado a objetos.

La figura siguiente muestra la jerarquía de objetos de JavaScript.

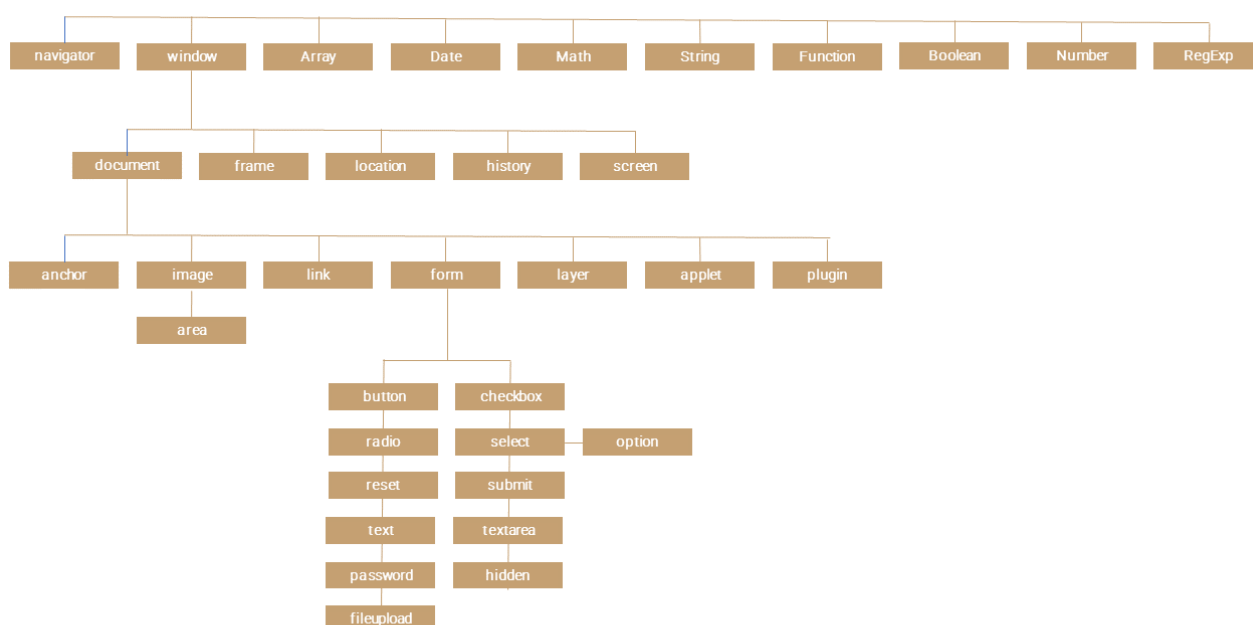


Figura 2. Jerarquía de objetos de JavaScript. .

A continuación se van a explicar los objetos más importantes, sus propiedades y métodos asociados.

3.1. Acceso a la página web. DOM (Document Objet Model)

En unidades y apartados anteriores se ha nombrado el DOM pero no se ha explicado su significado y uso. Ahora es el momento de abordarlo.

Las especificaciones del DOM están controladas por el [W3C](http://www.w3.org/), y los navegadores web las aplican en su software para dar soporte al DOM.

Pero ¿qué es el DOM?

El DOM es un estándar del W3C (World Wide Web Consortium) que permite el acceso y la modificación de los diferentes elementos del documento HTML.

Cuando el navegador carga una página web, el propio navegador crea, de forma automática, una estructura jerárquica con todos los objetos que componen dicho documento. Esta estructura se conoce con el nombre de DOM, y cada uno de los objetos se llama nodo. A partir de esta estructura DOM desde JavaScript se puede acceder a los nodos.

¿Qué se puede hacer a través del DOM? En realidad, el DOM es una API (Application Programming Interfaces, interfaz de programación de aplicaciones) que permite:

- Modificar los atributos y estilos de todos los elementos del documento HTML.
- Añadir o eliminar elementos y atributos existentes.
- Responder a los eventos ocurridos en la página.
- Crear nuevos eventos.

Mediante el DOM, el desarrollador puede, utilizando lenguajes de programación como JavaScript, acceder y modificar el contenido, la estructura lógica y el estilo en documentos HTML de forma dinámica.

Cada objeto del documento web es un nodo que constará de:

Nodo = elemento (etiqueta) + atributos + eventos + texto

La figura siguiente muestra la estructura jerarquía (en árbol) del DOM, y en ella vemos, en el nodo principal, el objeto `window()`, y de él parten los objetos `navigator()`, `screen()`, `history()`, `location()` y `document()`.

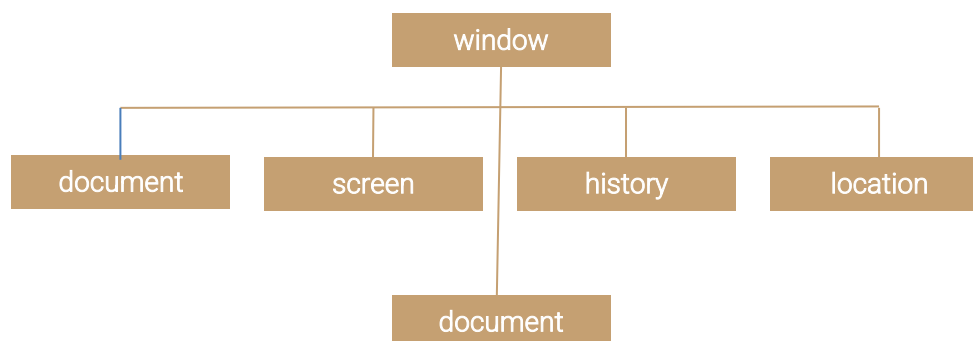


Figura 3.2. Estructura jerárquica del DOM.

En primer lugar, se estudia el objeto `window()`, y enlazado con él se estudia el objeto `document()`, que es el que permite acceder al árbol DOM de la página o pestaña del navegador actual. A continuación se estudian el resto de objetos que dependen de `window()`.

El objeto `document()` enlaza directamente con `window()` y contiene toda la jerarquía de objetos dentro de la página HTML.

Niveles del DOM

El W3C ha establecido los siguientes niveles o versiones de soporte DOM:

- Nivel 0: esta primera versión fue introducida por Netscape y Microsoft en los años noventa, y no es realmente un estándar. Se utilizaba para manipular HTML.
- Nivel 1: el W3C toma el mando y DOM1 es la primera versión estándar que publica. Es una API que permite el acceso o cambio del contenido del documento web, también permite reorganizar contenido. No permite acceder al DTD.
- Nivel 2: además de las capacidades del nivel 1, permite acceder al DTD, hojas de estilo y espacios de nombres. Permite trabajar con estilos dinámicos y eventos.
- Nivel 3: además de heredar toda la funcionalidad del DOM2, añade más métodos y propiedades y modifica algunos anteriores.

3.2. Objeto window()

El objeto `window()` es el principal en la jerarquía de objetos y contiene las propiedades y los métodos que permiten controlar la ventana del navegador. De él dependen todos los demás objetos de la jerarquía.

Por ese motivo se dice que es el contenedor principal de todo lo que se muestra en el navegador, y es el primer objeto que se carga en memoria cuando se abre una ventana o una pestaña en el navegador. A partir de él se pueden crear nuevos objetos `window()` que serán nuevas ventanas que se pueden controlar desde la ventana padre. También permite cerrar ventanas, mostrar mensajes, etcétera.

Los documentos se cargan dentro de la sección del objeto `window()`, y desde él se controla todo lo relativo a la ventana: barras de desplazamiento, tamaño de la ventana, barra de herramientas, de estado, etcétera.



IMPORTANTE

Para JavaScript, tanto las ventanas de navegador como las pestañas son ambos objetos `window()`.

El acceso a las propiedades y los métodos del objeto `window()` normalmente se hace de la forma:

```

window.nombrePropiedad
window.nombreMétodo( [parámetros] )

```

Los parámetros van entre corchetes, ya que son opcionales, y dependerá del método al que se llame si tiene parámetros o no.

Como el nombre del objeto `window()` siempre debe estar en la llamada, se puede obviar siempre que se esté referenciando a objetos dentro de esa misma ventana.

```

nombrePropiedad
nombreMétodo( [parámetros] )

```

La siguiente imagen muestra la jerarquía de objetos creados por el navegador al cargar una página web.

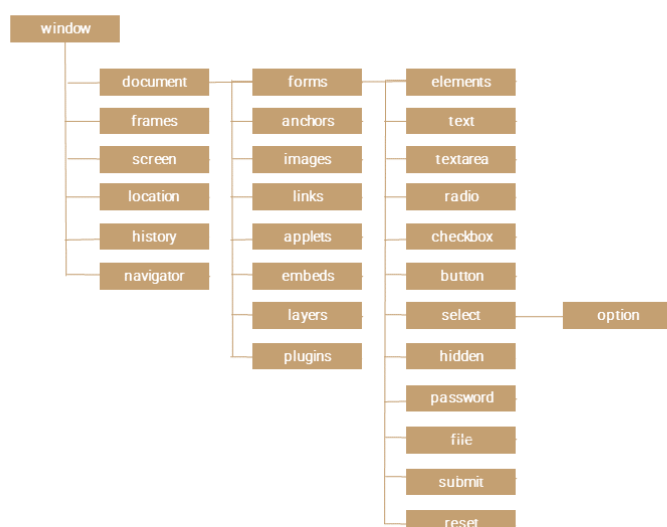


Figura 3.3. DOM generado al cargar una página web.

A continuación detallamos las propiedades y los métodos asociados a las ventanas en relación con su función.

A. Dar nombre a la ventana

Para asignar nombre u obtener el nombre de una ventana se utiliza la propiedad `name` del objeto `window()`,

Ejemplo de sintaxis:

```

document.write( "El nombre de la ventana es: [" + window.name +
"]<br/>" );
window.name = "ventana";
  
```

B. Abrir y cerrar ventana

Para abrir y cerrar una ventana se utilizan los métodos `open()` y `close()`, respectivamente, del objeto `window()`.

La ventana principal del navegador la abre el usuario al acceder a una URL, y desde un script de JavaScript no se puede crear esta ventana, pero sí se pueden abrir ventanas dentro de ella.

Los parámetros de `open()` pueden ser hasta tres:

- **URL** (opcional): dirección de la página web a cargar en la ventana creada. Si se indican comillas seguidas se abre una ventana en blanco.
- **Nombre** (opcional): el nombre de la ventana.
- **Opciones** (opcional): permite indicar las características de la nueva ventana, como son la altura de la ventana (`height`), el ancho de la ventana (`width`), mostrar la barra de direcciones (`location`), la barra de menú (`menubar`), si permite redimensionar la ventana (`resizable`),

mostrar barras de scroll (scrollbars), etcétera.

Utilizando el método `close()` solo se pueden cerrar las ventanas abiertas con `open()`.

El código siguiente crea una variable que, al ser global, es accesible desde cualquier punto del código JavaScript y, por lo tanto, podrá cerrar la ventana desde cualquier punto del programa:

```
var miVentana;

function abrir() { miVentana = window.open( "", "ventana",
"height=200,width=700,location=yes,resizable=no,scrollbars=yes" ); }
```

Para cerrar la ventana hay que referenciar la ventana a cerrar, ya que, si se utiliza directamente `window.close()`, se podría entender que se quiere cerrar la propia ventana y no la subventana `miVentana` creada.



EJEMPLO 9

Archivo ejemplo_9.html

Muestra la tarea de crear y abrir una subventana y luego cerrarla.

En el ejemplo aparecen nuevos elementos de JavaScript:

- `document.getElementById()` es un método del objeto `document()` que devuelve una referencia al elemento por su identificador `id`. La sintaxis es:
- `elemento = document.getElementById(id);`
donde `id` es una cadena (case sensitive) referida al identificador único del elemento buscado.
- `onload()`: incidimos de nuevo en él. Es un evento que se utiliza para ejecutar una función de JavaScript en cuanto una página web se haya cargado.

Vamos a explicar más a fondo el evento `onload()` y el motivo por el cual hay que incluirlo.

Cuando JavaScript interactúa con algún elemento HTML incluido en el body es necesario utilizar una función de inicio y a continuación el evento `onload()`.

¿Por qué se necesita esta función de inicio y cuál es su función?

Si JavaScript ha de referenciar un elemento HTML por cualquier motivo, como por ejemplo seleccionarlo, modificar su contenido, cambiarle el estilo CSS, etcétera, JavaScript necesita conocer ese elemento, saber que existe. Y en este punto es donde entra la ubicación del código JavaScript. Dicho código se puede ubicar en el `<head>` directamente o en un archivo .js externo vinculado desde el `<head>`, pero también se podría ubicar en el `<body>`. Y podría darse la circunstancia de que se ejecutara código JavaScript en `<head>` sin haberse cargado los elementos HTML del `<body>`.

La misión de la función de inicio es contener todas las acciones que se tienen que ejecutar de forma automática al cargarse la página web y los eventos de esta. Pero eso nos daría problemas por lo dicho anteriormente.

Su sintaxis genérica sería:

```
function inicio() {
    // Acciones iniciales y eventos de la página web
}
```

Lo que se quiere realmente es que todas las acciones incluidas en esta función no se ejecuten de forma automática al iniciarse la página, y para ello se utiliza el evento `onload()`. El código de la función de inicio sería como la declaración de la función, y realmente se ejecutarían estas acciones cuando la ventana del navegador haya cargado todo el archivo HTML, y el evento `onload()` sería el encargado de dicha ejecución.

La llamada al evento `onload()` se escribe siempre en el `<body>` y al inicio del código JavaScript. Consta de las siguientes partes:

1. **window**: indica que es la ventana del navegador la encargada de procesar la orden.
2. **onload**: indica el evento y que se activará solo cuando la ventana del navegador haya cargado completamente el HTML.
3. **inicio**: nombre de la función que contiene todas las acciones a ejecutar al iniciarse la página web.

La sintaxis sería: `window.onload = inicio`

Veamos ahora cómo acceder a las ventanas desde JavaScript.

C. Dar y quitar el foco a una ventana

En general, dar el foco significa activar un elemento y/o ponerlo en primer plano. Los métodos implicados son:

`focus()`: para dar el foco a una ventana.

`blur()`: para quitar el foco a una ventana.



EJEMPLO 10

Archivo ejemplo_10.html

El ejemplo muestra cómo abrir una ventana y luego se le quita el foco, devolviéndolo a la ventana padre.

Hay que tener en cuenta que es posible que el navegador tenga bloqueadas las ventanas emergentes, y además el método `blur()` no funciona en todos los navegadores. En este [enlace](#) puedes comprobar en qué navegadores funciona y a partir de qué versiones.

Las propiedades y los métodos aplicables a la gestión de ventanas son los siguientes:

Propiedades	Descripción
<code>closed</code>	Valor booleano. Indica la posibilidad de que se haya cerrado la ventana.

defaultStatus	Texto que se escribe por defecto en la barra de estado del navegador.
document	Devuelve una referencia al documento que contiene la ventana.
history	Objeto historial de páginas visitadas.
length	Devuelve el número de frames (incluyendo iframes) que hay dentro de una ventana.
location	Devuelve la localización del objeto ventana (URL del fichero).
name	Ajusta o devuelve el nombre de una ventana.
navigator	Devuelve el objeto navigator de una ventana.
opener	Devuelve la referencia a la ventana que abrió la ventana actual.
Parent	Devuelve la ventana padre de la ventana actual.
self	Devuelve la ventana actual. Se autorreferencia.
status	Ajusta el texto de la barra de estado de una ventana.

Tabla 1. Propiedades para la gestión de ventanas.

Los métodos más significativos del objeto `window()` son los siguientes:

Métodos	Descripción
<code>alert()</code>	Muestra una ventana emergente de alerta y un botón de aceptar.
<code>blur()</code>	Elimina el foco de la ventana actual.
<code>close()</code>	Cierra la ventana actual.
<code>confirm()</code>	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
<code>focus()</code>	Coloca el foco en la ventana actual.
<code>open()</code>	Abre una nueva ventana de navegación.
<code>prompt()</code>	Muestra una ventana de diálogo para introducir datos.

Tabla 2. Métodos para la gestión de ventanas.

En este [enlace](#) hay disponible más información sobre las propiedades y los métodos del objeto `window()`.

D. Controlar la navegación

JavaScript permite al usuario obtener información del navegador web y configurar algunos aspectos. Para ello se utilizan los siguientes objetos JavaScript:

1. `navigator()`: contiene información sobre el navegador web.
2. `screen()`: permite obtener información sobre la pantalla del usuario.

3. `history()`: permite acceder al historial de navegación.
4. `location()`: proporciona información sobre la URL actual.

El objeto `window()` sabemos que permite controlar la ventana del navegador web y puede referenciar a todos los objetos anteriores de la forma `window.navigator`, `window.location`, etcétera.

Veamos cada uno de ellos:

1. `navigator()`: este objeto proporciona información sobre el navegador web.

La tabla siguiente contiene las propiedades del objeto `navigator()` y lo que devuelve cada una de ellas:

Propiedad	Descripción
<code>appName</code>	Nombre en código del navegador.
<code>appVersion</code>	Información sobre la versión.
<code>cookieEnabled</code>	Muestra si las cookies están activadas.
<code>platform</code>	Sistema operativo para el que está compilado.
<code>userAgent</code>	Información sobre las cabeceras enviadas por el navegador web.

Tabla 3. Propiedades del objeto `navigator()`.



EJEMPLO 11

Archivo `ejemplo_11.html`

El ejemplo define una función `navegador()` que visualiza las características devueltas por las propiedades del objeto `navigator()`.

2. `screen()`: este objeto permite obtener información sobre la resolución de la pantalla que está utilizando el usuario.

El objeto `screen()` tiene las siguientes propiedades, que devuelven la siguiente información:

Propiedad	Descripción
<code>availHeight</code>	Alto en píxeles (excluyendo la barra de tareas).
<code>availWidth</code>	Ancho en píxeles (excluyendo la barra de tareas si estuviera vertical).
<code>height</code>	Alto en píxeles.
<code>width</code>	Ancho en píxeles.

Tabla 4. Propiedades del objeto `screen()`.



EJEMPLO 12

Archivo ejemplo_12.html

El ejemplo muestra la utilización de las propiedades del objeto `screen()` para conocer las características de la pantalla.

3. `history()` : este objeto muestra información sobre el historial de navegación.

El objeto `history()` tiene la propiedad `length`, que indica el número de URL visitadas. Y tiene los siguientes métodos:

Método	Acción en el navegador web
<code>back()</code>	Carga la URL anterior, como si se pulsara el botón de ir atrás.
<code>forward()</code>	Carga la siguiente URL, como si se pulsara el botón adelante.
<code>go()</code>	Carga una URL de las visitadas y guardadas en el historial.

Tabla 5. Métodos del objeto `history()`.

4. `location()` proporciona información sobre la URL que se está visitando.

El objeto `location()` tiene las siguientes propiedades que devuelven la siguiente información:

Propiedad	Descripción
<code>host</code>	El servidor y el puerto utilizados. Esta propiedad permite también cargar una URL.
<code>hostname</code>	El nombre del servidor.
<code>href</code>	La URL completa.
<code>pathname</code>	Obtiene solo la ruta.
<code>port</code>	El puerto de acceso.
<code>protocol</code>	El protocolo usado.
<code>search</code>	Los parámetros en la URL, si los hubiese.

Tabla 6. Propiedades del objeto `location()`.

El objeto `location()` tiene los siguientes métodos:

Método	Acción en el navegador web
<code>assign()</code>	Carga la URL que indiquemos como parámetro.
<code>reload()</code>	Recarga la URL actual.
<code>replace()</code>	Carga la URL que se indica como parámetro, sin reemplazar en el historial

	de navegación a la que había sido cargada previamente.
--	--

Tabla 7. Métodos del objeto `location()`.



EJEMPLO 13

Archivo ejemplo_13.html

Utilización muy sencilla del objeto `location()`. Se trata de mostrar un botón que dé opción a ir a la página de un diario y solicitar confirmación.

3.3. Objeto Document

Existe un objeto `Document` para cada página cargada en una ventana. Dicho objeto contiene todos los elementos visibles de la página.

El objeto `document ()` enlaza directamente con `window ()` y contiene toda la jerarquía de objetos dentro de la página HTML.

Dado el siguiente código HTML, a continuación se muestra la estructura del DOM correspondiente:

```
<html lang="es">
<head>
<meta charset=utf-8" />
<title>Ejemplo de DOM</title>
</head>
<body>
<h1>Título de nivel 1</h1>
<p>Párrafo</p>
</body>
</html>
```

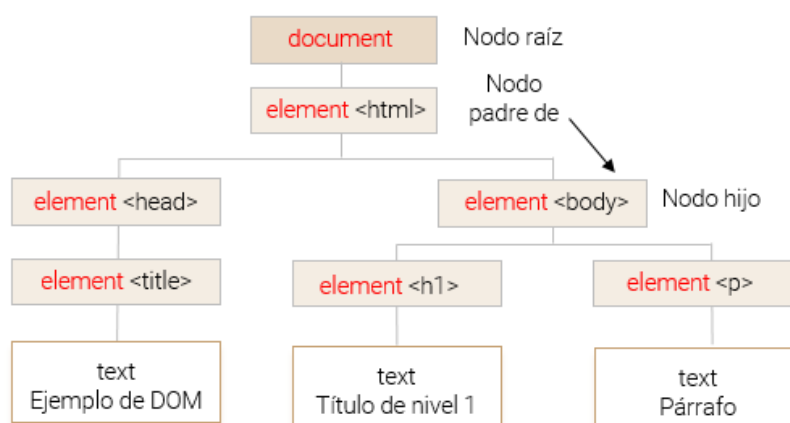


Figura 3.4. DOM creado para el código HTML.

Si, por ejemplo, el párrafo tuviera un atributo:

```
<p>Párrafo <strong>elemental</strong></p>
```

En este caso, el DOM genera los siguientes nodos:

- Nodo de tipo `element` para la etiqueta `<p>`.
- Nodo de tipo `text` con el 'Párrafo' de la etiqueta `<p>`.

- Como el contenido de `<p>` incluye en la etiqueta HTML `<strong>`, que es un atributo, esta etiqueta interior se transforma en un nodo de tipo `element` que la representa y que es hijo del `element <p>`.
- El contenido de la etiqueta `<strong>` genera a su vez otro nodo de tipo `text` que es hijo del nodo `<strong>`.

El DOM define 12 tipos de nodos, pero las páginas HTML normalmente manejan los siguientes:

- `document`: es la raíz de la estructura en árbol.
- `element`: nodo hijo del raíz, identifica la etiqueta `<html>` y a su vez es padre de otros nodos `element` que contienen etiquetas HTML.
- `attr`: identifica los atributos de las etiquetas de la forma "atributo=valor".
- `text`: identifica el texto encerrado por una etiqueta.
- `comment`: identifica los comentarios incluidos en la página HTML.

Acceder a los nodos	Añadir nodos al DOM	Eliminar nodos
• <code>getElementById()</code> • <code>getElementsByTagName()</code> • <code>getElementsByClassName()</code> • <code>getElementsByName()</code> • <code>parentNode</code> • <code>childNodes</code> • <code>firstChild</code>	Crear nodos • <code>createElement()</code> • <code>createTextNode()</code> Añadir nodos • <code>appendChild()</code> • <code>insertBefore()</code>	• <code>removeChild()</code> • <code>replaceChild()</code>
Modificar contenido de nodos	Modificar atributo style	Modificar atributo class
• Contenido elementos • <code>innerHTML</code> • <code>textContent</code> • Atributo elementos • <code>setAttribute()</code> • <code>getAttribute()</code> • <code>removeAttribute()</code> • <code>hasAttribute()</code>	• Acceder al objeto <code>style</code> • Dar propiedades <code>style</code> • Crear un objeto <code>style</code>	• <code>className</code> • <code>classList</code> • <code>add()</code> • <code>remove()</code> • <code>toggle()</code> • <code>contains()</code> • <code>items()</code>

Figura 3.5. Métodos y propiedades del DOM.

En el siguiente [enlace](#) hay disponible más información sobre el objeto `document()`.

Explicamos el funcionamiento de alguno de estos métodos y sus propiedades más importantes.

ACCEDER A LOS NODOS

Una vez se dispone ya del DOM, ya se pueden utilizar los métodos DOM para acceder directamente a cualquier nodo del árbol. Y acceder a un nodo equivale a acceder a una zona de la página y, por lo tanto, se puede gestionar esa zona de la página, como, por ejemplo acceder al valor de un elemento, asignarle valor a un elemento, crear y añadir nuevos elementos, etcétera.



IMPORTANTE

El acceso a los nodos y su gestión no será posible hacerla si el DOM no se ha construido, es decir, solo es posible cuando la página HTML se ha cargado por completo.

Vamos a ver algunos de estos métodos DOM:

A. Método `getElementById()`

Es el método más utilizado en general y, en particular, para el desarrollo de aplicaciones web dinámicas. Con este método se accede directamente a un nodo a través de su id (identificador) y devuelve el elemento que posee el atributo "id" con el valor especificado. Como el identificador id debe ser único para cada elemento de una misma página, el método devuelve solo el nodo que se quiere. ¿Cómo funciona? En primer lugar, se llama o invoca el objeto implicado, que es `document`, y a continuación el método `getElementById`. Como parámetro se le pasa el id del elemento que se quiere que devuelva.



EJEMPLO 14

Archivo ejemplo14.html

```
<div id="titulo">
  <h1>Ejemplo 14 Uso de getElementById()</h1>
</div>
<script>
  var tituloTexto = document.getElementById("titulo");
</script>
```

La ejecución de este archivo muestra una ventana con el título dado.

B. Método `getElementsByTagName()`

El método `getElementsByTagName(nombreEtiqueta)` devuelve en un array todos los elementos (nodos DOM) de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa al método. Para acceder a uno de ellos en concreto debemos proceder como ya vimos en el apartado anterior de los arrays.

Ejemplos:

```
var tituloSegundo = document.getElementsByTagName("h2")[1];
```

En este caso devolverá, de todos los elementos del documento con una etiqueta específica h2, el segundo elemento.

```
var parrafos = document.getElementsByTagName("p");
```

En este caso devolverá todos los párrafos de la página HTML. Si del array almacenado en parrafos lo que se quiere es el primer párrafo habría que escribir:

```
var primerParrafo = parrafos[0];
```

O recorrerlos todos mediante un bucle for:

```
for(var i=0; i<parrafos.length; i++) {
    var parrafo = parrafos[i]; }
```

C. Método document.querySelector()

El método `document.querySelector()` devuelve el primer elemento (nodo) que coincida con un selector CSS especificado. El selector o selectores CSS se pasan al método como parámetros.

Se puede indicar cualquier selector CSS, y en el identificador hay que añadir delante '#'.



EJEMPLO 15

Archivo ejemplo_15.html

El ejemplo utiliza el atributo "id" y el tipo de elemento, y muestra diferentes líneas de texto con diferentes atributos y, en concreto, al elemento h3 se le cambia el color desde JavaScript.

D. Método document.querySelectorAll()

Este método devuelve un array con todos los nodos que coincidan con el selector CSS. Para acceder a uno de ellos en concreto debemos proceder como ya vimos en el apartado de los arrays.



EJEMPLO 16

Archivo ejemplo_16.html

Siguiendo la línea del ejemplo anterior, con este método podemos acceder a cualquiera de los elementos tipo h3 que contenga el documento web y cambiar el estilo CSS indicando, en este caso, cuál sería el nuevo color solo para uno de ellos.

CREAR/ELIMINAR NODOS AL DOM. INSERTAR NODOS

Para insertar nodos en el DOM, en primer lugar hay que crearlos, y en segundo lugar, saber exactamente en qué punto del documento web se quiere insertar.

Para crear el nodo JavaScript dispone de dos métodos de `document()` :

A. Método `createElement()`

Crea un nodo de tipo `element`. La clase de elemento HTML que se quiere crear se le dice al método mediante el parámetro. Por ejemplo: `div`, `p`, `ul`, etcétera.

El ejemplo siguiente crea un elemento del tipo `h2` y se le asigna a la variable `nuevoElemento`.

```
var nuevoElemento = document.createElement("h2");
```

B. Método `createTextNode()`

Crea un nodo de tipo `text`. El texto se le pasa como argumento al método.

```
var nuevoElementoTexto = document.createTextNode("Nuevo elemento texto");
```

En el ejemplo se ha creado un elemento tipo `'text'` con el contenido `'Nuevo elemento texto'` y se le asigna a la variable `nuevoElementoTexto`.



IMPORTANTE

Hasta ahora lo que se ha hecho es crear nuevos nodos en el DOM. Pero si se quiere que estos nodos formen parte del documento HTML hay que insertarlos.

INSERTAR NODOS

Ya creado el nodo, se puede insertar o añadir al documento HTML. Para ello, JavaScript dispone de los métodos siguientes:

A. Método `appendChild()`

Añade un nodo como hijo de otro. Si ya existen hijos para ese nodo, este nuevo nodo se añadirá en la última posición. Se puede utilizar para todos los elementos de HTML.

```
nodoPadre.appendChild(nodoHijo);
```

Veamos cómo funciona con los ejemplos anteriores.

```
var nuevoElemento = document.createElement("h2");
var nuevoElementoTexto = document.createTextNode("Nuevo elemento texto");
nuevoElemento.appendChild(nuevoElementoTexto); /* se ha añadido el nodo
'nuevoElementoTexto' como hijo de 'nuevoElemento' */
document.body.appendChild(nuevoElemento); /* se ha añadido
'nuevoElemento' como hijo del elemento body */
```

Como se puede observar, la llamada al método `appendChild()` ha sido doble.

1. La primera vez, para indicar el texto del nuevo elemento de tipo `text` e insertarlo como hijo del nodo tipo `element` llamado `'nuevoElemento'`. Sería equivalente en HTML a haber añadido:

```
<h2>Nuevo elemento texto</h2>
```
2. La segunda, para insertar el nodo `'nuevoElemento'` como hijo del `body` del documento HTML.

B. Método `insertBefore()`

Añade un nodo como hijo de otro, pero en este caso se puede elegir la posición. Hay que indicar al método delante de qué nodo se quiere insertar. Se puede utilizar para todos los elementos de HTML.



EJEMPLO 17

Archivo `ejemplo_17.html`

En el ejemplo tenemos una lista con 3 módulos de 1.º de DAM y se quiere añadir en la segunda posición la línea correspondiente al módulo «Lenguaje de marcas».

En primer lugar, se crean los nodos `text` y el elemento de la lista `li`. A continuación se añade el texto al elemento `li` creado. Luego se obtiene el segundo elemento de `li` y el elemento padre `ul` con el método `getElementByTagName()`, y por último se añade el `nuevoElemento` delante del `segundoElemento` (hay que escribir los parámetros en el orden indicado). Observar que la llamada a `insertBefore()` se hace a través de la variable que ha obtenido el acceso al padre de la estructura en lista `ul`.

C. Método `removeChild()`

Elimina un nodo hijo de un elemento dado. Hay que llamar al método desde el nodo padre y se le pasa como parámetro el nodo hijo.



EJEMPLO 18

Archivo `ejemplo_18.html`

En el ejemplo se elimina el primer elemento de la lista, correspondiente a Programación.

4. Eventos de JavaScript

Evento es una acción del usuario o del navegador que sucede en el sistema y este informa para que se pueda responder. si se quiere. a dicha acción

Un evento es, por ejemplo, cuando el usuario hace clic sobre un botón en una página web, cuando el cursor se posiciona encima de una imagen o cuando el navegador carga la página web.

Una característica importante de los eventos es que se pueden manejar cuando se producen. Esto se hace a través de los “manejadores de eventos” que, en realidad, son funciones asociadas que se ejecutan cuando se produce el evento. Es decir, el manejador es el código que responde al evento.

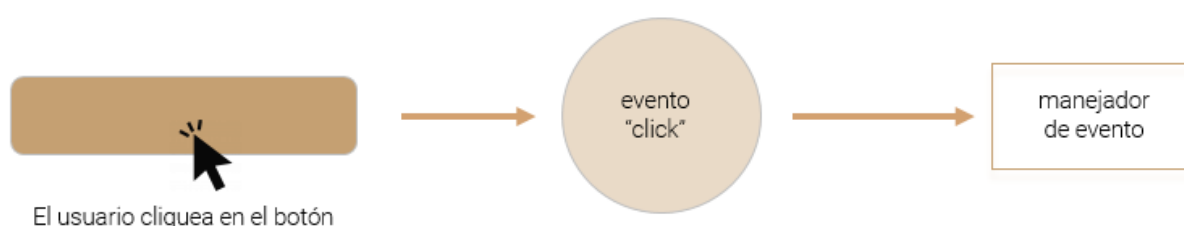


Figura 4.1. Estructura del evento.

La gestión de eventos de JavaScript consiste en asociar el “manejador de evento” a un elemento HTML, y esto se puede hacer de tres formas:

- Desde HTML: mediante un atributo de evento.
- Desde JavaScript:
 - mediante una propiedad de evento.
 - Desde el nuevo método estándar `addEventListener()`.

4.1. Mediante un atributo de evento (HTML)

Consiste en utilizar los atributos de eventos de los elementos de un documento HTML5. Estos atributos permiten asociar al elemento un determinado comportamiento cuando se produzca el evento. La figura siguiente muestra el esquema de funcionamiento para el atributo de evento `onclick()`.

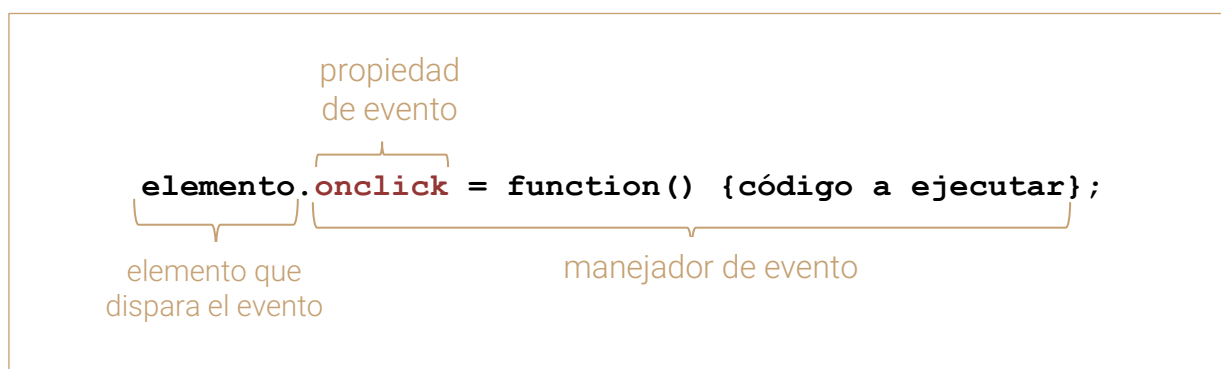


Figura 4.2. Funcionamiento del evento `onclick()`.

Por ejemplo:

```
<input type="button" onclick="alert('¡Hola!');" value="saluda"/>
```

En este ejemplo, al elemento HTML `<input>` se le ha asignado un atributo de evento, `onclick`. El valor asignado a este atributo es el código JavaScript que ejecutará la acción asociada; en este caso, mostrará el texto "¡Hola!".

Este mecanismo es sencillo, pero se puede complicar cuando hay varias acciones a ejecutar. En este caso se utilizan las llamadas a funciones que recopilan todas las acciones.

El ejemplo anterior se transformará en:

```
<input type="button" onclick="saludar()" value="saluda"/>
<script>
    function saludar() {
        alert("¡Hola!");
    };
</script>
```

4.2. Mediante una propiedad de evento (Javascript).

Consiste en asignar un manejador al elemento HTML desde Javascript sin necesidad de añadir atributos de eventos en HTML. De esta forma, los códigos HTML y JavaScript permanecen separados y el código está mejor organizado.

La figura siguiente muestra la sintaxis de funcionamiento.

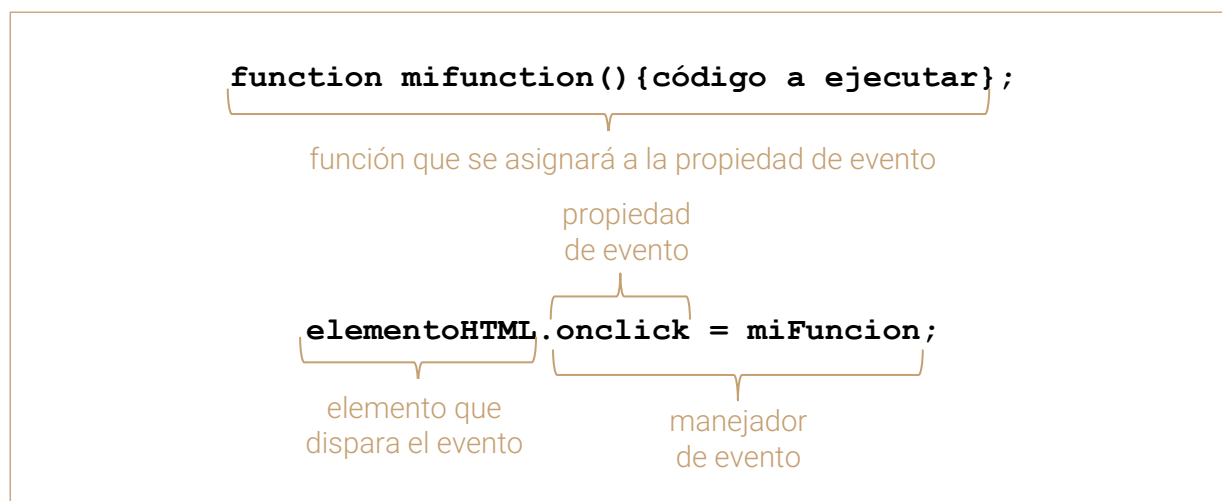


Figura 4.3. Propiedad de evento.



EJEMPLO 19

Archivo ejemplo_19.html

En el ejemplo 19, al formulario identificado por su id “formulario” se le asigna el manejador. A continuación se asigna a este manejador la función `Color()`, con la que se cambia el color de fondo de una caja de texto a la que se accede por su nombre. Para finalizar, se asigna al elemento HTML que dispara el evento la función `Color()` a través del manejador.

```
document.getElementById("Boton").onclick = Color;
```

Otra opción es asignar la función directamente al manejador en vez de hacer la llamada.

En nuestro ejemplo quedaría así:

```
document.getElementById("Boton").onclick = function Color() {  
document.formulario.caja_texto.style.backgroundColor='#f88067';}
```



IMPORTANTE

La asignación al manejador debe hacerse al final del documento para estar seguros de que todos los elementos implicados están ya cargados. Para ello se utiliza el evento `onload()` del objeto `window()`. Por ejemplo:

```
window.onload = function() { document.getElementById("id ").onclick =  
muestraMensaje;}
```

4.3. Mediante el método `addEventListener()`

El método `addEventListener()` modifica el DOM al cargar el documento y permite añadir eventos a los nodos de un documento web.

El método `addEventListener()`, conocido como “escuchar al evento”, permanece atento a la interacción de un usuario sobre un elemento y contiene el código necesario para responder a ese evento. El elemento puede ser cualquier elemento HTML5 existente en el documento web.

```
elemento_que_se_escucha.addEventListener('evento',función_a_lanzar,booleano);
```

`elemento_que_se_escucha` es cualquier elemento del documento, al que se puede acceder por su id o por su etiqueta.

Este método tiene tres argumentos:

- el nombre del evento.
- la función a ser ejecutada.
- un valor booleano que define el orden del flujo de eventos cuando hay varios eventos posibles.

Veamos el significado del valor booleano

- **true:** el evento se propaga de forma descendente (árbol DOM) desde la rama en la que está el elemento.
- **false:** el evento se propaga de forma ascendente desde el elemento para el cual se ha definido, dentro del árbol DOM.

El método `addEventListener()` es el recomendado por el W3C dentro del llamado modelo de registro avanzado de eventos.



EJEMPLO 20

Archivos `ejemplo_20.html` y `ejemplo_20.css`

El ejemplo 20 muestra cómo capturar un evento `click` utilizando el escuchador de eventos `addEventListener()`.

De la misma forma que el método `addEventListener()` nos permite capturar un evento, el método `removeEventListener()` nos permite eliminar un manejador de evento creado por `addEventListener()`. Ambos métodos tienen la misma sintaxis.

En la sección [Material de trabajo > Descarga de documentos](#) tienes disponible información sobre otra forma alternativa de asociar el “manejador de evento” a un elemento HTML (archivo [this.pdf](#)).

A. Clasificación de eventos

Un criterio de clasificación de los eventos es en base a la interacción del usuario con ellos:

- **Eventos del ratón:** el usuario utiliza el ratón para desencadenarlo.
- **Eventos de teclado:** el usuario utiliza el teclado para desencadenarlo.
- **Eventos de página:** cuando cambia el estado de la página (se carga, se descarga, se redimensiona, etcétera).
- **Eventos de formulario:** cuando cambia el estado de algún elemento de un formulario.



IMPORTANTE

En los siguientes enlaces están disponibles los listados completos de los eventos de JavaScript. Accediendo a cada uno de ellos se puede conocer en profundidad su funcionamiento, el nivel del DOM en el que están incluidos y la compatibilidad con los diferentes navegadores web.

[Eventos estándar.](#)

[Eventos no estándar.](#)

B. Ejecución de JavaScript desde enlaces

Se pueden utilizar los enlaces como mecanismo de llamada a la ejecución de código JavaScript como si se tratara del evento `onclick`. En este caso, al pulsar sobre el enlace se ejecutará el script.

La sintaxis del enlace es:

```
<a href="javascript: funcionJavascript()">Texto del enlace</a>
```

En el atributo `href` se escribe la palabra 'javascript' seguida de dos puntos, y a continuación se llama a la función JavaScript que contiene el código.



EJEMPLO 21

Archivo ejemplo_21.html

Este ejemplo muestra cómo utilizar los enlaces para ejecutar código JavaScript. Al pulsar en los enlaces el texto cambia de color.

En [Descarga de documentos](#) está disponible el archivo [eventos.pdf](#) con información relativa a los diferentes eventos disponibles para la gestión del ratón y teclado. Su contenido es una continuación del apartado correspondiente a eventos en JavaScript.

5. Formularios. Validación

Al cargar una página web, el navegador crea, de forma automática, un array llamado `forms` que contiene la referencia a todos los formularios de la página y un array `elements` para cada uno de ellos.

El objeto `form` es un subobjeto del objeto `document()` dentro de la jerarquía del DOM y se corresponde con la etiqueta `<form>` de HTML.

Un formulario puede ser enviado de dos formas:

- ejecutando el método `submit` de JavaScript.
- haciendo clic en el botón `submit` del formulario.

Hay dos métodos en JavaScript para obtener la referencia a un elemento de un formulario:

1. Accediendo al array `forms` y al array `elements` y, por lo tanto, haciendo uso de la sintaxis propia del recorrido de arrays.

Por ejemplo, la línea `document.forms[0]`; permite acceder al primer formulario de la página.

Para acceder a cada uno de los elementos que componen cada formulario (botones, listas desplegables, etcétera) se utiliza el array `elements`.

Por ejemplo, la línea `document.forms[0].elements[0]`; hace referencia al primer elemento del primer formulario de la página web.

Pero ¿qué ocurriría si en la página web original se cambian o añaden formularios? Cambia todo y no sirve el código JavaScript que se haya escrito. Por ese motivo no se aconseja utilizar directamente el array `document.forms` y se recomienda referenciar, tanto los formularios como sus elementos, mediante su nombre (atributo `name`) o su identificador (atributo `id`).

Por ejemplo:

```
var formularioPrincipal = document.formulario1;
var formularioSecundario = document.formulario2;
<form name="formulario1" >
  <input type="text" name="elemento1" /></form>
<form name="formulario2" >
  <input type="text" name="elemento2" /></form>
```

2. Utilizando los métodos DOM de acceso a nodos mediante su identificador. Por ejemplo, utilizando el método `getElementById()`.

Por ejemplo:

```
var formularioPrincipal = document.getElementById("formulario1");
var primerElemento = document.getElementById("elemento1");
<form name="formulario1" id="formulario1" >
  <input type="text" name="elemento1" id="elemento1" />
</form>
```


PROPIEDADES DEL OBJETO FORM

Propiedad	Descripción
action	Define la URL a la cual se envía el formulario para ser procesado.
encoding	Define el tipo de codificación de los datos que se utilizará al enviar el formulario (por ejemplo, text/plain).
length	Indica el número de formularios existentes en la página.
method	Determina el método de envío del formulario (GET o POST).
name	Nombre del formulario, que debe ser nombre único.
target	Define la ventana en la que se mostrarán o procesarán los resultados del formulario.

Tabla 1. Propiedades del objeto *form*.

MÉTODOS DEL OBJETO FORM

Método	Descripción
submit	Envía el formulario.
reset	Restablece el formulario a los valores por defecto.

Tabla 2. Métodos del objeto *form*.

Ejemplo de sintaxis:

```
<form name="formulario " action="http://localhost/UD4/formulario.php"
method="POST" target="_blank">

    /*Aquí vendrían los campos del formulario.... */

    <input type="text" name="nombre"><br>
    <input type="submit" value="enviar formulario"><br>
    <input type="reset" value="borrar">
</form>
```

La sintaxis sería similar en el caso de utilizar el método GET.

La tabla siguiente muestra los eventos del objeto *form* más importantes.

Evento	Descripción
onSubmit	Es el evento más importante del objeto <i>form</i> y se produce cuando se envía el formulario, bien porque se ha pulsado el botón submit, bien porque se ha ejecutado una función JavaScript que llama al método <i>submit()</i> que envía el formulario.
onReset	Se produce cuando se restablece el formulario. Funciona igual que el evento anterior.
onclick	Se produce cuando se hace clic con el ratón sobre un elemento. Se utiliza con los tipos de botones de HTML (<input type="button">, <input

	<code>type="submit"></code> , <code><input type="image"></code>).
onchange	Se produce cuando el usuario cambia el valor de un elemento de texto (<code><input type="text"></code> o <code><textarea></code>) o se selecciona una opción en una lista desplegable (<code><select></code>). Pero el evento solo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario.

Tabla 3. Eventos del objeto *form*.

El formulario se compone de elementos, y cada uno de ellos tiene las propiedades que muestra la siguiente tabla.

Propiedad	Descripción
type	Indica el tipo de elemento. En elementos tipo <code><input></code> (text, button, checkbox, etcétera.) coincide con el valor de su atributo <code>type</code> . En listas desplegables normales (elemento <code><select></code>) su valor es <code>select-one</code> . En listas que permiten seleccionar varios elementos a la vez, su valor es <code>select-multiple</code> . En elementos tipo <code><textarea></code> , su valor es <code>textarea</code> .
form	Referencia al formulario al que pertenece el elemento. <code>document.getElementById("id_del_elemento").form</code>
name	Es el valor del atributo <code>name</code> de HTML. Solo lee su valor.

Tabla 4. Propiedades de los elementos del objeto *form*.



EJEMPLO 22

Archivo ejemplo_22.html

El ejemplo muestra cómo obtener el número de formularios de una página web, es decir, el array `forms`. Se incluyen en el código dos formularios a modo de comprobación, pero no hacen nada.



EJEMPLO 23

Archivo ejemplo_23.html

El ejemplo muestra cómo obtener el número de elementos de todos los formularios de una página web. Se incluyen en el código dos formularios a modo de comprobación, pero no hacen nada.

Envío y validación de formularios

Validar un formulario implica chequear el formulario y comprobar que todos los datos han sido introducidos correctamente. Por ejemplo, si el formulario contiene un campo que debe tener un correo electrónico o un DNI, habrá que comprobar que dicho correo o DNI está bien escrito antes de pasar al siguiente campo.

Los mecanismos de validación de formularios se pueden aplicar en él:

- Lado servidor: se suelen utilizar programas PHP, entre otros.
- Lado cliente: se suelen utilizar scripts de JavaScript.

Validar en el lado servidor es, en general, más seguro, pero también más complejo. Sin embargo, validar en el lado cliente es más sencillo y rápido, ya que el formulario aún no se ha enviado y el usuario recibirá las notificaciones de los errores de forma instantánea. Pero realmente no descarga al servidor de esta tarea, ya que siempre hay que tener en cuenta la posibilidad de que el usuario tenga desactivado el uso de JavaScript en su navegador, con lo cual el proceso de validación en él quedaría anulado.

La validación se puede hacer campo a campo, a medida que se van rellenando, o en el momento en el que se pulse el botón de envío.

¿Qué es lo que se valida al introducir el usuario los datos de entrada?

1. En campos requeridos que exista un valor.
2. En campos numéricos que se haya introducido un valor numérico.
3. En campos que deben seguir un patrón concreto (fecha, correo electrónico, DNI, etcétera) que lo cumpla.



IMPORTANTE

Si el usuario desactiva en su navegador la ejecución de JavaScript automáticamente, este proceso de validación no se realiza. Por ese motivo, el envío del formulario no debe depender solo de JavaScript.

El proceso de validación normalmente se hace mediante una función, y la sintaxis para incorporarla al formulario es la siguiente:

```
<form action="" method="" id="" name="" onsubmit="return validar()">
    /* campos del formulario */
</form>
```

La estructura de la función `validar()` es la siguiente (para los mensajes se ha utilizado `console.log()`):

```
function validar() {
    if (condicion para el primer campo) {
```

```
// Si no se cumple...
console.log('[ERROR] Mensaje al usuario con indicaciones...');
return false;}
else if (condicion para el segundo campo) {
    // Si no se cumple...
    console.log('[ERROR] Mensaje al usuario con indicaciones...');
    return false;}
...
else if (condicion para el último campo) {
    // Si no se cumple...
    console.log('[ERROR] Mensaje al usuario con indicaciones...');
    return false;}
// Llegados a este punto, todas las condiciones se han cumplido
return true;}
```

Comentamos algunos detalles del funcionamiento del evento `onsubmit()`. Si `onsubmit()` devuelve:

- `true`: el formulario se envía. Todas las validaciones han ido bien y el envío del formulario es efectivo.
- `false`: el formulario no se envía. alguna de las validaciones no ha ido bien, hay algún campo incorrecto.

El evento `onsubmit()` del formulario permite controlar si el usuario ha intentado enviarlo, entonces comprueba que los datos son correctos (`onsubmit` recibe `true`), y si no lo son cancela el envío (`onsubmit` recibe `false`).

La llamada al evento `onsubmit()` dentro del formulario se hace siguiendo la sintaxis:

```
onsubmit="return validar() "
```

Por lo tanto, el formulario solo se envía si la función `validar()` devuelve `true`.



EJEMPLO 24

Archivo ejemplo_24.html

El código de este ejemplo crea un formulario con dos campos, nombre y apellidos, y lo que hace es validar que los campos no estén vacíos desde Javascript.

Vemos que la función `validar()` se llama cuando se pulsa el botón Enviar, y lo que hace es comprobar que los campos Nombre y Apellidos no están vacíos. Si lo está alguno de ellos o ambos, muestra un mensaje. Cuando todo está bien se llama al programa en PHP (no existe y el navegador lo avisa) que lo procesa.

Si tuviéramos un formulario y quisiéramos que este se validase antes de ser enviado al servidor, la forma correcta de hacerlo sería mediante una escucha:

```
document.getElementById('enviar').addEventListener('click', validar, false);
```

Y en el formulario HTML:

```
<button type="submit" id="enviar">Enviar formulario</button>
```

A continuación se explica la validación para algunos elementos del formulario. Información más completa está disponible en este [enlace](#).

A. Validar un campo de texto obligatorio

Si en el formulario existen campos obligatorios del tipo text o textarea hay que impedir que se queden vacíos.

Desde JavaScript esto se puede hacer de la forma siguiente:

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^s+$/ .test(valor) ) {
    return false;}
```

Veamos las condiciones de la estructura if:

1ª. El valor introducido no tiene que ser nulo (null), es decir, se debe haber introducido algún valor de tipo objeto, array, numérico, cadena de texto o booleano.

2ª. La longitud del texto introducido debe ser distinta de cero caracteres.

3ª. Es la más compleja, porque lleva una expresión regular (expresiones con comodines que sustituyen caracteres). En este caso es (/^s+\$/ .test(valor)), y concretamente lo que hace es impedir que el valor introducido solo tenga espacios en blanco.

En el [enlace](#) hay disponible más información sobre las expresiones regulares en JavaScript.

B. Validar un campo de texto con valores numéricos

En este caso, el campo tipo text o textarea debe contener un valor numérico. Desde JavaScript esto se puede hacer de la forma siguiente:

```
valor = document.getElementById("campo").value;
if( isNaN(valor) ) {
    return false;}
```

Utiliza la función `isNaN()` (is Not a Number), que comprueba si el valor pasado a la función es un número válido para poder operar con él. La función devuelve true si la variable valor no es un número, y devuelve false si la variable valor es un número.

Ejemplos de valores devueltos por la función `isNaN()` :

```
isNaN("3");           // false
isNaN(3.1416);        // false
isNaN(+14.2);         // false
isNaN("18a");         // true
```

C. Validar una dirección de correo electrónico

En este caso, el campo de tipo texto debe contener una dirección de correo electrónico válida. Desde JavaScript esto se puede hacer de la forma siguiente:

```
valor = document.getElementById("campo").value;
if( !(/\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)]/).test(valor)) ) {
    return false;}
```

Como se puede comprobar, se utiliza también una expresión regular y bastante compleja. Los patrones de los correos electrónicos son variados, y esto complica la expresión regular que los pueda identificar. La dada aquí es de las más sencillas, pero no es la única.

D. Validar una fecha

Validar una fecha es algo complicado por la variedad de formas en que se pueden introducir por parte del usuario.

El siguiente código utiliza la función interna de JavaScript `Date(anyo, mes, dia)` , que permite construir una fecha dados el día, el mes y el año. Hay que tener en cuenta que los meses van del 0 al 11, y que si los datos dados por el usuario no son correctos no se podrá montar bien la fecha y, en consecuencia no se podrá validar el formulario.

```
var anyo = document.getElementById("anyo").value;
var mes = document.getElementById("mes").value;
var dia = document.getElementById("dia").value;
valor = new Date(anyo, mes, dia);
if( !isNaN(valor) ) {
    return false;}
```



Unidad 5.

Estructura de documentos en lenguajes de marcas



1. Definición de la estructura de documentos XML

En este apartado se van a estudiar las características del ejemplar. Comenzamos repasando ambas partes de la estructura: el prólogo y el ejemplar.

Prólogo

Contiene toda la información necesaria para que el documento XML sea correctamente interpretado. Consta de la declaración XML y la declaración de tipo de documento.

La declaración XML (instrucciones de procesamiento) incluye la versión de XML utilizada y la codificación de caracteres utilizada. La declaración es el comienzo del archivo XML:

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

Antes del elemento “declaración XML” no se puede añadir otro elemento, comentarios o espacios en blanco.

La declaración o definición de tipo de documento o DTD indica el tipo de documento que se está creando. Siempre comienza por la cadena:

```
<!DOCTYPE nombre_tipo...>
```

Ejemplar o cuerpo del documento

El cuerpo del documento contiene los datos o el texto del documento XML. La estructura es en árbol con un único elemento raíz o ejemplar, y de él parten una serie de elementos anidados que son los que contienen los bloques de texto. Cada elemento lleva una marca o etiqueta de apertura y una de cierre. Cada elemento puede tener atributos. Globalmente, el cuerpo puede contener:

- Elementos
- Atributos
- Entidades predefinidas
- Instrucciones de proceso
- Secciones CDATA

Comentarios

Se pueden introducir en cualquier lugar del cuerpo o del prólogo, pero nunca dentro de las declaraciones, etiquetas u otros comentarios. La sintaxis es:

```
<!-- Texto de comentario -->
```


1.1. Declaración de tipo de documento

XML es un metalenguaje, y como tal permite que el usuario cree su propio lenguaje, pero para ello siempre debe existir un diccionario que lo defina. Para este fin se utiliza la declaración de tipo de documento DTD.

Por lo tanto, XML permite la creación de nuevos DTD, y por ese motivo, entre otros, este metalenguaje se llama extensible.

La DTD es un archivo de texto cuyo contenido son las definiciones de las etiquetas y sus atributos con los que se puede trabajar en un determinado documento.

La DTD contiene la estructura de los datos en un documento XML con todas las restricciones a tener en cuenta y así permitir su validación. También se le puede llamar gramática o plantilla.

En general, la utilización de la DTD es opcional, aunque conveniente. En función de si se utiliza o no DTD, los documentos XML serán “válidos” o solo “bien formados”.

Un documento XML está “bien formado” si:

1. Sigue las reglas sintácticas.
2. Además:
 - Contiene un único elemento raíz.
 - Todas las etiquetas están correctamente anidadas.

Un documento XML es “válido” si:

1. Está bien formado y
2. La estructura encaja con la declaración del tipo de documento (DTD).

Es decir, si el documento XML puede contener elementos, atributos y entidades debe cumplir lo indicado en las declaraciones de esos elementos, atributos, entidades y notaciones, además de no tener errores sintácticos.

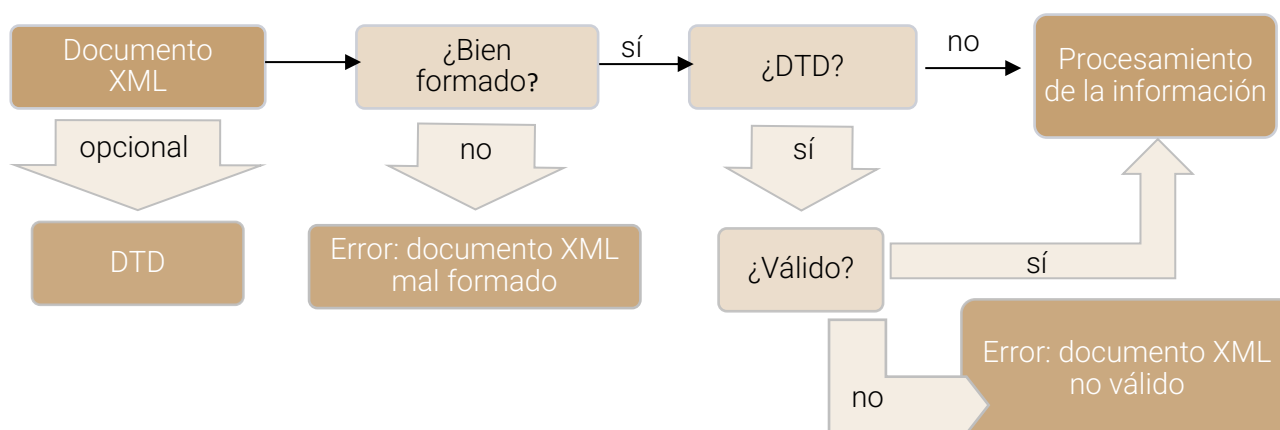


Figura 1.1. Documentos XML válidos vs. bien formados.

La DTD a utilizar para validar el documento XML viene dada en la etiqueta DOCTYPE.

En la sintaxis siempre se incluye el nombre del ejemplar (nombre_tipo) del documento XML.

```
<!DOCTYPE nombre_tipo ...>
```

Una DTD se puede escribir de forma interna o externa al archivo XML. En cualquier caso, siempre hay que escribir DOCTYPE para asociar el documento XML a la DTD.

En resumen, la DTD describe:

1. Los elementos y el orden en el que pueden aparecer en el documento XML.
2. Los elementos obligatorios y opcionales.
3. Los atributos del elemento, ya sean opcionales u obligatorios.
4. Si los atributos pueden tener valores predeterminados.

Por otra parte, un documento XML se puede asociar o vincular a una DTD interna y externa de forma simultánea.

Para los ejemplos introducimos el elemento `<!ELEMENT` y el tipo `#PCDATA`, que veremos con detalle en el apartado 2. Solo decir ahora que con `<!ELEMENT` se identifica cada uno de los elementos declarados, y `#PCDATA` es la regla que determina el tipo de contenido de dicho elemento. Para esta regla el tipo es texto.

A. DTD interna

Si la definición de tipo de documento DTD se incluye en el propio documento la sintaxis es:

```
<!DOCTYPE nombre_tipo [... declaraciones de marcado...] >
```



EJEMPLO 1A

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE libros[
  <!ELEMENT libros(autor)>
  <!ELEMENT autor (nombre, titulo)>
  <!ELEMENT nombre (#PCDATA)>
  <!ELEMENT titulo (#PCDATA)> ]>
<libros>
<autor>
  <nombre>Nombre_Autor</nombre>
  <titulo>Lenguaje de marcas.</titulo>
</autor>
</libros>
```

En la DTD se indica que “libros” es el elemento raíz del documento XML.

En `autor (nombre, titulo)` indica que cada elemento “autor” ha de contener dos elementos (hijos): “nombre” y “titulo”.

En `#PCDATA` (Parsed Character Data), escrito entre paréntesis ‘()’, se indica que los elementos “nombre” y “titulo” pueden contener texto (cadenas de caracteres) analizable por un procesador XML también llamado analizador (*parser*).



IMPORTANTE

La DTD no se muestra en el navegador. Las declaraciones incluidas en ella pertenecen exclusivamente a este documento. y no las puede compartir.

B. DTD externa

Existen dos tipos de DTD externa: privada y pública.

1. Sintaxis de la DTD **externa privada**:

```
<!DOCTYPE elemento-raíz SYSTEM "URI">
```

En este caso, se especifica un URI donde pueden localizarse las declaraciones. En el ejemplo anterior equivaldría a crear un archivo libros.dtd con las líneas:

```
<!ELEMENT libros (autor)>
<!ELEMENT autor (nombre, titulo)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
```

Y en la declaración habría que escribir:

```
<!DOCTYPE libros SYSTEM "libros.dtd">
```

2. Sintaxis de la DTD **externa pública**:

```
<!DOCTYPE elemento-raíz PUBLIC "identificador-público" "URI">
```

Los identificadores públicos son una forma de localizar recursos DTD, son como una entrada a un catálogo. Por ejemplo:

```
<!DOCTYPE nombre PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

Si se escribe PUBLIC, en lugar de SYSTEM, y a continuación una URL se está indicando que la DTD es accesible vía web, porque es pública o se trata de una DTD estándar.

De forma genérica, la sintaxis sería:

```
<!DOCTYPE curso PUBLIC "fpi" "uri">
```

- curso: es el nombre del nodo raíz.
- fpi: es el Formal Public Identifier (identificador público formal), que es una cadena de texto que se utiliza para identificar un documento y forma parte de la declaración de tipo de documento.
- uri: es la ruta (path) absoluta o relativa hasta la DTD.

Si se combinan ambas opciones, DTD interna y externa la sintaxis es:

```
<!DOCTYPE curso PUBLIC "fpi" "uri" [... declaraciones ... ]>
```

La pregunta que se plantea ahora es: ¿cuándo utilizar una DTD interna o externa?

- Interna: si la DTD se utiliza para validar un único documento XML.
- Externa: si la misma DTD se utiliza para validar más de un documento XML.

También existe la posibilidad de utilizar DTD mixtas, internas y externas, pero que no abordamos en el módulo.

1.2. Definición de la sintaxis de documentos XML. Generalidades

En anteriores unidades se explicó que todo documento XML consta de:

Documento XML = datos + marcado

En este punto se enumeran, explican y/o repasan una serie de recomendaciones relativas a la sintaxis de un documento XML.

1. En un documento XML, las etiquetas describen su estructura.
2. Los elementos incluidos en el documento XML están organizados de forma jerárquica en estructura de árbol y, por lo tanto, permiten el anidamiento.
3. El orden en el que se incluyen los elementos en el documento XML es importante.
4. Los elementos tienen atributos, pero no pueden existir dos atributos con el mismo nombre.
5. Los atributos definen las propiedades de los elementos.
6. Los atributos son siempre hojas del árbol de contenidos, pero no todas las hojas son atributos, pueden ser elementos o atributos.

Para decidir cuándo un dato del documento XML es un elemento o un atributo se puede tener en cuenta lo siguiente:

Un dato es un elemento cuando cumpla alguna de las condiciones siguientes:

- El dato es parte esencial del documento XML.
- La información se expresa de forma estructurada, incluidas subestructuras.
- El valor del dato cambia con frecuencia y en función del valor de uno de sus atributos.

Un dato es un atributo cuando cumpla alguna de las condiciones siguientes:

- El dato es secundario en el documento XML.
- La información se expresa de forma abreviada.
- El dato es de pequeño tamaño y solo puede tener unos cuantos valores fijos.

2. Definiciones de tipo de documento o DTD

A continuación se detallan los cuatro tipos de declaraciones de las que disponemos para elaborar nuestro DTD: ELEMENT, ATTLIST, ENTITY y NOTATION.

2.1. Declaración de elementos terminales

Son elementos terminales de la DTD los que se corresponden con hojas (no contienen más hojas) de la estructura en árbol del documento XML asociado.

La sintaxis es: `<!ELEMENT nombre_elemento regla>`

La regla define el contenido del elemento.

Reglas:

EMPTY: el elemento es vacío, es decir, no tiene contenido.

```
<!ELEMENT elementoempty EMPTY>
```

En el documento XML al utilizar, este elemento lo podemos hacer de esta forma:

```
<elementoempty></elementoempty> o también <elementoempty/>
```

ANY: el elemento puede contener cualquier cosa (texto y otros elementos).

```
<!ELEMENT elementoany ANY>
```

En el documento XML, al utilizar este elemento, lo podemos hacer de esta forma:

```
<ejemplo>Esto es <a>un ejemplo</a></ejemplo>
```

(#PCDATA): el elemento solo puede contener texto. No permite anidación. #PCDATA debe escribirse entre paréntesis.

```
<!ELEMENT elementopdata (#PCDATA)>
```

En el documento XML, al utilizar este elemento, lo podemos hacer de esta forma:

```
<elementopdata>Este es el elemento PCDATA</elementopdata>
```

**EJEMPLO 1B**

Crear una DTD interna que se corresponda con la siguiente estructura de datos de un documento XML:

```
<libro> El libro está recogido en la colección <ubicacion/> con el
Título siguiente <titulo>Mi primer libro</titulo> sobre el autor
<autor/> no tenemos mucha información. En el libro <descripcion> cuento
<a>la experiencia</a>de escribir un libro</descripcion> Hasta la próxima
</libro>
```

Solución

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE libro [
  <!ELEMENT libro ANY>
  <!ELEMENT ubicacion EMPTY>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT autor EMPTY>
  <!ELEMENT descripcion ANY> ]>
```

2.2. Declaración de elementos no terminales

Son elementos no terminales del DTD los que se corresponden con ramas (que contienen más ramas u hojas) de la estructura en árbol del documento XML asociado.

, (coma): el elemento contiene los elementos en el orden indicado. El elemento A está formado por los elementos B y C, y necesariamente en ese orden:

```
<!DOCTYPE A [ <!ELEMENT A (B, C)>
  <!ELEMENT B EMPTY>
  <!ELEMENT C EMPTY> ]>
```

| (o lógico): el elemento A contiene uno de los dos elementos B o C.

```
<!DOCTYPE A [
  <!ELEMENT A (B|C)>
  <!ELEMENT B EMPTY>
  <!ELEMENT C EMPTY> ]>
```

? (interrogante): el elemento C puede aparecer o no, pero solo una vez. Es opcional.

```
<!DOCTYPE A [
  <!ELEMENT A (B, C?)>
  <!ELEMENT B EMPTY>
  <!ELEMENT C EMPTY> ]>
```

+ (suma): el elemento B ha de aparecer una o más veces (no puede no aparecer).

```
<!DOCTYPE A [
    <!ELEMENT A (B+, C)>
    <!ELEMENT B EMPTY>
    <!ELEMENT C EMPTY> ]>
```

***** (asterisco): significa que el elemento C puede aparecer cero, una o varias veces.

```
<!DOCTYPE A [
    <!ELEMENT A (B, C*)>
    <!ELEMENT B EMPTY>
    <!ELEMENT C EMPTY> ]>
```



EJEMPLO 2

Crear una DTD interna que se corresponda con la siguiente estructura de datos de un documento XML:

```
<zumos>
  <zumo>
    <codigo>8480017006912</codigo>
    <precio>2</precio>
  </zumo>
  <zumo>
    <precio>3</precio>
  </zumo>
</zumoss>
```

Solución

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE zumos [
    <!ELEMENT zumos (zumo)*>
    <!ELEMENT zumo (codigo?, precio)>
    <!ELEMENT codigo (#PCDATA)>
    <!ELEMENT precio (#PCDATA)> ]>
```

2.3. Declaración de atributos para los tipos de elementos

La declaración de atributos asociados a un elemento utilizan la siguiente sintaxis:

`<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorInicialAtributo >`
donde:

- `nombreElemento`: nombre del elemento que tiene asociado el atributo.
- `nombreAtributo`: nombre del atributo.
- `tipoAtributo`: tipo de dato.
- `valorInicialAtributo`: valor predeterminado del atributo (aunque también puede indicar otras cosas).

Se puede utilizar la misma declaración de atributos cuando el elemento tiene varios atributos asociados o utilizar una declaración diferente para cada uno de ellos.

```
<!ATTLIST nombreElemento
    nombreAtributo1 tipoAtributo1 valorInicialAtributo1
    nombreAtributo2 tipoAtributo2 valorInicialAtributo2 >
```

Los tipos de atributos más utilizados son los siguientes:

CDATA: el atributo es una cadena de texto.

```
<!DOCTYPE ejemplo [
    <!ELEMENT ejemplo EMPTY>
    <!ATTLIST ejemplo valor CDATA #REQUIRED> ]>
```

En el ejemplo de sintaxis se está diciendo que el elemento `ejemplo` tiene un atributo `valor` de tipo texto, y es obligatorio que aparezca.

ENUMERADO: el atributo es una lista de valores separados por barras verticales.

```
<!ATTLIST coche color (naranja | blanco| azul| verde) #REQUIRED>
```

ID: declara un atributo identificador del elemento que debe ser único y no se puede repetir en otros elementos o atributos.

```
<!DOCTYPE ejemplo [
    <!ELEMENT ejemplo (libro*)>
    <!ELEMENT libro (#PCDATA) >
    <!ATTLIST libro codigo ID #REQUIRED>    ]>
```

IDREF: permite hacer referencia a identificadores. El valor del atributo debe coincidir con el valor del atributo ID de otro elemento que existe en el documento.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE directorio [
<!ELEMENT directorio (persona)+ >
<!ELEMENT persona (#PCDATA) >
<!ATTLIST persona id ID #REQUIRED madre IDREF #IMPLIED padre IDREF #IMPLIED>
]>
```



```
<directorio>
  <persona id="p1">Pedro</persona>
  <persona id="p2">Victoria</persona>
  <persona id="p3" madre="p2" padre="p1">Carmen</persona>
</directorio>
```

En esta porción de código se está indicando que Carmen es la hija de Pedro y Victoria, ya que los atributos padre y madre de tipo IDREF contienen los ID de Pedro y Victoria.

NMTOKEN: indica que el atributo solo contiene letras, dígitos y los caracteres punto '.', guion '-', subrayado '_' y dos puntos ':'. El valor de este atributo no tiene por qué ser único, y por lo tanto puede aparecer varias veces en un documento XML.

Valores predeterminados de los atributos

Para un atributo puede haber cuatro valores predeterminados. Hay que tener en cuenta que los valores predeterminados son incompatibles entre sí y, por lo tanto, no se pueden combinar en la declaración de un atributo, ya que pueden entrar en contradicción.

Los valores predeterminados de los atributos son los siguientes:

"Valor del atributo": en este caso damos al atributo un valor por defecto; si no se especifica ningún otro valor, nuestro atributo queda inicializado con el valor por defecto que le hemos dado.

#REQUIRED: el atributo es obligatorio y no se especifica ningún valor predeterminado.

```
<!ATTLIST alumno edad CDATA #REQUIRED>
```

#IMPLIED: el atributo no es obligatorio y no se especifica ningún valor predeterminado.

```
<!ATTLIST variable tipo CDATA #IMPLIED>
```

El "tipo" de variable no es obligatorio.

#FIXED valor: el atributo tiene un valor fijo, donde "valor" indica que el atributo tiene un valor predeterminado y que no se puede cambiar.

```
<!ATTLIST articulo color CDATA #FIXED "naranja">
```

El atributo color del elemento articulo debe ser de color naranja.



EJEMPLO 3

Crear una DTD correspondiente a la siguiente estructura de datos de un documento XML:

```
<articulos>
  <articulo color="naranja">libreta</articulo>
  <articulo color="naranja">block</articulo>
  <articulo>cuaderno</articulo>
</articulos>
```

Solución

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE articulos [
  <!ELEMENT articulos (articulo)+>
  <!ELEMENT articulo (#PCDATA)>
  <!ATTLIST articulo color CDATA #FIXED "naranja">]>
```

2.4. Declaración de entidades

Cuando en un documento XML se quieren utilizar valores constantes, en la DTD correspondiente se utilizan entidades.

Una entidad es un nombre y un valor asignado (constante) a dicho nombre. En el documento XML se pueden utilizar estos valores simplemente referenciando el nombre de la entidad.

Cuando se procesa el documento XML se sustituyen las entidades por los valores correspondientes dados en la DTD.

Para definir una entidad en una DTD se usa el elemento `<!ENTITY>`. Se utiliza `SYSTEM` cuando las DTD son privadas. Solo se verán estas.

Las entidades se clasifican en:

Generales: pueden ser utilizadas desde el cuerpo del documento XML y desde la DTD. Se referencian `&nombreEntidad;`

Veámoslo con un ejemplo:

Si en un documento necesitamos hacer referencia a menudo a un enlace, como <https://www.w3c.es>, podemos crear una entidad cuyo identificador sea `w3c` y su valor sea la propia referencia en HTML (etiqueta `<a></a>`, con atributo `href`). La entidad quedaría definida de la forma siguiente:

```
<!ENTITY w3c"<a href='http://www.w3c.es'>w3c</a>">
```

Si necesitamos incluir muchas veces este enlace en un documento, en vez de escribir cada vez la estructura `<a></a>` se puede referenciar la entidad creada de la forma `&w3c;`. Cada vez que en el texto se escriba `&w3c;` se sustituirá por el link `<a href='http://www.w3c.es'>w3c</a>`.

Las entidades generales pueden ser:

- **Generales internas analizables** (significa que pueden ser parseadas). La sintaxis es:
`<!ENTITY nombreEntidad "valorEntidad">`
dentro del cuerpo del documento se referencia: `&nombreEntidad;`
- **Generales externas analizables**: establecen una relación entre el documento XML y otro documento a través de la URL de este último. Ambos archivos son analizados. La sintaxis es:
`<!ENTITY nombreEntidad SYSTEM "URI">`
- **Generales externas no analizables** (significa que no pueden ser analizadas)
`<!ENTITY nombreEntidad SYSTEM "URI" NDATA notación>`
Se utilizan cuando contienen un tipo de datos cualquiera no analizable, como puede ser una imagen. Con NDATA (Notation Data) se indica que la entidad no es analizable.

Paramétricas: solo pueden ser utilizadas en la DTD y permiten nombrar partes de la DTD y luego referenciarlas en la misma DTD. Se referencian de la forma: `%nombreEntidad;`

Veamos un ejemplo de entidad paramétrica externa. Supongamos que la DTD está en el archivo `alumno.dtd`.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE alumno SYSTEM "http://localhost/UD5/alumno.dtd">
<alumno>
    <nombre>Luis Martínez</nombre>
    <edad/>
    <ciudad>Segovia </ciudad>
</alumno>
```

Un posible contenido del archivo `alumno.dtd` sería:

```
<!ENTITY %dato "(#PCDATA)">
<!ELEMENT alumno (nombre, edad?, ciudad)>
<!ELEMENT nombre %dato;>
<!ELEMENT edad EMPTY>
<!ELEMENT ciudad %dato;>
```

2.5. Declaración de notaciones

Las notaciones se utilizan en la DTD cuando hay que indicar que el documento XML contiene entidades externas que no son XML. Por ejemplo, supongamos que el documento XML contiene una imagen. Esta entidad externa no la puede analizar el procesador XML, no la entiende. En este caso, de alguna forma hay que avisar de que esta entidad debe ser procesada por un programa que sí pueda hacerlo.

Las notaciones en una DTD pueden ser privadas y públicas. Solo veremos las privadas, y para ellas se utiliza SYSTEM. La sintaxis es:

```
<!NOTATION nombreNotación SYSTEM aplicacion>
```

Veamos un ejemplo de notación en una DTD privada. Si para cada grupo de alumnos se quiere incluir necesariamente un logo en formato PNG y que su valor sea una entidad, en el formato de dicha entidad hay que utilizar una notación, en este caso “png”. Vamos a suponer que en el sistema disponemos de una aplicación ejecutable capaz de procesar imágenes png.

La DTD correspondiente podría ser:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE alumnos [
<!NOTATION png SYSTEM "aplicaciones/aplicacionpng.exe">
<!ENTITY 1A SYSTEM "logo1A.png" NDATA png>
<!ENTITY 1B SYSTEM "logo1B.png" NDATA png>
<!ELEMENT alumnos (grupo)*>
<!ELEMENT grupo EMPTY>
<!ATTLIST grupo fotografia ENTITY #REQUIRED>    ]>
<alumnos>    <grupo fotografia="1A"/>
    <grupo fotografia="1B"/>
</alumnos>
```

En la [Documentación de trabajo y estudio](#) hay disponible el archivo [DTD_resumen.pdf](#) cuyo contenido es una recopilación de la sintaxis DTD.

3. XML Schema

Los orígenes del lenguaje XML Schema datan de 1998, cuando el W3C comienza con la elaboración de las recomendaciones. La primera versión oficial se publica en 2001, y la revisión última es de 2004. Como ya se ha dicho en la presentación, los grupos de trabajo del W3C relativos a este estándar están cerrados.

La recomendación del W3C comprende tres partes:

- XML Schema [Parte 0](#) Primer: es una primera aproximación al lenguaje, no normativa y que proporciona una gran cantidad de ejemplos y explicaciones detalladas.
- XML Schema [Parte 1](#) Structures: contiene la descripción de los componentes del lenguaje.
- XML Schema [Parte 2](#) Datatypes: contiene la definición de los tipos de datos incorporados en el lenguaje y sus restricciones.

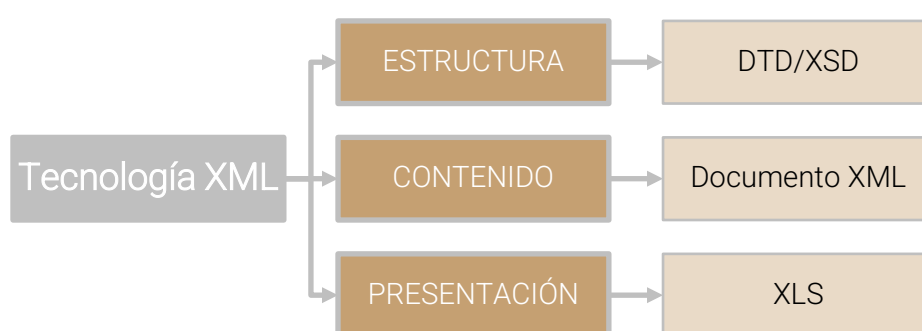


Figura 3.1. Esquema de la tecnología XML.

En apartados anteriores se ha explicado que el uso de las DTD permite diseñar vocabularios XML, pero su uso presenta algunos inconvenientes:

1. El lenguaje utilizado no es XML.
2. La DTD no es extensible, no se puede añadir vocabularios a una DTD definida.
3. No permite el uso de tipos de datos (entero, fecha, etcétera), solo cadenas de texto.
4. No permite el uso de espacios de nombres.

Por el contrario, utilizando esquemas XML:

1. La sintaxis es XML.
2. Permite trabajar con diferentes tipos de datos.
3. Permite trabajar con datos definidos por el usuario.
4. Permite aplicar restricciones a los tipos de datos.

Los esquemas XML son archivos en texto plano con extensión .xsd. Son los llamados documentos XSD (XML Schema Definition).

XSD (XML Schema Definition) es un lenguaje, normalmente conocido como XML Schema, utilizado para definir la estructura de un documento XML y así permitir su validación.

Por otra parte, para generar el esquema los elementos XML que se utilizan deben pertenecer al espacio de nombres XML Schema (<https://www.w3.org/2001/XMLSchema>).

Un esquema XML se compone de:

- **<xs:schema>** que contiene declaraciones para todos los elementos y atributos que puedan aparecer en el documento XML asociado y que sea válido. Sería el ejemplar.

La declaración de un esquema en un documento XML es la siguiente:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> ...
</xs:schema>
```

Se puede utilizar tanto "xs" como "xsd" como prefijos.

- **<xs:element>** serían los elementos hijos del ejemplar.

Veamos un ejemplo muy sencillo de esquema. Dado el siguiente documento XML, ¿cuál sería su esquema XML?

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE alumno>
<alumno edad="20">Alumno Apellido1 Apellido2</alumno>
```

A continuación se incluye su esquema XML correspondiente:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="alumno" type="xs:string"/>
</xs:schema>
```

En el código vemos la parte del ejemplar `<xs:schema ...>`, que en este caso solo incluye el espacio de nombres y la parte de los hijos del ejemplar, que en este caso solo incluye el "alumno" de tipo string. A continuación se verán los diferentes tipos de datos admitidos.

3.1. Tipos de datos

Los atributos y elementos declarados en un esquema XML deben pertenecer a un tipo de dato concreto, y este representa los posibles valores que puede tomar el atributo `type`.

En el [enlace](#) se detallan las definiciones de los tipos de datos recogidos en la parte 2 de la recomendación del W3C.

Los tipos de datos se dividen en:

- Tipos de datos simples o primitivos: son aquellos que no tienen elementos hijos ni atributos.
- Tipos de datos complejos: aquellos que tienen elementos hijos y/o atributos.

A. Tipos de datos simples

Los tipos de datos simples se pueden utilizar en los valores de los atributos y en los elementos que contienen solo datos de tipo carácter. Se nombran utilizando la sintaxis:

```
<xs:simpleType name="nombreElemento">
```

Existen 19 tipos de datos simples predefinidos primitivos, agrupados en cuatro categorías. Veamos los más significativos.

Categorías	Datos simples	Descripción
Tipos cadena		
	String	Cadena finita de caracteres.
	anyURI	Representa una URI estándar de Internet.
	ID, IDREF, ENTITY, NOTATION, MTOKEN	Declara enlaces a contenido externo no XML. Igual que las DTD.
Tipos binario codificado		
	boolean	Representa valores lógicos: true o false.
	hexBinary	Dato binario codificado como una serie de pares de dígitos hexadecimales.
Tipos numéricos		
	integer	Número entero positivo o negativo.
	positiveInteger	Número entero positivo.
	negativeInteger	Número entero negativo.
	decimal	Número decimal. Por ejemplo 6,25.
	float	Número de punto flotante de 32 bits de precisión simple formato IEEE 754.
	double	Número de punto flotante de 64 bits de doble precisión.
Tipos de fecha/hora		
	dateTime	Representa una fecha y hora absolutas según el calendario gregoriano, en formato "YYYYMM-DDThh:mm:ss".
	duration	Representa una duración de tiempo expresado en años, meses, días, horas, minutos y segundos. El formato utilizado es: PnYnMnDTnHnMnS. Por ejemplo, para representar una duración de 3 años, 6 meses, 5 días, 8 horas, 9 minutos y 10 segundos habría que escribir: P3Y6M5DT8H9M10S.
	time	Hora en el formato hh:mm:ss.
	date	Fecha específica del calendario gregoriano en formato YYYY-MM-DD.
	gYearMonth	Representa un mes de un año determinado del calendario gregoriano mediante el formato YYYY-MM.
	gYear	Representa un año gregoriano, con formato YYYY.

	gMonthDay	Representa un día de un mes del calendario gregoriano mediante el formato –MM-DD. Por ejemplo –05-01 representa el día 1 del mes de mayo.
	gDay	Indica el ordinal del día del mes del calendario gregoriano mediante el formato –DD, es decir el 8.º día del mes será –08.
	gMonth	Representa el mes del calendario gregoriano mediante el formato –MM. Por ejemplo, marzo es –03.

Tabla 1. Datos simples predefinidos primitivos.

Como ya se ha dicho, el XML Schema incluye el uso de espacios de nombres. Y también ya se ha explicado que la función de los espacios de nombres es poder definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento.

En los tipos de datos simples es posible utilizar lo que se llaman las facetas de limitación de tipos o restricciones de tipos, mediante el elemento `xs:restriction`. Estas facetas permiten indicar que un determinado valor debe estar dentro de un rango de valores, ser mayor o menor que otro valor, etcétera.

```
<xs:simpleType name="nombreElemento">
  <xs:restriction base="xs:string">
```



IMPORTANTE

Las facetas permiten definir restricciones sobre los posibles valores de atributos o elementos.

La tabla 2 describe las facetas más significativas:

Facetas en XSD	Descripción
length, minlength, maxlength	length: especifica una longitud fija. minlength: especifica una longitud mínima. maxlength: especifica una longitud máxima.
enumeration	Especifica una lista de valores admitidos.
whitespace	Indica el tratamiento de espacios en blanco, tabulaciones, saltos de línea y retornos de carro que puedan aparecer. Valores posibles: preserve/replace, collapse.
(max/min) (In/Ex)clusive	Límite superior/inferior del tipo de datos. Inclusive: el valor dado forma parte del conjunto de valores válidos. Exclusive: el valor dado no forma parte del conjunto de valores válidos.
totalDigits, fractionDigits	Número de dígitos totales y decimales de un número decimal.
pattern	Permite construir máscaras que han de cumplir los datos de un elemento. Ver Tabla 3.

Tabla 2. Facetas de limitación.

La tabla 3 describe el conjunto de caracteres para la generación de máscaras o patrones.

Patrón	Significado
[A-Z a-z]	Letra.
[A-Z]	Letra mayúscula.
[a-z]	Letra minúscula.
[0-9]	Dígitos decimales.
\D	Cualquier carácter excepto un dígito decimal.
(A)	Cadena que coincide con A.
A B	Cadena que es igual a la cadena A o a la B.
AB	Cadena que es la concatenación de las cadenas A y B.
A?	Cero o una vez la cadena A.
A+	Una o más veces la cadena A.
A*	Cero o más veces la cadena A.
[abcd]	Alguno de los caracteres que están entre corchetes.
[^abcd]	Cualquier carácter que no esté entre corchetes.
\t	Tabulación.

Tabla 3. Caracteres para la generación de patrones.

Veamos algunos ejemplos de algunas de las diferentes facetas.

1. xs:length

Crear un elemento de nombre “estado” de tipo cadena de texto con una longitud máxima de 12 caracteres y dos valores posibles (libre, ocupado).

```
<xs:simpleType name="estado">
  <xs:restriction base="xs:string">
    <xs:maxLength value="12"/>
    <xs:enumeration value="libre "/>
    <xs:enumeration value="ocupado"/>
  </xs:restriction>
</xs:simpleType>
```

2. xs:minExclusive y xs:maxInclusive

Definir un elemento de nombre “mes” con la restricción de que el valor que tome no pueda ser menor que 1 ni mayor que 12.

```
<xs:element name="mes" type="numeroMes"/>
<xs:simpleType name="numeroMes">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="12"/>
  </xs:restriction>
</xs:simpleType>
```

El tipo numeroMes definido de esta forma podría ser utilizado por otros elementos, ya que no está

contenido en el elemento “mes”.

3. **xs:enumeration**

Definir un elemento de nombre “semaforo” con la restricción de que los únicos valores admitidos son: “verde”, “naranja” y “rojo”.

```
<xs:element name="semaforo ">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="verde"/>
      <xs:enumeration value="naranja "/>
      <xs:enumeration value="rojo"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

4. **xs:whiteSpace**

Definir un elemento de nombre “dirección” con la restricción de que se han de preservar (preserve) los espacios en blanco, las tabulaciones, los saltos de línea y los retornos de carro que puedan aparecer en él.

```
<xs:element name="direccion">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

5. **xs:totalDigits**

Definir un elemento de nombre “calificaciones” de dos dígitos cuyo valor es un número entero comprendido entre 1 y 30, ambos inclusive.

```
<xs:simpleType name="calificaciones">
  <xs:restriction base="xs:integer">
    <xs:totalDigits value="2"/>
    <xs:minExclusive value="0"/>
    <xs:maxInclusive value="30"/>
  </xs:restriction>
</xs:simpleType>
```

6. **xs:pattern**

Definir un elemento de nombre “letra” con la restricción de que el único valor admitido es una letra mayúscula de la A a la Z:

```
<xs:element name="letra">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
```

`</xs:element>`

A. Tipos de datos compuestos

Los tipos de datos compuestos permiten definir estructuras de datos complejas. Están formados por las declaraciones de uno o más elementos y/o atributos o referencias a ellos. Se nombran utilizando la sintaxis:

```
<xs:complexType name="nombreElemento">
```

Podría ocurrir que el elemento complejo no tuviera elementos y tuviera al menos un atributo. En este caso, se dice que el elemento complejo es vacío.

También podría ocurrir que el elemento complejo fuera mixto, con uno o más elementos, y tener o no atributos.



EJEMPLO 4

```
<xs:element name="dado">
  <xs:complexType>
    <xs:attribute name="numero" type="numeroDado"/>
  </xs:complexType>
</xs:element>
<xs:simpleType name="numeroDado">
  <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="1"/>
    <xs:maxExclusive value="6"/>
  </xs:restriction>
</xs:simpleType>
```

Vemos que el elemento de tipo complejo llamado "dado" no contiene otros elementos, pero sí un atributo de nombre "numero".

3.2. Elementos del lenguaje

XML Schema requiere, al igual que la DTD, la declaración de los elementos que se utilizarán en el documento XML. En los ejemplos anteriores ya se ha utilizado la sintaxis adecuada para ello, pero es ahora cuando la introducimos.

La figura siguiente muestra la sintaxis utilizada en la declaración de los elementos.

```
< Element name="Nombre_Elemento" type="Tipo_Elemento" />
```

Figura 3.1. Sintaxis de los elementos.

```
<xs:element name="x" type="y"/>
```

Por lo tanto, en la declaración se necesita dar un nombre al elemento y decir de qué tipo es.

De los ejemplos anteriores: `<xs:element name="mes" type="numeroMes"/>`

Respecto de los atributos, en XML Schema estos proporcionan información adicional del elemento correspondiente. La figura siguiente muestra la sintaxis de los atributos.

```
< Attribute name="Nombre_Atributo" type="Tipo_Atributo" >
```

Figura 2.3. Sintaxis de los atributos.

```
<xs:attribute name="x" type="y"/>
```

Ejemplo de esquema correspondiente a un documento XML con el elemento de nombre "curso", y cada curso dispone del atributo grupo:

```
<curso grupo="B">2</curso>
```

La declaración del elemento y atributo sería:

```
<xs:element name="curso" type="xs:integer"/>
  <xs:attribute name="grupo" type="xs:string"/>
```

Hay que tener en cuenta que:

- Todos los atributos pueden tomar como valor un tipo simple.
- Cuando un elemento tiene al menos un atributo ese elemento se dice que es complejo.

El atributo use permite indicar si la aparición del atributo en el elemento con el que se ha asociado es requerido u opcional.

Veamos los tipos de declaración de atributos:

1. **fixed:** el valor del atributo es fijo.

```
<xs:attribute name="grupo" type="xs:string" fixed="A"/>
```

2. **default:** valor por defecto del atributo.

```
<xs:attribute name="grupo" type="xs:string" default="A"/>
```

3. **required:** el atributo es obligatorio.

```
<xs:attribute name="grupo" type="xs:string" use="required"/>
```

4. **optional:** el atributo es opcional. Equivalente a no indicar nada.

```
<xs:attribute name="grupo" type="xs:string" use="optional"/>
```

El siguiente ejemplo nombra todos los tipos de declaración de atributos de forma resumida.

```
<xs:attribute
  name="nombre del elemento"
  type="tipo global de datos"
  ref="declaración del elemento global"
  form="cualificación" <!--qualified o unqualified -->
  id="identificador"
  default="valor por defecto"
  fixed="valor fijo" use="uso" <!-- prohibited, optional o required -->
```

3.3. Indicadores

Los indicadores permiten controlar cómo se van a utilizar los elementos en un documento XML. Hay siete tipos de indicadores, que se pueden agrupar en:

1. **Indicadores de orden:** secuencia (sequence), todo (all) y elección (choice).

`xs:sequence`, especifica el orden en el que obligatoriamente deben aparecer los elementos hijo de un elemento.

`xs:all`, indica que los elementos hijo pueden aparecer en cualquier orden y cada uno de ellos debe aparecer una sola vez.

`xs:choice`, indica que solo se puede escribir uno de los elementos hijo.



EJEMPLO 5

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido1" type="xs:string"/>
      <xs:element name="apellido2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2. **Indicadores de ocurrencia:** `maxOccurs` y `minOccurs`.

maxOccurs: establece el número máximo de veces que puede aparecer un elemento. Por defecto es 1. Para especificar un número ilimitado de veces se utiliza `unbounded`.

minOccurs: establece el número mínimo de veces que puede aparecer un elemento. Por defecto es 1.



EJEMPLO 6

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombreCompleto" type="xs:string"/>
      <xs:element name="mote" type="xs:string"
        maxOccurs="5" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

3. **Indicadores de grupo:** si es de elementos (`group`) y si es de atributos (`attributeGroup`).

`xs:group`, agrupa un conjunto de declaraciones de elementos relacionados.

`xs:attributeGroup`, define grupos de atributos.



EJEMPLO 7

El siguiente ejemplo define un grupo de personas y luego puede referenciarse desde otro elemento.

```
<xs:group name="personaGrupo">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
    <xs:element name="aniversario" type="xs:date"/>
  </xs:sequence>
</xs:group>
<xs:element name="persona" type="personaInfo"/>
<xs:complexType name="personaInfo">
  <xs:sequence>
    <xs:group ref="personaGrupo"/>
    <xs:element name="pais" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

3.4. Asociación del esquema con el documento XML.

Para asociar el esquema creado con el documento XML correspondiente se referencia el archivo del esquema mediante su URI en la zona del ejemplar, junto al espacio de nombres. Debe ir precedido del prefijo "xsi:".

Por ejemplo, se tiene el archivo XML alumnos.xml y su esquema correspondiente alumnos.xsd. La forma de asociarlos dentro del archivo XML sería la siguiente:

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

<alumnos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://localhost/UD5/alumnos.xsd">
```

3.5. Documentacion del esquema

Se ha estudiado cómo crear un esquema y cómo asociarlo al documento XML que lo requiere. Pero en ningún momento se ha hablado de cómo documentar dicho esquema.

Es importante documentar los programas que se elaboran para facilitar la tarea de reconocimiento posterior de los procesos realizados. ¿Qué información puede ser de utilidad? Por ejemplo, el autor, si tiene derechos de autor, qué representa dicho esquema, etcétera.

En XML Schema se pueden utilizar los comentarios del lenguaje, pero no es conveniente porque no se garantiza que no se puedan perder durante alguna transformación que pueda sufrir el archivo correspondiente.

Para ello, XML Schema dispone de un elemento `xs:annotation` que permite guardar información adicional. Este elemento, a su vez, puede contener una combinación de otros dos, que son:

- `xs:documentation`: puede contener elementos de esquema y elementos XML bien estructurados.

También permite determinar el idioma del documento mediante el atributo `xml:lang`.

```
<xs:annotation>
  <xs:documentation xml:lang="es-es">
    <modulo>Lenguajes de marcas y sistemas de gestión de
      información.</modulo>
    <fecha_creación> 2021</fecha_creacion>
  </xs:documentation>
</xs:annotation>
```

- `xs:appinfo`: guarda información para los programas de software.

También se utiliza para generar ayuda contextual para cada elemento declarado en el esquema.

```
<xs:annotation>
  <xs:appinfo>
    <texto_ayuda>Introducir el nombre completo del
tema</texto_ayuda>
  </xs:appinfo>
</xs:annotation>
```

3.6. Validacion

La validación de los esquemas creados es muy importante. De ella dependerá el correcto funcionamiento de los documentos XML.

Existen validadores online que funcionan de la forma “copiar y pegar” para luego chequear. Entre ellos están:

<https://www.freeformatter.com/xml-validator-xsd.html>

<https://www.liquid-technologies.com/online-xsd-validator>

<https://www.utilities-online.info/xsdvalidation>

Algunos de ellos dan la opción de validar por separado el archivo XML y el archivo XSD. Cuando ambos son correctos se pueden validar conjuntamente para comprobar la correspondencia entre ellos.

4. Definición de documentos JSON

El formato JSON fue definido por Douglas Crockford en 2002 y más tarde se publicó como norma tanto por parte de la [IETF](#) como del [ECMA](#). Concretamente, está definido en el Standard ECMA-404

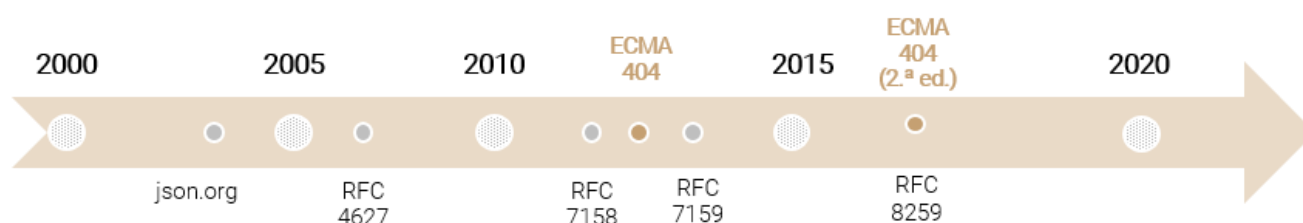


Figura 4.1. Evolución de JSON.

Según la definición de JSON anterior, se trata de un formato ligero para el intercambio de datos. Y aquí es donde entra el concepto de serialización.

Cuando se quiere enviar alguna estructura de datos a través de las redes hay que realizar un proceso previo por el cual dichos datos pasan a tener un formato especial para leerlos y almacenarlos (como un flujo de bytes); por ejemplo, leer datos de una base de datos y enviarlos a través de Internet. Este proceso se llama serializar, y es muy utilizado cuando se interactúa con servicios web.



Figura 4.2. Logotipo de JSON.

Por lo tanto, JSON permite la serialización (convertir un objeto en una cadena) y también la deserialización, que es la operación inversa (convertir cadena en objeto).

Veamos cuáles son las características más importantes de JSON.

- JSON es un formato de datos.
- El tamaño de los archivos que se transfieren es ligero.
- Muchos de los lenguajes de programación disponen de funciones para interpretar cadenas JSON y convertir datos en cadenas JSON válidas.
- Los archivos JSON son archivos de texto plano con codificación UTF-8, que es compatible con todos los sistemas.

Las tablas siguientes muestran las ventajas y desventajas del uso de JSON y XML.

Formatos	Ventajas	Desventajas
JSON	Es autodescriptivo y fácil de entender.	La notación puede ser algo confusa.
	Es una alternativa a XML.	No es extensible.
	Es más rápido y ligero en cualquier navegador.	No soporta grandes cargas, solo datos comunes.
	Es más fácil de leer que XML.	Para la seguridad requiere de mecanismos externos como expresiones regulares.
	Se analiza (<i>parsea</i>) más rápido.	
	Velocidad de procesamiento alta.	
	Puede ser entendido de forma nativa por los analizadores de JavaScript.	
XML	Tiene un formato estructurado y fácil de comprender.	El formato es sumamente estricto.
	Separa radicalmente la información o el contenido de su presentación o formato.	Lleva más tiempo procesarlo.
	Está diseñado para ser utilizado en cualquier lenguaje o alfabeto.	Complejidad de analizador (<i>parser</i>).
	Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.	Un error en cualquier parte del formato puede hacer que todo el documento sea inválido.
	Tiene soporte para cualquier tipo de datos.	
	Se pueden definir estructuras complejas y reutilizables.	

Tabla 1. Comparativa JSON / XML.

4.1. Sintaxis de JSON

JSON tiene dos tipos de elementos:

- matrices** (arrays): son listas de valores separados por comas y se escriben entre corchetes [].

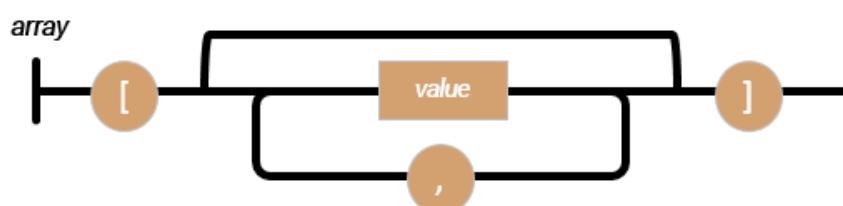


Figura 4.3. Sintaxis de la matriz.

- **objetos** (objects): son listas de parejas nombre:valor. Las parejas están separadas por comas. Los objetos se escriben entre llaves {}, y los nombres de las parejas se escriben siempre entre comillas dobles.

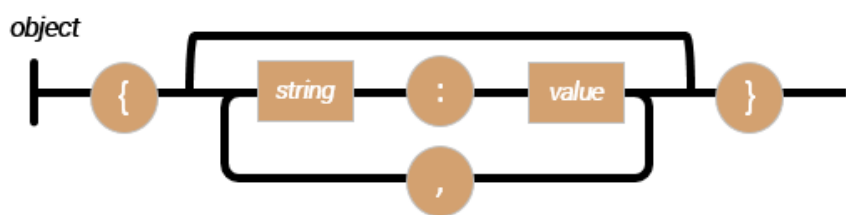


Figura 4.4. Sintaxis del objeto.



EJEMPLO 8

El código siguiente es un archivo JSON:

```
{ "alumnos": [{
    "nombre": "Nombre1 Apellido1",
    "edad": "20",
    "sexo": "m" },
  {
    "nombre": "Nombre2 Apellido2",
    "edad": "22",
    "sexo": "f" } ]
}
```

El archivo contiene un array “alumnos”, tiene dos valores y cada valor tiene tres atributos con su valor.

El mismo archivo en formato XML sería:

```
<?xml version="1.0" encoding="utf-8"?>
<alumnos>
  <alumno>
    <nombre>Nombre1 Apellido1</nombre>
    <edad>20/edad>
    <sexo>m</sexo>
  </alumno>
  <alumno>
    <nombre>Nombre2 Apellido2</nombre>
    <edad>22</edad>
    <sexo>f</sexo>
  </alumno>
</alumnos>
```

Otras reglas de JSON son las siguientes:

- Un documento JSON está formado por un único elemento (un objeto o una matriz).
- Tanto en los objetos como en las matrices, el último elemento no puede ir seguido de una coma.
- Los espacios en blanco y los saltos de línea no son significativos. No importa el número de espacios en blanco o saltos de línea que pueda haber separando cualquier elemento o símbolo del documento.
- Los ficheros JSON no pueden contener comentarios.

Los tipos de valores para los objetos o las matrices pueden ser:

- **Números:**

Pueden ser enteros, decimales o en notación exponencial.

El separador decimal es el punto.

Un número negativo empieza por el signo menos (-).

El indicador de la notación exponencial es e o E.

Los números positivos no pueden empezar por el signo +.

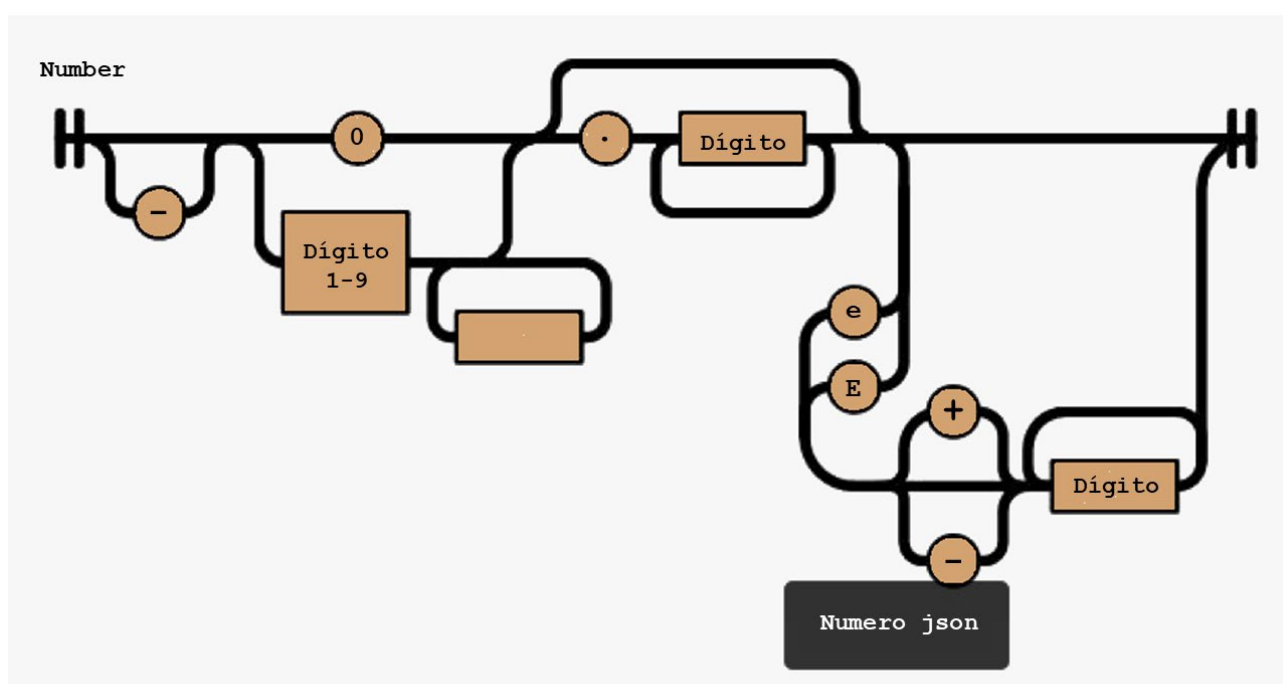


Figura 4.5. Sintaxis de number.

- **Cadenas (strings):**

Una cadena es una secuencia establecida de cero o más caracteres Unicode.

Las cadenas se escriben entre comillas dobles ("").

Los caracteres especiales y los valores Unicode se escriben con una \contrabarra.

Los caracteres que deben escribirse siempre como caracteres especiales son \" (comillas), \\ (contrabarra), \b (retroceso), \f (salto de página), \n (salto de línea), \r (retorno de carro), \t

(tabulador) y los caracteres unicode (\u...).

El carácter barra (/) puede escribirse como carácter / o como carácter especial \ (suele ser necesario cuando el contenido es código HTML y la barra indica un cierre de etiquetas).

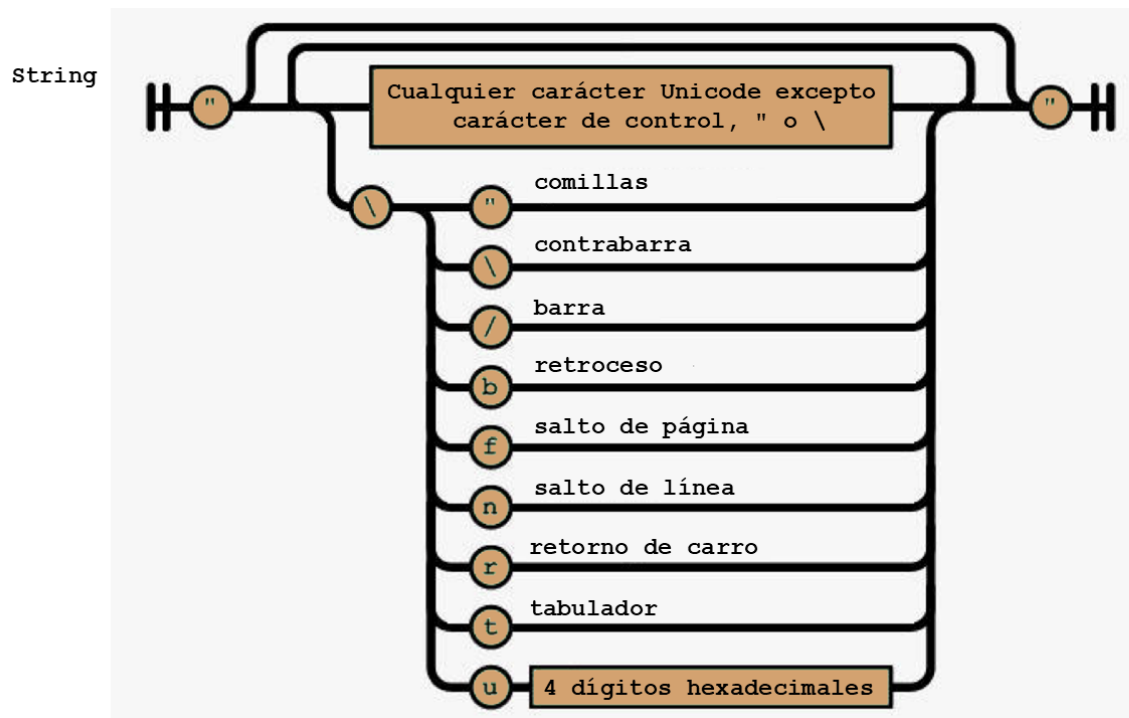


Figura 4.6. Sintaxis de una string.

- Valores true, false y null.

Se escriben sin comillas.

- Objetos y matrices.

Puede haber objetos y matrices dentro de objetos y de matrices, sin límite de anidamiento.

4.2. Ejemplos de uso de JSON



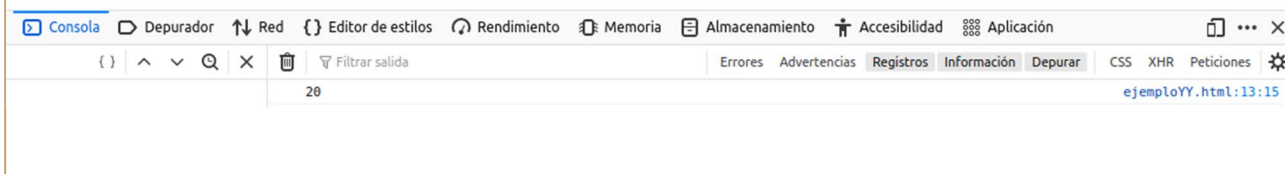
EJEMPLO 9

El ejemplo solo implica JavaScript y un navegador. En el código HTML siguiente se crea un objeto JSON, y a través de la consola se accede a uno de sus atributos.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo 9 uso de JSON</title>
  </head>
  <body>
    <script>
      const Nombre1 = {
        "edad" : "20",
        "nombre" : "Nombre1 Apellido1",
        "sexo" : "m"  };
      console.log( Nombre1.edad );
    </script>
  </body>
</html>
```

En este ejemplo se declara una constante (ya que no se van a modificar los datos) llamada Nombre1, y se definen tres atributos dentro de llaves {}.

La salida generada en consola muestra la edad.



En el caso de utilizar el código del ejemplo 10 completo, al tener el array alumnos dos valores se puede referenciar cualquiera de ellos o de sus atributos.



EJEMPLO 10

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo 10 JSON</title>
  </head>
  <body>
    <script>
      const alumnos =
        [{
          "edad": "20",
          "nombre": "Nombre1 Apellido1",
          "sexo": "m"
        }, {
          "edad": "22",
          "nombre": "Nombre2 Apellido2",
          "sexo": "f"
        }
      ];
      //Imprime el atributo nombre del segundo elemento
      console.log( alumnos[1].nombre );
    </script>
  </body>
</html>
```

En la consola del navegador muestra Nombre2 Apellido2.

Vemos que el array lo referenciamos utilizando el nombre de la constante (o variable) declarada y los [], sabiendo que la posición 1 comienza en el 0. A continuación se indica el nombre del atributo a visualizar.



EJEMPLO 11

Archivo ejemplo_11.html

En este código JSON tiene 3 atributos profesor, alumno1 y alumno2. Cada uno de ellos, a su vez, contiene una lista de atributos que no tienen por qué ser idénticos en todos ellos.

La consola del navegador muestra el nombre del profesor y el teléfono fijo del alumno1.



IMPORTANTE

Es importante que antes de probar el código JSON se valide. Se puede utilizar, por ejemplo, la web <https://jsonlint.com>, que permite copiar y pegar nuestro código y validarlo.

4.3. Utilidades de JSON

El lenguaje XML para la transferencia de datos utiliza el protocolo de mensajería SOAP (Simple Object Access Protocol). El formato de intercambio de datos JSON, sin embargo, utiliza la tecnología REST (Representational State Transfer), que es más sencilla y versátil porque soporta tanto JSON como XML y HTML. El éxito de JSON por su sencillez ha potenciado el uso de REST.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

REST es el estándar más lógico, eficiente y habitual en la creación de API para servicios de Internet.

En la actualidad, los desarrollos de software que se comunican con un navegador web o una aplicación móvil suelen utilizar JSON como formato de datos.

Se puede validar y analizar sintácticamente JSON de forma online o utilizando ciertos métodos desde los lenguajes de programación.

Existen varios validadores/analizadores online. Se ha mencionado uno anteriormente (<https://jsonlint.com>), y ahora nombramos otro: <http://json.parser.online.fr>.

Desde JavaScript está disponible un analizador nativo, el método `JSON.parse(cadena)`. Desde otros lenguajes de programación se pueden utilizar librerías específicas disponibles en este [enlace](#).

El método `JSON.parse(cadena)` permite convertir una cadena en formato JSON en una variable JavaScript que contiene el valor de esa cadena.

```
var alumno = JSON.parse('{"nombre ": "Nombre1 ", "edad ": 20, "sexo ": m }');
```

Otro método importante es `JSON.stringify(objeto)`, cuya finalidad es convertir un objeto, array o valor en su representación como cadena en formato JSON.

```
var alumno = {
  nombre : "Nombre1",
  edad: "20",
  sexo : 'm'}
var cadena = JSON.stringify(alumno);
```


Las siguientes son utilidades en línea que permiten manipular o examinar directamente datos codificados en JSON:

`JSON Lint`: analiza y valida código JSON. Disponible en este [enlace](#).

`JSON Viewer online`: es un editor que permite visualizar el árbol con todos los datos del archivo JSON y resaltar los diferentes tipos de datos. Para ello utiliza colores que se pueden personalizar. También permite la edición de los datos. Disponible en este [enlace](#).

`onlineJSONtools`: es un conjunto de utilidades JSON para trabajar con estructuras de datos JSON. Todas ellas son gratuitas y sencillas de usar. Su funcionamiento consiste simplemente en cargar JSON y obtener los resultados. Disponible en este [enlace](#).

`CSV JSON`: se trata de un conversor de formato de JSON a CSV. Para la conversión de CSV a otros formatos (PDF, SQL, XML, YAML, etcétera) están las herramientas proporcionadas por [Convertcsv.com](#). CSV JSON está disponible en este [enlace](#).

`JSON-generator`: permite generar código basado en una plantilla JSON predefinida. La salida generada JSON se puede guardar como un archivo .json. Disponible en este [enlace](#).

5. Introducción a YAML

YAML es un lenguaje de marcas ligero que se caracteriza por su legibilidad y facilidad para escribirlo. Comparativamente es más legible que JSON y bastante más que XML, pero no los sustituye completamente.

¿Por qué, entonces, utilizar YAML en lugar de JSON o XML? Por razones básicas de facilidad:

- El formato utilizado es más amigable.
- Se entiende muy fácilmente.
- Facilita el mapeo de estructuras de datos complejas.

Como resumen, podemos decir que YAML es un:

- Lenguaje de marcado ligero: aunque esta no es realmente su fortaleza.
- Formato de intercambio de datos: esta acepción es más acertada, ya que su uso más generalizado es este.
- Formato de serialización de datos: esta acepción es la más acertada. Y, aunque esta definición es similar a la anterior, existe un punto de diferencia, y es su alta integración con JavaScript, PHP o Python, entre otros lenguajes de programación, además de su legibilidad.

5.1. Reglas sintácticas de YAML

- Los archivos YAML deben estar codificados en UTF-8 o UTF-16.
- Los archivos YAML pueden tener la extensión .yaml o .yml.
- El comienzo del archivo se marca con tres guiones: ---
- La sangría de espacios se utiliza para marcar la estructura de los datos. Utiliza la indentación para marcar la jerarquía. La indentación debe hacerse siempre con espacios en blanco. La tabulación está prohibida, daría error y no funcionaría.
- Los datos escalares se representan como pares `nombre: valor`, separando el nombre del valor con dos puntos y un espacio en blanco (:). Ejemplo, `usuario: 'username'`.
- Los valores en cada par `nombre: valor` pueden encerrarse entre comillas simples (') o dobles ("), o sin entrecomillar. Pero cuando el valor incluya espacios en blanco, o ciertos caracteres especiales, deben entrecomillarse. Y si el carácter son las propias comillas estas deben escaparse con (\).
- Los comentarios en YAML comienzan con el carácter '#'.
- Los arrays se componen de elementos que son pares `nombre: valor`. Estos elementos se pueden declarar uno debajo de otro o en una línea, entre llaves '{ }', precediendo la lista de un guion y un espacio '-' y separando cada par del siguiente por una coma y un espacio en blanco ', '. Ejemplo:

```
evento_0:
```

```
- {evento: "Examen", fecha: "17/06/2021", hora: "18: 00", lugar:
"Aula 08", asistencia: true}
```

La siguiente tabla muestra un ejemplo de archivo YAML/JSON.

YAML	JSON
<pre>- Alumnos: - Alumno: numExpediente: 1 nombre: Nombre1 apellido: Apellido1 - Alumno: numExpediente: 2 nombre: Nombre2 apellido: Apellido2</pre>	<pre>{ "Alumnos": [{ "Alumno": { "numExpediente": 1, "nombre": "Nombre1", "apellido": "Apellido1" } }, { "Alumno": { "numExpediente": 2, "nombre": "Nombre2", "apellido": "Apellido2" } }] }</pre>

Tabla 1. Código comparativo YAML/JSON.

Para la creación y edición de archivos YAML se suele utilizar Sublime Text Editor, que está disponible en diferentes sistemas operativos (Windows, GNU/Linux, Mac OS) y que proporciona facilidades a los desarrolladores. La descarga es gratuita, pero no es software libre (Open Source).

También se puede utilizar Visual Studio Code.

5.2. Tipos de datos en YAML

Números: se expresan directamente sin comillas. Ejemplos:

- Entero 12
- Octal 012
- Hexadecimal 0x1af
- Real 3.14
- Exponencial 3.14e+10

Nulo: el valor nulo se puede indicar utilizando null o ~.

Booleano: valores true y false.

Fechas: se sigue la norma ISO-8601. Ejemplo: 2021-06-15T09:43:00. Se puede utilizar también el formato corto: 2021-06-15.

Listas: es un conjunto de elementos, y cada elemento se representa por un guion seguido de un espacio. Por ejemplo, la lista de los módulos formativos del ciclo sería:

- entornos
- sistemas
- procesos

Los miembros de las listas también se pueden agrupar entre corchetes ([y]), separando cada uno del siguiente por una coma y un espacio en blanco (,). La lista anterior quedaría así:

```
[entornos , sistemas , procesos]
```

Arrays: es un conjunto de elementos clave: valor. Debe haber un espacio en blanco entre ':' y el valor correspondiente. La lista anterior pasaría a ser:

- entornos:
 - nombre: entornos
 - ciclo : dam
 - curso : primero
- sistemas:
 - nombre: sistemas
 - ciclo : dam
 - curso : primero
- procesos:
 - nombre: procesos
 - ciclo : dam
 - curso : segundo

Este array se podría escribir de la forma:

- entornos : {nombre: entornos , ciclo: dam , curso: primero}
- sistemas : {nombre: sistemas , ciclo: dam , curso: primero}
- procesos : {nombre: procesos , ciclo: dam , curso: segundo}

Las listas y los arrays se pueden anidar. Por ejemplo:

```
modulos:
  primero:
    entornos
    sistemas
    programación
```

Para no tener que repetir los nombres de elementos se puede utilizar el ampersand (&) como marca o ancla, y luego se referencian con asterisco (*). De esta forma se evitan las duplicaciones.

- anclajes:
- presentación: &p Presentación
 - formacion: &f Formación

luego se referencian de la forma:

*p

*f

Otras características de YAML son las siguientes:

- Un documento YAML puede estar compuesto por varios documentos que se separan entre ellos por tres guiones seguidos (---) y terminan cada uno con tres puntos (...).
- Un documento YAML se puede iniciar con directivas, que son marcas que no forman parte del flujo de datos serializados pero que pueden usarse con fines informativos para la API encargada de leer el YAML. Las directivas se inicializan con el signo de porcentaje (%), y pueden ser de dos tipos:
 - %YAML x.x para indicar la especificación de YAML (versión) que estamos empleando.
 - %TAG se utiliza como prefijo para las URL incluidas en el documento.

5.3. Validar YAML

Para validar los archivos de YAML se pueden utilizar online estos enlaces:

<http://www.yamllint.com>

<https://codebeautify.org/yaml-validator>

<https://jsonformatter.org/yaml-validator>

<https://onlineyamltools.com/validate-yaml>

Como siempre, insistir en la necesidad de validar los archivos YAML para comprobar que son correctos.

En la sección **Descarga de documentos** está disponible el archivo **YAML_resumen.pdf**, cuyo contenido es una recopilación de la sintaxis de YAML.



Unidad 6.

Conversión y adaptación de documentos para el intercambio de información



1. Técnicas de transformación de documentos XML

En unidades anteriores se ha explicado cómo la definición de la DTD proporciona estructura a un documento XML. Y de esa forma, al conocer la estructura del documento, ya es posible diseñar aquellas transformaciones que le son aplicables.

Cada posible presentación requiere la ejecución de una secuencia de instrucciones para extraer información del documento XML y que se guardará como archivo XSL (Extensible Stylesheet Language).

Las tecnologías a utilizar en la transformación de documentos XML (son [recomendaciones](#) de la W3C) son las siguientes:

- **XSLT** (XSL Transformations): permite definir el modo de transformar un documento XML en otro. Establece las reglas a aplicar sobre la selección de nodos que hace XPath en el documento. Las recomendaciones están disponibles en este [enlace](#).
- **XSL-FO** (XSL Formatting Objects): permite transformar XML en un formato legible e imprimible por el usuario, por ejemplo en un documento PDF. Son vocabularios especializados. Las recomendaciones están disponibles en este [enlace](#).
- **XPath** (XML Path Language): permite el acceso a los diferentes componentes de un documento XML. Hace la selección de nodos a los que se refieren las acciones. Las recomendaciones están disponibles en este [enlace](#).

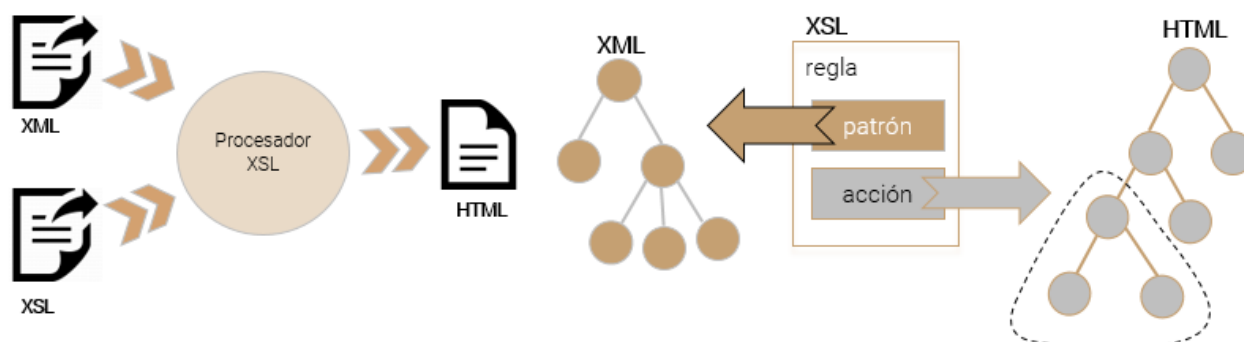


Figura 1.1. Proceso de transformación XML.

En la actualidad, el término utilizado de forma general para las transformaciones, y que engloba todas estas tecnologías, es XSL, y también se conocen con el nombre de hojas de estilo para XML.

XSL (eXtensible Stylesheet Language, lenguaje de hojas de estilo extensible) es un lenguaje o vocabulario de XML utilizado para transformar documentos XML.

XSL se utiliza para transformar o reexpresar el contenido de un documento en un formato conocido. Funciona a base de hojas de transformaciones (archivos .xsl), cuyo objetivo es seleccionar con qué partes del documento original se va a trabajar, componerlo con partes de otros (si procede) y con qué formato se va a presentar.

La transformación más utilizada con XSL es pasar de documento XML a documento HTML.

Estructura de una hoja de estilo XSL:

- Una hoja de estilos XSL es un documento XML con extensión .xsl, en el que se definen las transformaciones a realizar en el documento mediante una serie de elementos y atributos.
- La hoja de estilo incluye el elemento `stylesheet`, que declara el espacio de nombres para poder utilizar los elementos y atributos XSL.
- Para utilizar una hoja de estilos XSL en un documento se utiliza la instrucción de procesamiento `xml-stylesheet`.
- El elemento utilizado en la hoja de estilos para realizar las transformaciones es la plantilla.

XSL describe estas transformaciones como una secuencia de instrucciones, como si se tratara del código fuente de un programa, pero con sintaxis XML.

En estas instrucciones XSL, y para diferenciar del resto del documento, se añade siempre el prefijo `xsl:` a las etiquetas de este vocabulario. De esta forma queda delimitado el espacio de nombres y se evitan conflictos que pudieran aparecer con etiquetas de igual nombre que formen parte del resultado de la transformación.

Los documentos XML con los que se debe trabajar han de estar bien formados (siguen las reglas de la sintaxis XML) y han de ser válidos (cumplen con la definición de una gramática).

No hay que confundir las hojas de estilo CSS con las hojas de estilo XSL. El W3C creó una generalización de las hojas de estilo CSS que denominó XSL, y que permite dar formato al documento resultante de la transformación y aplicar determinadas plantillas.

Con XSL se genera un documento HTML a partir de un documento XML. Son dos documentos distintos. Con CSS, el navegador web recibe un documento XML y lo formatea utilizando las reglas CSS para presentarlo en pantalla, de forma que sea más fácilmente legible. Pero es el mismo documento.

Como resumen, diremos que XSL es el conjunto de tecnologías que conforman una hoja de estilo. Dentro de estas tecnologías destacan XPath para realizar la selección de los nodos y XSLT para realizar las transformaciones a dichos nodos.

2. XPath. Estructura y sintaxis

Xpath aparece en 1999, la última versión data de 2017 y las recomendaciones del W3C están disponibles en este [enlace](#).

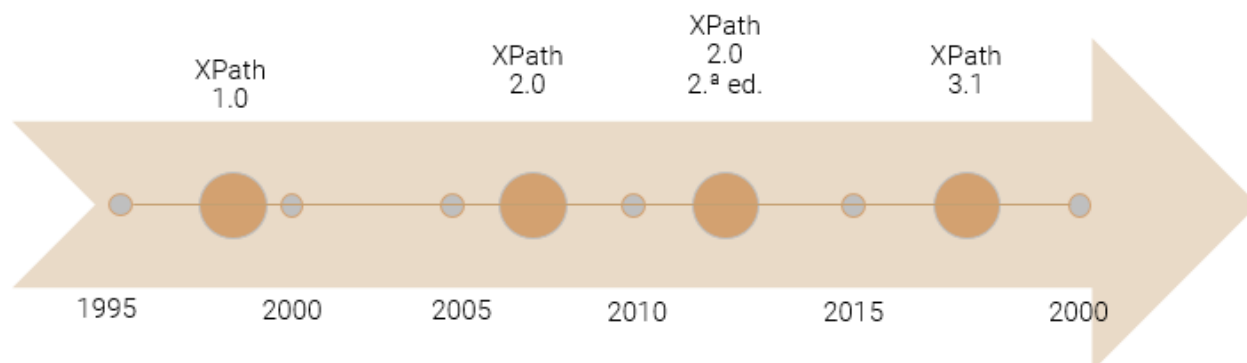


Figura 2.1. Línea de tiempo de las versiones de XPath.

El mecanismo de funcionamiento de XPath se basa en la interpretación del documento XML como una secuencia de elementos ordenados bajo la estructura de árbol. Esta estructura en árbol de los datos sería equivalente a la del modelo de objetos del documento (DOM), ya estudiado.

Para localizar los elementos en el documento XML, XPath utiliza el sistema de rutas de localización de Unix/GNU Linux, cuyos elementos son los nodos, los ejes, las pruebas de nodo y los predicados. Todos ellos se estudian a continuación.

Esta tecnología suele estar integrada junto con otras herramientas para el procesamiento de documentos XML. En concreto y globalmente están:

- XPath: para la navegación dentro de documentos XML.
- XQuery: para hacer consultas a documentos XML. XPath localiza los elementos y Xquery extrae información de ellos. Se estudia en la próxima unidad.
- XSLT: para la transformación de los datos de documentos XML.

Como resumen, se puede decir que XPath es un lenguaje que permite:

- Transformar el formato de los datos de un documento XML.
- Definir un vocabulario que ha de cumplir un documento XML.
- Acceder a los datos de un documento XML.

XPath trabaja con expresiones que son como condiciones (predicados) que se aplican sobre una estructura de árbol correspondiente a la jerarquía de datos de un documento XML, y devuelve todo lo que coincide (match) con ese predicado. Estas expresiones se pueden asimilar, en cierta medida, a las expresiones regulares.

2.1. Modelo de datos de XPath

XPath representa la estructura jerárquica de los datos del documento XML como un árbol dirigido, cuyos elementos son nodos y un elemento es padre de los elementos que contiene.

Hay dos nodos especialmente importantes, que son:

- **nodo contexto:** es el nodo del árbol que representa el documento XML y que se toma como punto partida de la expresión.
- **nodo actual:** es el nodo que se obtiene al final de una expresión, o, si son varios nodos, cada uno de los nodos de la secuencia devuelta por la expresión.

Hay siete tipos de nodos:

1. Nodo raíz o nodo documento

El nodo raíz es la raíz del árbol y contiene el ejemplar del documento XML. Se identifica por '/'.

El nodo raíz puede tener como hijos nodos elemento, nodos de instrucción de procesamiento y comentario, correspondientes a las instrucciones de procesamiento y comentarios que aparezcan en el prólogo.

El valor de cadena del nodo raíz es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo raíz, en orden, del documento.

2. Nodos elemento

Hay un nodo elemento por cada elemento del documento XML, a excepción de la declaración XML y la definición del tipo de documento.

Todos los nodos elemento tienen un nodo padre, que es el nodo raíz del documento.

Los nodos elemento pueden tener un identificador único (ID). Este es el valor del atributo que se declararía en la DTD como de tipo ID. No puede haber dos elementos en un documento con el mismo ID.

3. Nodos texto

Son nodos texto los caracteres del documento XML que no están marcados con ninguna etiqueta. En cada nodo texto se agrupan todos los datos de carácter que sea posible.

Los nodos texto no pueden tener hijos.

4. Nodos atributo

Cada nodo elemento puede tener asociado un conjunto de nodos atributo. El nodo elemento es el padre de cada uno de esos nodos atributo, pero los nodos atributo no son hijos de su elemento padre. Solo se consideran etiquetas añadidas al nodo elemento.

Los nodos atributo no pueden tener hijos.

Cada uno de estos nodos consta de un identificador y de un valor asignado.

5. Nodos espacio de nombres

Los nombres de elementos y atributos de un documento XML validado tienen asignado un espacio de nombres. Esta asignación viene dada en la DTD al principio del documento.

Un nodo elemento es padre de un nodo espacio de nombres, pero no a la inversa. Es decir, un nodo espacio de nombre no es hijo de su nodo padre. Los elementos tienen nodos espacio de nombres con estas condiciones:

- Tienen uno por cada atributo que empiece por xmlns:.
- Para cada atributo situado sobre un elemento previo cuyo nombre empiece por xmlns:.

6. Nodos instrucción de procesamiento

Las instrucciones de procesamiento de los documentos XML se encuentran fuera del árbol del documento y se denominan nodos de instrucción de procesamiento.

El nodo de instrucción de procesamiento comienza por <? y termina con ?>.

7. Nodos comentario

En un documento XML, XPath procesa el contenido marcado como comentario como un nodo de comentario.

El valor de cadena de los comentarios es el contenido del comentario sin incluir el inicio <!-- ni la terminación -->.



IMPORTANTE

La declaración DOCTYPE no se considera como nodo.

EJEMPLO 1

Tenemos el documento XML siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <titulo>Transbordo en Moscú</titulo>
    <autor>Eduardo Mendoza</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pequeña historia de la Transición</titulo>
    <autor fechaNacimiento="11/11/1948">Victoria Prego</autor>
    <fechaPublicacion año="2021"/>
  </libro>
  <libro>
    <titulo>DESPUES</titulo>
    <autor fechaNacimiento="21/09/1947">Stephen King</autor>
    <fechaPublicacion año="2021"/>
  </libro>
</biblioteca>
```

La figura siguiente muestra el esquema en árbol con todos los nodos y sus tipos que se generarían.

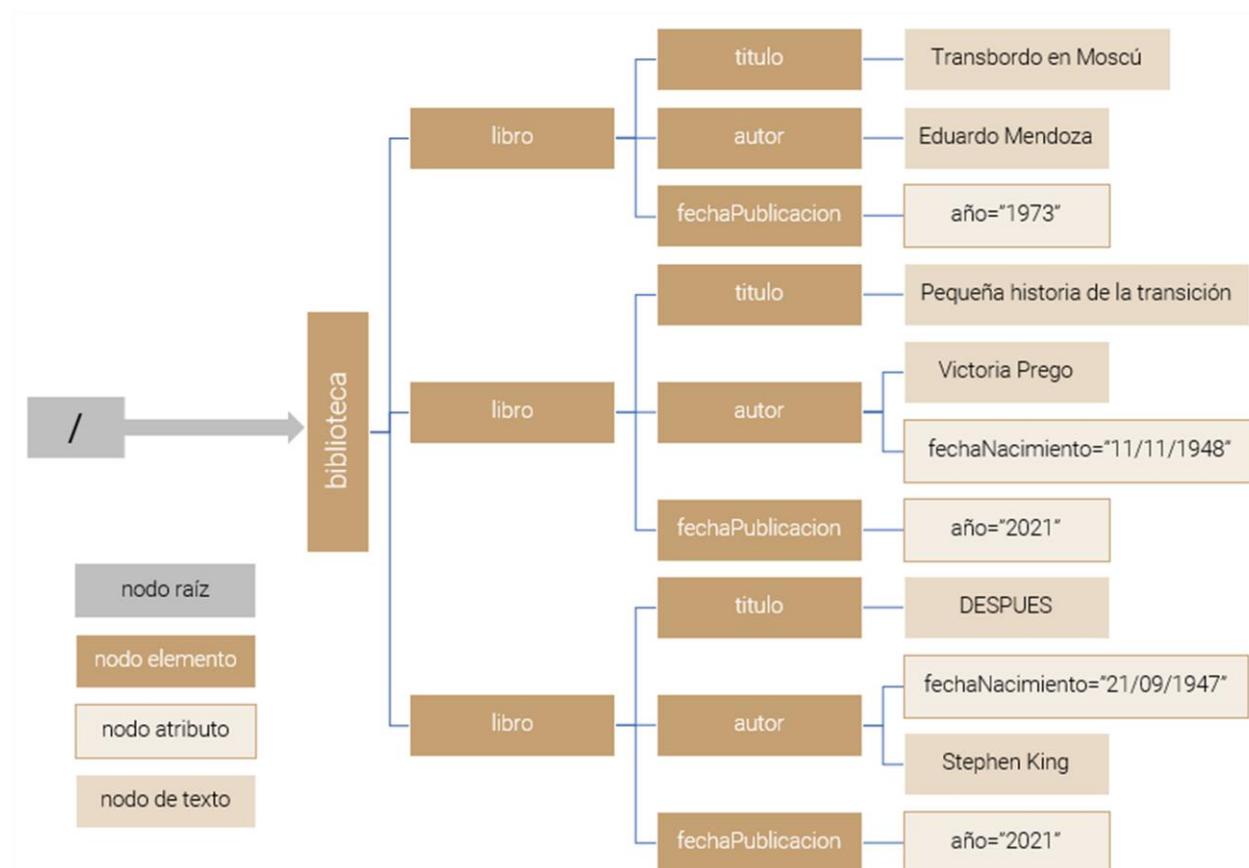


Figura 2.2. Árbol de nodos XML.

Los nodos atributos y de texto son diferentes a los nodos elemento. El nodo atributo no se considera como hijo, sino como una etiqueta adosada al elemento. El texto contenido por una etiqueta sí que se considera hijo del elemento.

2.2. Expresiones en XPath

Una expresión XPath es una cadena de texto que representa un recorrido en el árbol del documento XML.

Las rutas que se muestran trabajando en modo terminal en GNU/Linux son muy similares a expresiones de XPath sencillas.

La evaluación de una expresión XPath consiste en buscar si hay nodos en el documento XML que se ajustan al recorrido definido en la expresión. Y el resultado de la evaluación serán todos los nodos que se ajustan a la expresión. Es importante tener en cuenta que, para poder evaluar una expresión XPath, el documento debe estar bien formado.

Esta evaluación consta de una serie de búsquedas o pasos de localización, y cada una de ellos tiene las siguientes partes:

- **eje:** permite seleccionar un subconjunto de nodos del documento XML basándose en sus nombres, y corresponde a recorridos en el árbol del documento.
- **prueba de nodo:** funciona como un filtro que concreta un conjunto de nodos entre todos los que se sitúan en el eje.
- **predicado:** establece las condiciones. Se escribe entre corchetes a continuación del eje. Si el eje y la prueba de nodo han seleccionado unos nodos, el predicado permite restringir esa selección a los que cumplan determinadas condiciones.

Los pasos de localización de una expresión XPath siguen la sintaxis:

```
Eje::prueba de nodo[predicado1][predicado2]...
```

En la ruta nos podemos encontrar con:

- /, que actúa como separador de los pasos de una ruta.
- ::, que actúa como separador entre el eje y la prueba de nodo.

Veamos ahora cuáles son los caracteres que se pueden utilizar para crear expresiones XPath:

Patrón	Descripción
/	Si está al principio de la expresión indica el nodo raíz; si no, indica "hijo". Debe ir seguida del nombre de un elemento.
//	Para referenciar a un nodo que es descendiente de otro. También es válido para el nodo raíz.
.	Referencia al nodo actual.
..	Referencia al nodo padre.
*	Referencia a cualquier contenido de un nodo.
[]	Referencia a un nodo por su nombre y permite hacer selecciones. Su contenido es el predicado.
@	Referencia el contenido de un atributo.

Tabla 1. Patrones para la creación de expresiones XPath.

2.3. Ejes (Axis, en inglés)

Un eje forma parte de una expresión XPath.

Los ejes, dentro de una expresión XPath, definen la dirección que se toma en la estructura jerárquica a partir del nodo de contexto.

De esta forma, la expresión XPath, de la que forma parte el eje, permite seleccionar una colección de elementos.

La descripción de los tipos de ejes de XPath (archivo Tipos_ejes_XPath.pdf) está disponible en los [Documentos de trabajo y estudio](#).

Veamos con ejemplos cómo funcionan los ejes. Tomamos como base el código del Ejemplo 1.



EJEMPLO 2

Vamos a crear diferentes expresiones XPath utilizando los patrones vistos antes.

Los resultados se han obtenido utilizando la herramienta online <https://www.freeformatter.com/xpath-tester.html#ad-output>.

También está disponible online la herramienta <http://videlibri.sourceforge.net/cgi-bin/xidelcgi>.

Patrón / (raíz o hijo)

Ruta	Resultado
/biblioteca/libro/autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
/biblioteca/autor	No devuelve nada, porque <autor> no es hijo de <biblioteca>.
/biblioteca/libro/autor /@fechaNacimiento	Attribute='fechaNacimiento=11/11/1948' Attribute='fechaNacimiento=21/09/1947'

Tabla 2. Ejemplos de uso del patrón /.

En el primer ejemplo se está obteniendo el contenido de <autor>. En el tercer ejemplo se está accediendo al atributo fechaNacimiento de los autores que la tienen como atributo de <autor>.

Patrón // (descendiente)

Ruta	Resultado
/biblioteca//autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//@año	Attribute='año=1973 ' Attribute='año=2021 ' Attribute='año=2021 '

Tabla 3. Ejemplos de uso del patrón //.

En el primer ejemplo se referencia al nodo autor que es descendiente de biblioteca y muestra el mismo resultado que el Ejemplo 1 del anterior patrón. En el segundo ejemplo se muestra el mismo resultado, ya que la ruta es equivalente. En el tercer ejemplo solo se muestran los años de edición, atributo que tienen todos los nodos descendientes.

Patrón ../ (elemento padre)

Ruta	Resultado
/biblioteca/libro/autor /@fechaNacimiento/..	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//@fechaNacimiento/ ../	Element='<libro><titulo>Pequeña historia de la Transición</titulo><autor fechaNacimiento="11/11/1948">Victoria Prego</autor><fechaPublicacion año="2021"/></libro>' Element='<libro><titulo>DESPUES</titulo><autor fechaNacimiento="21/09/1947">Stephen King</autor><fechaPublicacion año="2021"/></libro>'

Tabla 4. Ejemplos de uso del patrón ../.

En el primer ejemplo se muestra la fecha de nacimiento de aquellos autores que la tienen y también el nombre del autor al hacer referencia al nodo <autor> (..). En el segundo ejemplo, además de la fecha de nacimiento de los autores que la tienen, también se muestra información de los nodos padre de dos niveles, es decir, información del libro.

Patrón | (permite indicar varios recorridos)

Ruta	Resultado
<code>//autor//titulo</code>	Element='<titulo>Transbordo en Moscú</titulo>' Element='<autor>Eduardo Mendoza</autor>' Element='<titulo>Pequeña historia de la Transición</titulo>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<titulo>DESPUES</titulo>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
<code>//autor//titulo//@año</code>	Element='<titulo>Transbordo en Moscú</titulo>' Element='<autor>Eduardo Mendoza</autor>'Attribute='año=1973 ' Element='<titulo>Pequeña historia de la Transición</titulo>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>'Attribute='año=2021 ' Element='<titulo>DESPUES</titulo>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'Attribute='año=2021 '

Tabla 5. Ejemplos de uso del patrón |.

En el ejemplo primero nos muestra el autor y el título, hace dos recorridos. En el segundo ejemplo muestra el autor, el título y el atributo año de publicación. Hay tres recorridos.

2.4. Predicado

Predicado es una expresión booleana que añade un nivel de selección al paso de localización.

XPath permite usar una serie de predicados acompañando a los ejes, cuya función es hacer una búsqueda más precisa del elemento que queremos.

El predicado constituye la tercera fase de un paso de localización, y es opcional. Contiene los criterios de filtro y se escribe como expresiones que pueden contener expresiones de ruta, funciones, operadores y cadenas, entre otros.

El ejemplo siguiente muestra la sintaxis utilizada:

```
child::libro [last()]
```


Hay que indicar el tipo de eje y, a continuación, '::. Luego el nodo del árbol que se va a comprobar y, opcionalmente, un tipo de predicado. El predicado va separado por corchetes '[' y se escribe al final de la ruta de localización.

En el ejemplo se ha aplicado el eje Child al nodo libro. Se ha utilizado el predicado [last()], cuya función es devolver el último de los elementos hermanos del nodo activo. Globalmente, la expresión indica que se selecciona el último hijo del nodo libro.

La ventaja del uso de los predicados es que, una vez se han seleccionado varios nodos simultáneamente, con los predicados se puede seleccionar uno en concreto que cumple alguna condición específica. Por ejemplo, en la siguiente ruta de localización, en el archivo XML, que contiene recetas, se está seleccionando el nombre del ingrediente cuyo código es '1'.

/receta/ingredientes/ingrediente[@codigo="1"]/nombre

La tabla siguiente incluye todo aquello que se puede incluir en un predicado:

Elementos a incluir en un predicado	Descripción	
Ejes	Permiten seleccionar el subárbol dentro del nodo contexto que cumple un patrón. Pueden ser o no de contenido.	
	child	Eje utilizado por defecto. Su forma habitual es la barra, "/", aunque también puede ponerse /child::
	attribute	Selecciona los atributos que se quiera. Es el único eje que no es de contenido.
	descendant	Selecciona todos los nodos que descienden del conjunto de nodos contextos. Se corresponde con la doble barra, //, aunque se puede usar descendant::
	self	Referido al nodo contexto, y se corresponde con el punto ".".
	parent	Selecciona los nodos padre. Se referencia con los dos puntos, "..".
	ancestor	Devuelve todos los nodos de los que el nodo contexto es descendiente.
Nodos test	Permiten restringir lo que devuelve una expresión Xpath. Se agrupan en función de los ejes a los que se puede aplicar.	
	Aplicable a cualquier eje.	
		"*", solo devuelve elementos, atributos o espacios de nombres, pero no permite obtener nodos de texto o comentarios de cualquier tipo.
		nod(), devuelve los nodos de todos los tipos.
	Solo aplicables a ejes de contenido	
		text(), devuelve cualquier nodo de tipo texto.
		comment(), devuelve cualquier nodo de tipo comentario.
		processing-instruction(), devuelve cualquier tipo de instrucción de proceso.

Tabla 6. Elementos de un predicado.

Los predicados pueden unirse utilizando operadores lógicos: and, or o not.

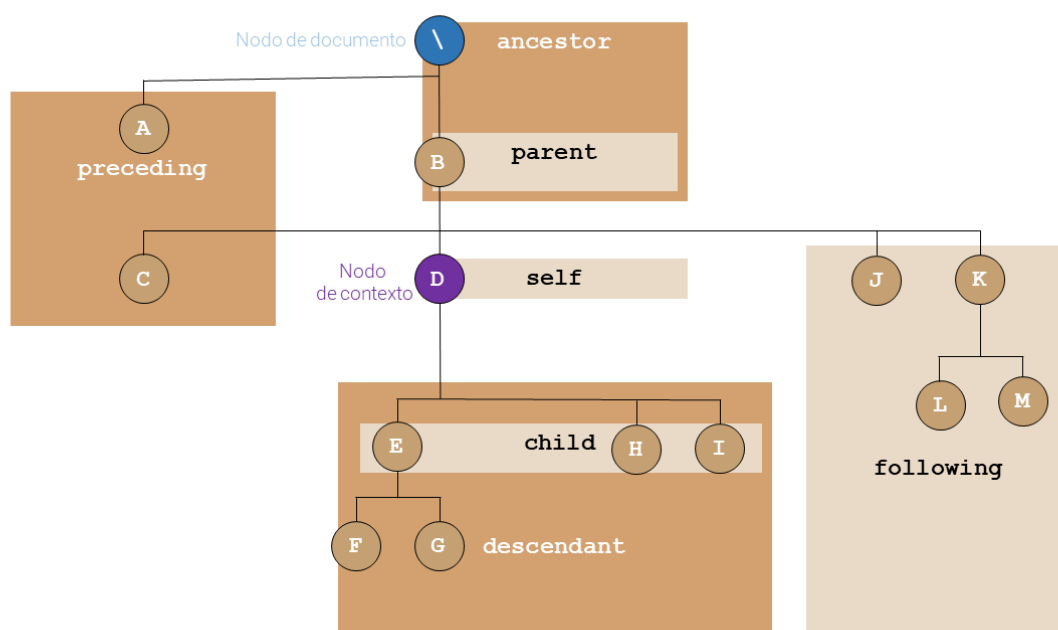


Figura 2.3. Relación entre los ejes y nodos.



EJEMPLO 3

Veamos ejemplos de uso de diferentes predicados sobre el mismo código del Ejemplo 1. Los resultados se han obtenido utilizando la herramienta online <https://www.freeformatter.com/xpath-tester.html#ad-output>.

Recuerda que el predicado se escribe entre corchetes a continuación del eje.

Patrón [`@atributo`] (selección de elementos que tienen el atributo dado).

Ruta	Resultado
<code>//autor[@fechaNacimiento]</code>	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'

Tabla 7. Ejemplos de uso del patrón [`@atributo`].

Patrón [número] (si hay varios resultados, selecciona uno de ellos por número de orden; last () selecciona el último de ellos)

Ruta	Resultado
//libro[1]	Element='<libro> <titulo>Transbordo en Moscú</titulo> <autor>Eduardo Mendoza</autor> <fechaPublicacion año="1973"/></libro>'
//libro[last()-1]	Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor> <fechaPublicacion año="2021"/></libro>'

Tabla 8. Ejemplos de uso del patrón [numero].

Patrón [condicion] (selecciona nodos que cumplen la condición).

En la condición se pueden utilizar los operadores incluidos en las tablas 10, 11 y 12. También puede utilizar el valor de un atributo (utilizando @) o el texto que contiene el elemento.

En el primer caso se seleccionan los elementos <fechaPublicacion>, cuyo atributo año es mayor que 2010. En el segundo caso se seleccionan los elementos <libro>, cuyo <autor> tiene como contenido "Victoria Prego". En el tercer caso se hace referencia al propio valor del elemento seleccionado.

Ruta	Resultado
//fechaPublicacion[@año>2010]	Element='<fechaPublicacion año="2021"/>' Element='<fechaPublicacion año="2021"/>'
//libro[autor="Victoria Prego"]	Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor><fechaPublicacion año="2021"/> </libro>'
//autor[.="Victoria Prego"]	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>'
//libro[autor="Victoria Prego" or fechaPublicacion/@año="1973"]	Element='<libro> <titulo>Transbordo en Moscú</titulo> <autor>Eduardo Mendoza</autor> <fechaPublicacion año="1973"/></libro>' Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor> <fechaPublicacion año="2021"/></libro>'

Tabla 9. Ejemplos de uso del patrón [condicion].

En el cuarto caso se ha utilizado un predicado con una condición compuesta. En concreto, se seleccionan libros de Victoria Prego o que se hayan publicado en 1973.

Las tablas siguientes son un resumen de los operadores disponibles, clasificados en operadores aritméticos, lógicos y de comparación.

Operadores aritméticos	Función
+	suma
-	resta
*	multiplicación
div	resultado entero de la división
mod	módulo

Tabla 10. Operadores aritméticos.

Operadores de comparación	Función
=	igual
!=	no igual
<	menor que
>	mayor que
<=	menor que o igual
>=	mayor que o igual

Tabla 11. Operadores de comparación.

Operadores lógicos	Función
and	operador Y lógico
or	operador O lógico

Tabla 12. Operadores lógicos.

2.5. Funciones de Xpath

Xpath 1.0 define 25 funciones y, a partir de Xpath 2.0, se dispone de 111 funciones para la descripción de rutas de localización. En el siguiente [enlace](#) están disponibles las recomendaciones del W3C para las funciones XPath.

Las funciones XPath se pueden usar en las expresiones XPath y en los predicados.

La tabla siguiente describe las funciones más importantes que se pueden utilizar desde XPath.

Función	Descripción
boolean()	Convierte a un valor booleano. Si es un número, es true si es distinto de cero. Si es un conjunto de nodos, es true si contiene algún nodo. Si es una cadena, es true si no es la cadena vacía.
not()	Aplicada sobre un predicado devuelve true si el predicado es falso, y falso si el predicado es true.
true()	Devuelve el valor true.
false()	Devuelve el valor false.
count()	Devuelve el número de nodos que forman un conjunto de nodos.
name()	Devuelve un nombre de un nodo.
local-name()	Devuelve el nombre del nodo actual o del primer nodo de un conjunto de nodos.
namespace-uri()	Devuelve el URI del espacio de nombres.
position()	Devuelve la posición de un nodo en su contexto comenzando en 1. Por ejemplo, para seleccionar los dos primeros elementos de tipo elemento de un archivo XML se indica <code>//elemento[position()<=2]</code>
last()	Devuelve el último elemento del conjunto dado.
normalize-space()	Devuelve el resultado de normalizar una cadena eliminando espacios en blanco redundantes. <code>normalize-space(' Hola mundo ') = 'Hola mundo'</code>
string()	Convierte a cadena de texto. Si es un número devuelve su representación como cadena. Si es booleano devuelve true o false. Si es un conjunto de nodos se aplica al primer nodo. Si es un nodo devuelve el valor de cadena de dicho nodo.
concat()	Devuelve dos cadenas de texto concatenadas. <code>concat('uno', 'dos') = 'unodos'</code>
string-length()	Devuelve la cantidad de caracteres que forman una cadena de caracteres. <code>string-length('camina') = 6</code>
sum()	Devuelve la suma de los valores numéricos de cada nodo en un conjunto de nodos determinado.

Tabla 13. Funciones XPath.

Ejemplos de uso de funciones:

Aplicamos diferentes funciones sobre el código del Ejemplo 1.

1. Todos los datos del segundo ciclo:

```
/biblioteca/libro[position()=2]
```

Muestra los datos del libro Pequeña historia de la Transición.

2. Todos los datos del último libro:

```
/biblioteca/libro[last()]
```

Muestra los datos del tercer libro, de título *Después*.

3. Nombres de los autores.

```
/biblioteca/libro/autor[name()]
```

Muestra los nombres de los tres autores.

4. Todos los nodos cuyo nombre tenga una longitud mayor que 10.

```
//*[string-length(name())>10]
```

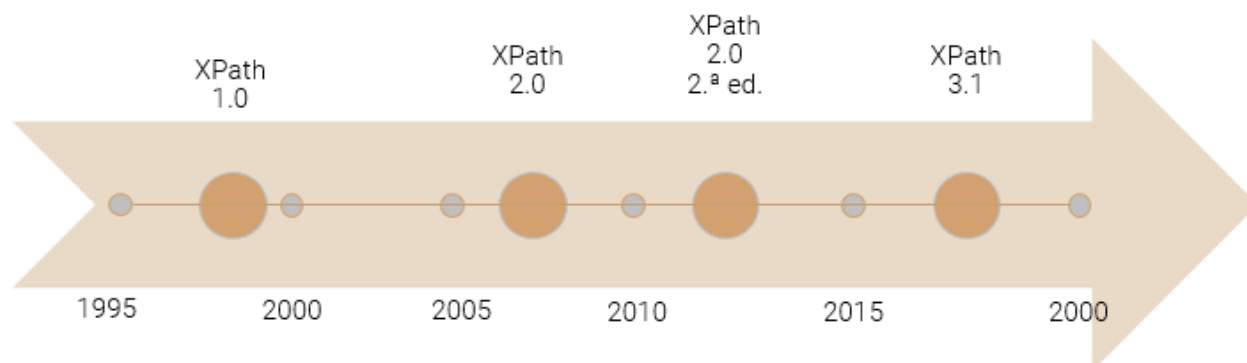
Muestra los nodos relativos a la fecha de publicación, ya que tiene un nombre de más de 10 caracteres.

5. Los datos de los libros de las dos primeras posiciones.

```
//libro[position()<=2]
```

2. XPath. Estructura y sintaxis

Xpath aparece en 1999, la última versión data de 2017 y las recomendaciones del W3C están disponibles en este [enlace](#).



El mecanismo de funcionamiento de XPath se basa en la interpretación del documento XML como una secuencia de elementos ordenados bajo la estructura de árbol. Esta estructura en árbol de los datos sería equivalente a la del modelo de objetos del documento (DOM), ya estudiado.

Para localizar los elementos en el documento XML, XPath utiliza el sistema de rutas de localización de Unix/GNU Linux, cuyos elementos son los nodos, los ejes, las pruebas de nodo y los predicados. Todos ellos se estudian a continuación.

Esta tecnología suele estar integrada junto con otras herramientas para el procesamiento de documentos XML. En concreto y globalmente están:

- XPath: para la navegación dentro de documentos XML.
- XQuery: para hacer consultas a documentos XML. XPath localiza los elementos y Xquery extrae información de ellos. Se estudia en la próxima unidad.
- XSLT: para la transformación de los datos de documentos XML.

Como resumen, se puede decir que XPath es un lenguaje que permite:

- Transformar el formato de los datos de un documento XML.
- Definir un vocabulario que ha de cumplir un documento XML.
- Acceder a los datos de un documento XML.

XPath trabaja con expresiones que son como condiciones (predicados) que se aplican sobre una estructura de árbol correspondiente a la jerarquía de datos de un documento XML, y devuelve todo lo que coincide (match) con ese predicado. Estas expresiones se pueden asimilar, en cierta medida, a las expresiones regulares.

2.1. Modelo de datos de XPath

XPath representa la estructura jerárquica de los datos del documento XML como un árbol dirigido, cuyos elementos son nodos y un elemento es padre de los elementos que contiene.

Hay dos nodos especialmente importantes, que son:

- **nodo contexto:** es el nodo del árbol que representa el documento XML y que se toma como punto partida de la expresión.
- **nodo actual:** es el nodo que se obtiene al final de una expresión, o, si son varios nodos, cada uno de los nodos de la secuencia devuelta por la expresión.

Hay siete tipos de nodos:

1. Nodo raíz o nodo documento

El nodo raíz es la raíz del árbol y contiene el ejemplar del documento XML. Se identifica por '/'.

El nodo raíz puede tener como hijos nodos elemento, nodos de instrucción de procesamiento y comentario, correspondientes a las instrucciones de procesamiento y comentarios que aparezcan en el prólogo.

El valor de cadena del nodo raíz es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo raíz, en orden, del documento.

2. Nodos elemento

Hay un nodo elemento por cada elemento del documento XML, a excepción de la declaración XML y la definición del tipo de documento.

Todos los nodos elemento tienen un nodo padre, que es el nodo raíz del documento.

Los nodos elemento pueden tener un identificador único (ID). Este es el valor del atributo que se declararía en la DTD como de tipo ID. No puede haber dos elementos en un documento con el mismo ID.

3. Nodos texto

Son nodos texto los caracteres del documento XML que no están marcados con ninguna etiqueta. En cada nodo texto se agrupan todos los datos de carácter que sea posible.

Los nodos texto no pueden tener hijos.

4. Nodos atributo

Cada nodo elemento puede tener asociado un conjunto de nodos atributo. El nodo elemento es el padre de cada uno de esos nodos atributo, pero los nodos atributo no son hijos de su elemento padre. Solo se consideran etiquetas añadidas al nodo elemento.

Los nodos atributo no pueden tener hijos.

Cada uno de estos nodos consta de un identificador y de un valor asignado.

5. Nodos espacio de nombres

Los nombres de elementos y atributos de un documento XML validado tienen asignado un espacio de nombres. Esta asignación viene dada en la DTD al principio del documento.

Un nodo elemento es padre de un nodo espacio de nombres, pero no a la inversa. Es decir, un nodo espacio de nombre no es hijo de su nodo padre. Los elementos tienen nodos espacio de nombres con estas condiciones:

- Tienen uno por cada atributo que empiece por xmlns:.
- Para cada atributo situado sobre un elemento previo cuyo nombre empiece por xmlns:.

6. Nodos instrucción de procesamiento

Las instrucciones de procesamiento de los documentos XML se encuentran fuera del árbol del documento y se denominan nodos de instrucción de procesamiento.

El nodo de instrucción de procesamiento comienza por <? y termina con ?>.

7. Nodos comentario

En un documento XML, XPath procesa el contenido marcado como comentario como un nodo de comentario.

El valor de cadena de los comentarios es el contenido del comentario sin incluir el inicio <!-- ni la terminación -->.



IMPORTANTE

La declaración DOCTYPE no se considera como nodo.



EJEMPLO 1

Tenemos el documento XML siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <titulo>Transbordo en Moscú</titulo>
    <autor>Eduardo Mendoza</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pequeña historia de la Transición</titulo>
    <autor fechaNacimiento="11/11/1948">Victoria Prego</autor>
    <fechaPublicacion año="2021"/>
  </libro>
  <libro>
    <titulo>DESPUES</titulo>
    <autor fechaNacimiento="21/09/1947">Stephen King</autor>
    <fechaPublicacion año="2021"/>
  </libro>
</biblioteca>
```

La figura siguiente muestra el esquema en árbol con todos los nodos y sus tipos que se generarían.

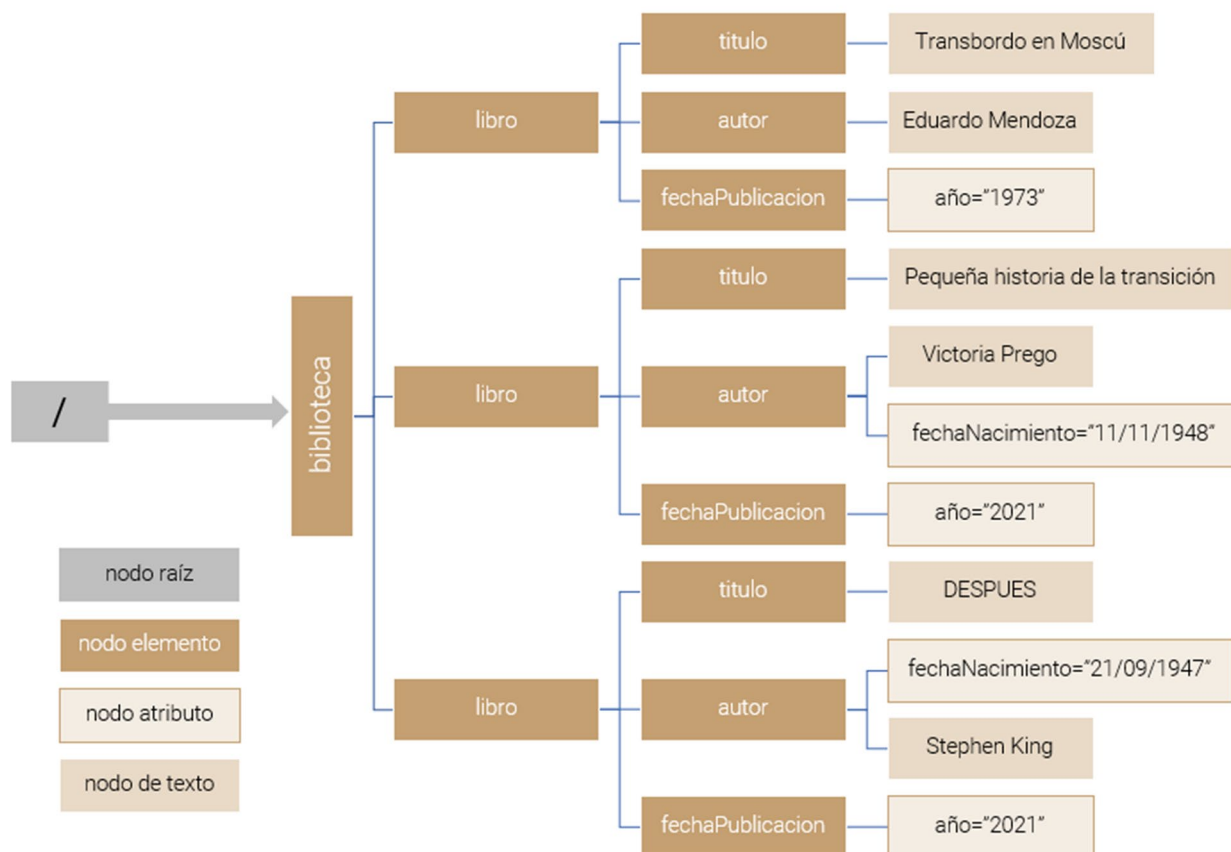


Figura 2.2. Árbol de nodos XML.

Los nodos atributos y de texto son diferentes a los nodos elemento. El nodo atributo no se considera como hijo, sino como una etiqueta adosada al elemento. El texto contenido por una etiqueta sí que se considera hijo del elemento.

2.2. Expresiones en XPath

Una expresión XPath es una cadena de texto que representa un recorrido en el árbol del documento XML.

Las rutas que se muestran trabajando en modo terminal en GNU/Linux son muy similares a expresiones de XPath sencillas.

La evaluación de una expresión XPath consiste en buscar si hay nodos en el documento XML que se ajustan al recorrido definido en la expresión. Y el resultado de la evaluación serán todos los nodos que se ajustan a la expresión. Es importante tener en cuenta que, para poder evaluar una expresión XPath, el documento debe estar bien formado.

Esta evaluación consta de una serie de búsquedas o pasos de localización, y cada una de ellos tiene las siguientes partes:

- **eje:** permite seleccionar un subconjunto de nodos del documento XML basándose en sus nombres, y corresponde a recorridos en el árbol del documento.
- **prueba de nodo:** funciona como un filtro que concreta un conjunto de nodos entre todos los que se sitúan en el eje.
- **predicado:** establece las condiciones. Se escribe entre corchetes a continuación del eje. Si el eje y la prueba de nodo han seleccionado unos nodos, el predicado permite restringir esa selección a los que cumplan determinadas condiciones.

Los pasos de localización de una expresión XPath siguen la sintaxis:

```
Eje::prueba de nodo[predicado1][predicado2]...
```

En la ruta nos podemos encontrar con:

- /, que actúa como separador de los pasos de una ruta.
- ::, que actúa como separador entre el eje y la prueba de nodo.

Veamos ahora cuáles son los caracteres que se pueden utilizar para crear expresiones XPath:

Patrón	Descripción
/	Si está al principio de la expresión indica el nodo raíz; si no, indica "hijo". Debe ir seguida del nombre de un elemento.
//	Para referenciar a un nodo que es descendiente de otro. También es válido para el nodo raíz.
.	Referencia al nodo actual.
..	Referencia al nodo padre.
*	Referencia a cualquier contenido de un nodo.
[]	Referencia a un nodo por su nombre y permite hacer selecciones. Su contenido es el predicado.
@	Referencia el contenido de un atributo.

Tabla 1. Patrones para la creación de expresiones XPath.

2.3. Ejes (Axis, en inglés)

Un eje forma parte de una expresión XPath.

Los ejes, dentro de una expresión XPath, definen la dirección que se toma en la estructura jerárquica a partir del nodo de contexto.

De esta forma, la expresión XPath, de la que forma parte el eje, permite seleccionar una colección de elementos.

La descripción de los tipos de ejes de XPath (archivo Tipos_ejes_XPath.pdf) está disponible en los [Documentos de trabajo y estudio](#).

Veamos con ejemplos cómo funcionan los ejes. Tomamos como base el código del Ejemplo 1.



EJEMPLO 2

Vamos a crear diferentes expresiones XPath utilizando los patrones vistos antes.

Los resultados se han obtenido utilizando la herramienta online <https://www.freeformatter.com/xpath-tester.html#ad-output>.

También está disponible online la herramienta <http://videlibri.sourceforge.net/cgi-bin/xidelcgi>.

Patrón / (raíz o hijo)

Ruta	Resultado
/biblioteca/libro/autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
/biblioteca/autor	No devuelve nada, porque <autor> no es hijo de <biblioteca>.
/biblioteca/libro/autor /@fechaNacimiento	Attribute='fechaNacimiento=11/11/1948' Attribute='fechaNacimiento=21/09/1947'

Tabla 2. Ejemplos de uso del patrón /.

En el primer ejemplo se está obteniendo el contenido de <autor>. En el tercer ejemplo se está accediendo al atributo fechaNacimiento de los autores que la tienen como atributo de <autor>.

Patrón // (descendiente)

Ruta	Resultado
/biblioteca//autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//autor	Element='<autor>Eduardo Mendoza</autor>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//@año	Attribute='año=1973' Attribute='año=2021' Attribute='año=2021'

Tabla 3. Ejemplos de uso del patrón //.

En el primer ejemplo se referencia al nodo autor que es descendiente de biblioteca y muestra el mismo resultado que el Ejemplo 1 del anterior patrón. En el segundo ejemplo se muestra el mismo resultado, ya que la ruta es equivalente. En el tercer ejemplo solo se muestran los años de edición, atributo que tienen todos los nodos descendientes.

Patrón ../ (elemento padre)

Ruta	Resultado
/biblioteca/libro/autor /@fechaNacimiento/..	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//@fechaNacimiento/ ../	Element='<libro><titulo>Pequeña historia de la Transición</titulo><autor fechaNacimiento="11/11/1948">Victoria Prego</autor><fechaPublicacion año="2021"/></libro>' Element='<libro><titulo>DESPUES</titulo><autor fechaNacimiento="21/09/1947">Stephen King</autor><fechaPublicacion año="2021"/></libro>'

Tabla 4. Ejemplos de uso del patrón ../.

En el primer ejemplo se muestra la fecha de nacimiento de aquellos autores que la tienen y también el nombre del autor al hacer referencia al nodo <autor> (..). En el segundo ejemplo, además de la fecha de nacimiento de los autores que la tienen, también se muestra información de los nodos padre de dos niveles, es decir, información del libro.

Patrón | (permite indicar varios recorridos)

Ruta	Resultado
//autor//titulo	Element='<titulo>Transbordo en Moscú</titulo>' Element='<autor>Eduardo Mendoza</autor>' Element='<titulo>Pequeña historia de la Transición</titulo>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<titulo>DESPUES</titulo>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'
//autor//titulo//@año	Element='<titulo>Transbordo en Moscú</titulo>' Element='<autor>Eduardo Mendoza</autor>'Attribute='año=1973' Element='<titulo>Pequeña historia de la Transición</titulo>' Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>'Attribute='año=2021' Element='<titulo>DESPUES</titulo>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'Attribute='año=2021'

Tabla 5. Ejemplos de uso del patrón |.

En el ejemplo primero nos muestra el autor y el título, hace dos recorridos. En el segundo ejemplo muestra el autor, el título y el atributo año de publicación. Hay tres recorridos.

2.4. Predicado

Predicado es una expresión booleana que añade un nivel de selección al paso de localización.

XPath permite usar una serie de predicados acompañando a los ejes, cuya función es hacer una búsqueda más precisa del elemento que queremos.

El predicado constituye la tercera fase de un paso de localización, y es opcional. Contiene los criterios de filtro y se escribe como expresiones que pueden contener expresiones de ruta, funciones, operadores y cadenas, entre otros.

El ejemplo siguiente muestra la sintaxis utilizada:

```
child::libro [last()]
```

Hay que indicar el tipo de eje y, a continuación, '::'. Luego el nodo del árbol que se va a comprobar y, opcionalmente, un tipo de predicado. El predicado va separado por corchetes '[]' y se escribe al final de la ruta de localización.

En el ejemplo se ha aplicado el eje Child al nodo libro. Se ha utilizado el predicado [last()], cuya función es devolver el último de los elementos hermanos del nodo activo. Globalmente, la expresión indica que se selecciona el último hijo del nodo libro.

La ventaja del uso de los predicados es que, una vez se han seleccionado varios nodos simultáneamente, con los predicados se puede seleccionar uno en concreto que cumple alguna condición específica. Por ejemplo, en la siguiente ruta de localización, en el archivo XML, que contiene recetas, se está seleccionando el nombre del ingrediente cuyo código es '1'.

```
/receta/ingredientes/ingrediente[@codigo="1"]/nombre
```

La tabla siguiente incluye todo aquello que se puede incluir en un predicado:

Elementos a incluir en un predicado	Descripción	
Ejes	Permiten seleccionar el subárbol dentro del nodo contexto que cumple un patrón. Pueden ser o no de contenido.	
	child	Eje utilizado por defecto. Su forma habitual es la barra, "/", aunque también puede ponerse /child::
	attribute	Selecciona los atributos que se quiera. Es el único eje que no es de contenido.
	descendant	Selecciona todos los nodos que descienden del conjunto de nodos contextos. Se corresponde con la doble barra, //, aunque se puede usar descendant::
	self	Referido al nodo contexto, y se corresponde con el punto ".".
	parent	Selecciona los nodos padre. Se referencia con los dos puntos, "..".
	ancestor	Devuelve todos los nodos de los que el nodo contexto es

		descendiente.
Nodos test	Permiten restringir lo que devuelve una expresión Xpath. Se agrupan en función de los ejes a los que se puede aplicar.	
	Aplicable a cualquier eje.	
		"*", solo devuelve elementos, atributos o espacios de nombres, pero no permite obtener nodos de texto o comentarios de cualquier tipo.
		nod () , devuelve los nodos de todos los tipos.
	Solo aplicables a ejes de contenido	
		text () , devuelve cualquier nodo de tipo texto.
		comment () , devuelve cualquier nodo de tipo comentario.
		processing-instruction() , devuelve cualquier tipo de instrucción de proceso.

Tabla 1. Elementos de un predicado.

Los predicados pueden unirse utilizando operadores lógicos: and, or o not.

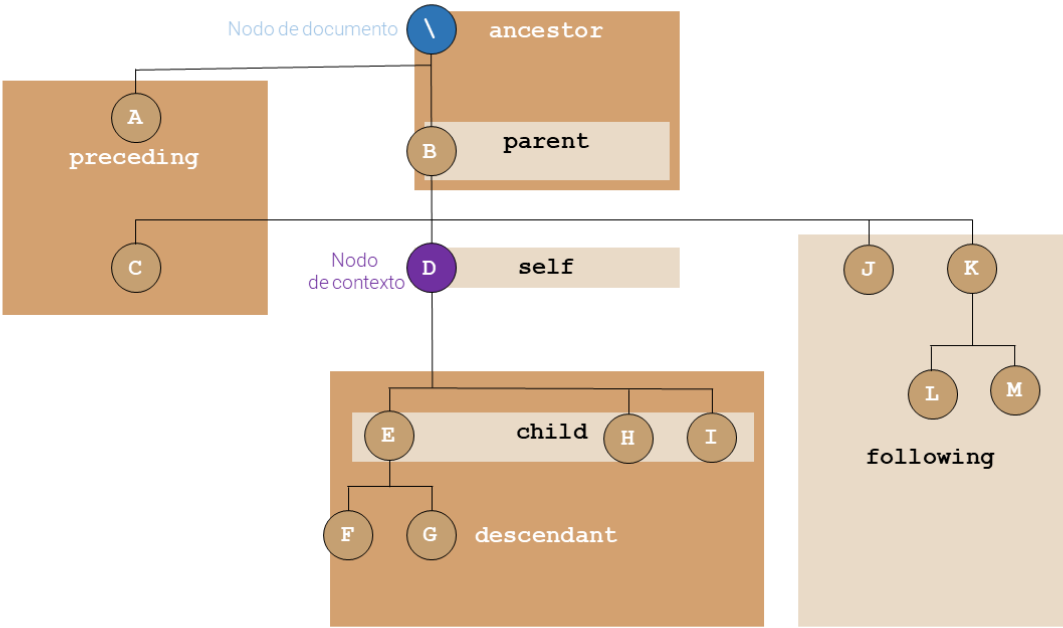


Figura 2.1. Relación entre los ejes y nodos.



EJEMPLO 3

Veamos ejemplos de uso de diferentes predicados sobre el mismo código del Ejemplo 1. Los resultados se han obtenido utilizando la herramienta online <https://www.freeformatter.com/xpath-tester.html#ad-output>.

Recuerda que el predicado se escribe entre corchetes a continuación del eje.

Patrón [@atributo] (selección de elementos que tienen el atributo dado).

Ruta	Resultado
//autor[@fechaNacimiento]	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>' Element='<autor fechaNacimiento="21/09/1947">Stephen King</autor>'

Tabla 2. Ejemplos de uso del patrón [@atributo].

Patrón [número] (si hay varios resultados, selecciona uno de ellos por número de orden; last () selecciona el último de ellos)

Ruta	Resultado
//libro[1]	Element='<libro> <titulo>Transbordo en Moscú</titulo> <autor>Eduardo Mendoza</autor> <fechaPublicacion año="1973"/></libro>'
//libro[last()-1]	Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor> <fechaPublicacion año="2021"/></libro>'

Tabla 3. Ejemplos de uso del patrón [numero].

Patrón [condicion] (selecciona nodos que cumplen la condición).

En la condición se pueden utilizar los operadores incluidos en las tablas 10, 11 y 12. También puede utilizar el valor de un atributo (utilizando @) o el texto que contiene el elemento.

En el primer caso se seleccionan los elementos <fechaPublicacion>, cuyo atributo año es mayor que 2010. En el segundo caso se seleccionan los elementos <libro>, cuyo <autor> tiene como contenido "Victoria Prego". En el tercer caso se hace referencia al propio valor del elemento seleccionado.

Ruta	Resultado
//fechaPublicacion[@año>2010]	Element='<fechaPublicacion año="2021"/>' Element='<fechaPublicacion año="2021"/>'
//libro[autor="Victoria Prego"]	Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor><fechaPublicacion año="2021"/> </libro>'
//autor[.="Victoria Prego"]	Element='<autor fechaNacimiento="11/11/1948">Victoria Prego</autor>'
//libro[autor="Victoria Prego" or fechaPublicacion/@año="1973"]	Element='<libro> <titulo>Transbordo en Moscú</titulo> <autor>Eduardo Mendoza</autor> <fechaPublicacion año="1973"/></libro>' Element='<libro> <titulo>Pequeña historia de la Transición</titulo> <autor fechaNacimiento="11/11/1948">Victoria Prego</autor> <fechaPublicacion año="2021"/></libro>'

Tabla 4. Ejemplos de uso del patrón [condicion].

En el cuarto caso se ha utilizado un predicado con una condición compuesta. En concreto, se seleccionan libros de Victoria Prego o que se hayan publicado en 1973.

Las tablas siguientes son un resumen de los operadores disponibles, clasificados en operadores aritméticos, lógicos y de comparación.

Operadores aritméticos	Función
+	suma
-	resta
*	multiplicación
div	resultado entero de la división
mod	módulo

Tabla 5. Operadores aritméticos.

Operadores de comparación	Función
=	igual
!=	no igual
<	menor que
>	mayor que
<=	menor que o igual
>=	mayor que o igual

Tabla 6. Operadores de comparación.

Operadores lógicos	Función
and	operador Y lógico
or	operador O lógico

Tabla 7. Operadores lógicos.

2.5. Funciones de Xpath

Xpath 1.0 define 25 funciones y, a partir de Xpath 2.0, se dispone de 111 funciones para la descripción de rutas de localización. En el siguiente [enlace](#) están disponibles las recomendaciones del W3C para las funciones XPath.

Las funciones XPath se pueden usar en las expresiones XPath y en los predicados.

La tabla siguiente describe las funciones más importantes que se pueden utilizar desde XPath.

Función	Descripción
boolean()	Convierte a un valor booleano. Si es un número, es true si es distinto de cero. Si es un conjunto de nodos, es true si contiene algún nodo. Si es una cadena, es true si no es la cadena vacía.
not()	Aplicada sobre un predicado devuelve true si el predicado es falso, y falso si el predicado es true.
true()	Devuelve el valor true.
false()	Devuelve el valor false.
count()	Devuelve el número de nodos que forman un conjunto de nodos.
name()	Devuelve un nombre de un nodo.
local-name()	Devuelve el nombre del nodo actual o del primer nodo de un conjunto de nodos.
namespace-uri()	Devuelve el URI del espacio de nombres.
position()	Devuelve la posición de un nodo en su contexto comenzando en 1. Por ejemplo, para seleccionar los dos primeros elementos de tipo elemento de un archivo XML se indica <code>//elemento[position()<=2]</code>
last()	Devuelve el último elemento del conjunto dado.
normalize-space()	Devuelve el resultado de normalizar una cadena eliminando espacios en blanco redundantes. <code>normalize-space(' Hola mundo ') = 'Hola mundo'</code>
string()	Convierte a cadena de texto. Si es un número devuelve su representación como cadena. Si es booleano devuelve true o false. Si es un conjunto de nodos se aplica al primer nodo. Si es un nodo devuelve el valor de cadena de dicho nodo.
concat()	Devuelve dos cadenas de texto concatenadas. <code>concat('uno', 'dos') = 'unodos'</code>
string-length()	Devuelve la cantidad de caracteres que forman una cadena de caracteres. <code>string-length('camina') = 6</code>
sum()	Devuelve la suma de los valores numéricos de cada nodo en un conjunto de nodos determinado.

Tabla 8. Funciones XPath.

Ejemplos de uso de funciones:

Aplicamos diferentes funciones sobre el código del Ejemplo 1.

1. Todos los datos del segundo ciclo:

```
/biblioteca/libro[position()=2]
```

Muestra los datos del libro Pequeña historia de la Transición.

2. Todos los datos del último libro:

```
/biblioteca/libro[last()]
```

Muestra los datos del tercer libro, de título *Después*.

3. Nombres de los autores.

```
/biblioteca/libro/autor[name()]
```

Muestra los nombres de los tres autores.

4. Todos los nodos cuyo nombre tenga una longitud mayor que 10.

```
//*[string-length(name())>10]
```

Muestra los nodos relativos a la fecha de publicación, ya que tiene un nombre de más de 10 caracteres.

5. Los datos de los libros de las dos primeras posiciones.

```
//libro[position()<=2]
```

3. XSLT

XSLT (Transformaciones XSL) es un lenguaje de programación declarativo que permite generar documentos en diferentes formatos a partir de documentos XML.

Algunas de sus principales características son las siguientes:

- El documento XML es el documento inicial al que se aplican las transformaciones.
- La hoja de estilo XSLT es el documento que contiene las reglas de transformación que se van a aplicar al documento inicial XML. Estas hojas de estilo son también archivos XML.
- El procesador XSLT es el programa que aplica al documento inicial XML las reglas de transformación incluidas en la hoja de estilo XSLT y genera el documento final.
- El resultado de la ejecución del programa es un nuevo documento (que puede ser un documento XML o no).

La figura siguiente ilustra este proceso.

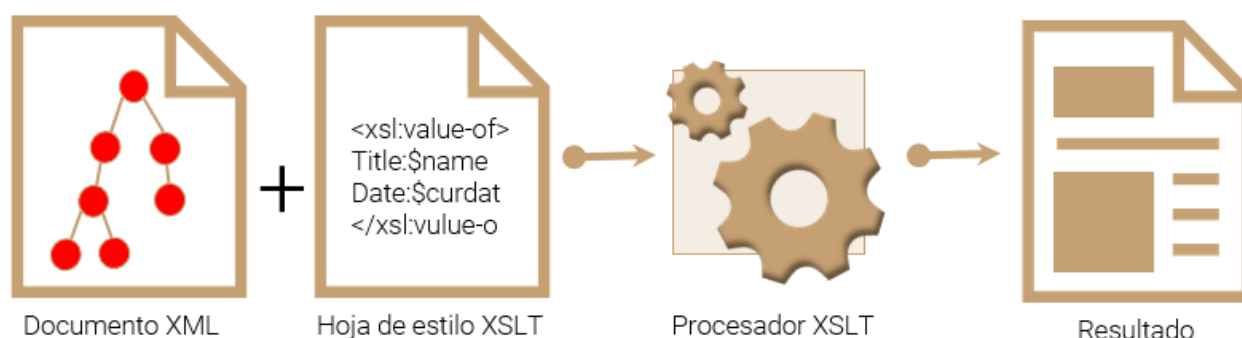


Figura 3.1. Proceso de transformación XSL.

El lenguaje XSLT está estandarizado por el W3C y ha publicado varias versiones, siendo la última la versión 2.0 de marzo de 2021, accesible desde este [enlace](#).

3.1. Estructura de una hoja de estilo o plantilla

Una hoja de estilo o plantilla XSLT es también un documento XML y debe estar bien formado. Las hojas de estilo se guardan siempre en archivos independientes con extensión .xsl, y deben comenzar con la declaración XML:

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

Los elementos básicos que puede contener una plantilla son los siguientes:

- **Elementos XSLT:** están precedidos por el prefijo xsl:, pertenecen al espacio de nombres xsl, están definidos en el estándar del lenguaje y son interpretados por cualquier procesador XSLT.
- **Elementos LRE (elemento de resultado literal):** son cualquier elemento que no sea instrucción. No pertenecen a XSLT, sino que se repiten en la salida sin más. En una plantilla todo lo que no empiece por 'xsl:' se considera un elemento literal y se transcribe exactamente a la salida.

El elemento raíz de la hoja de estilo XSLT es `stylesheet` (también puede utilizarse `transform`, ambas son equivalentes). Este elemento contendrá todos los demás, debe ir precedido por el prefijo `xsl` y el correspondiente al espacio de nombres para hojas de estilo XSLT. Los nombres de los elementos XSLT proceden de un mismo espacio de nombres, deben escribirse precedidos por el prefijo `xmlns` y la URL: `http://www.w3.org/1999/XSL/Transform`.

El elemento raíz queda así:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
.....
</xsl:stylesheet>
```

Entre las marcas de inicio y fin del elemento raíz `xsl:stylesheet` se escriben las reglas de transformación definidas mediante el elemento `xsl:template`. Cada regla indica qué instancias de los elementos del documento XML se van a transformar y cómo lo hará cada una de ellas.



EJEMPLO 4

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="//nombre">
<h2>
<xsl:value-of select="." />
</h2>
</xsl:template>
</xsl:stylesheet>
```

En este ejemplo, la regla se aplicará a todas las instancias del elemento *nombre*. Esto se indica mediante el atributo `match` que acompaña al elemento `xsl:template`. Entre las etiquetas de inicio y fin del elemento `xsl:template` se escribe la transformación que se debe realizar, es decir, qué texto y qué marcas se escribirán en el documento resultado de la transformación cada vez que se encuentre una instancia del elemento *nombre* en el documento origen.

`<xsl:value-of...>` recupera y escribe en el documento resultado el valor del elemento que está siendo procesado. Se estudia en el siguiente subapartado.

Dentro del elemento raíz `<xsl:stylesheet>`, además de las plantillas, se pueden encontrar los llamados elementos de alto nivel, como `<xsl:output>`



EJEMPLO 5

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="xml"/>
  <xsl:template match="/">
  </xsl:template>
</xsl:stylesheet>
```

En este código encontramos:

- El elemento de alto nivel `<xsl:output>` indica el tipo de salida producida.
- La instrucción `<xsl:template>` es una plantilla.
 - El atributo `match` indica los elementos afectados por la plantilla y contiene una expresión XPath.
 - El contenido de la instrucción define la transformación a aplicar (si la instrucción no contiene nada, como en el ejemplo anterior, sustituirá el nodo por nada, es decir, eliminará el nodo, aunque conservará el texto contenido en el elemento).

Para asociar un archivo de plantilla XSLT a un documento XML hay que añadir, después del prólogo, la línea siguiente:

```
<?xml-stylesheet type="text/xsl" href="ruta_del_archivo_xsl.xml"?>
```

Con ella se indica al procesador el lugar donde se encuentra la hoja de estilos con la que dar formato a los datos del documento XML.

3.2. Utilización de plantillas. Plantillas múltiples

El elemento `xsl:template` permite controlar el formato de salida que se aplica a ciertos datos de entrada. Dentro de la plantilla tenemos las instrucciones, que son también elementos XSL.



IMPORTANTE

Para probar los ejemplos no es necesario abrir los archivos en el navegador. Basta con comprobar el resultado en el editor que se esté utilizando.

Si se utiliza el navegador web hay que tener en cuenta que los navegadores no respetan los saltos de línea ni los espacios en blanco, ni muestran las etiquetas.

Retomamos el código del Ejemplo 1.

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <titulo>Transbordo en Moscú</titulo>
    <autor>Eduardo Mendoza</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pequeña historia de la Transición</titulo>
    <autor fechaNacimiento="11/11/1948">Victoria Prego</autor>
    <fechaPublicacion año="2021"/>
  </libro>
  <libro>
    <titulo>Después</titulo>
    <autor fechaNacimiento="21/09/1947">Stephen King</autor>
    <fechaPublicacion año="2021"/>
  </libro>
</biblioteca>
```

Veamos en primer lugar el caso en el que no exista plantilla o esté vacía. Si no hay plantilla, el procesador recorre todos los nodos y extrae el texto contenido en cada nodo.

En el siguiente ejemplo, el resultado incluye el contenido de los nodos <título> y <autor>, ya que no hay ninguna plantilla.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Eduardo Mendoza Pequeña historia de la transición Victoria Prego Después Stephen King</pre>

Tabla 1. Ejemplo plantilla inexistente.

Si existe plantilla pero está vacía, el procesador no genera ningún resultado en el documento final ni recorre los nodos hijos.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="autor"> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Pequeña historia de la transición Después</pre>

Tabla 2. Ejemplo plantilla vacía.

En este caso, el resultado incluye el contenido de los nodos <titulo>, ya que no hay regla para ellos, pero los de <autor> se pierden porque la plantilla está vacía.

Si una plantilla vacía se aplica al nodo raíz, el procesador no genera ningún resultado en el documento final ni recorre ningún nodo hijo. Cuando existe más de una plantilla (en el ejemplo siguiente hay dos) en el documento XSL hay que tener en cuenta que, si ambas actúan sobre el mismo nodo, el procesador solo aplica una regla a cada nodo, la última.

XSLT	Resultado aplicación plantilla
<pre> <?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="libro"> <p><xsl:value-of select="autor"/></p> </xsl:template> <xsl:template match="libro"> <p><xsl:value-of select="titulo"/></p> </xsl:template> </xsl:stylesheet> </pre>	<pre> <?xml version="1.0" encoding="UTF-8"?> <p>Transbordo en Moscú</p> <p>Pequeña historia de la transición</p> <p>Después</p> </pre>

Tabla 3. Ejemplo de varias plantillas.

Observa que se ha generado texto adicional, como son los párrafos.

La instrucción <xsl:apply-templates> permite que se apliquen a los subelementos o subnodos las reglas que les correspondan. La figura siguiente muestra la estructura sintáctica del uso de apply-templates.

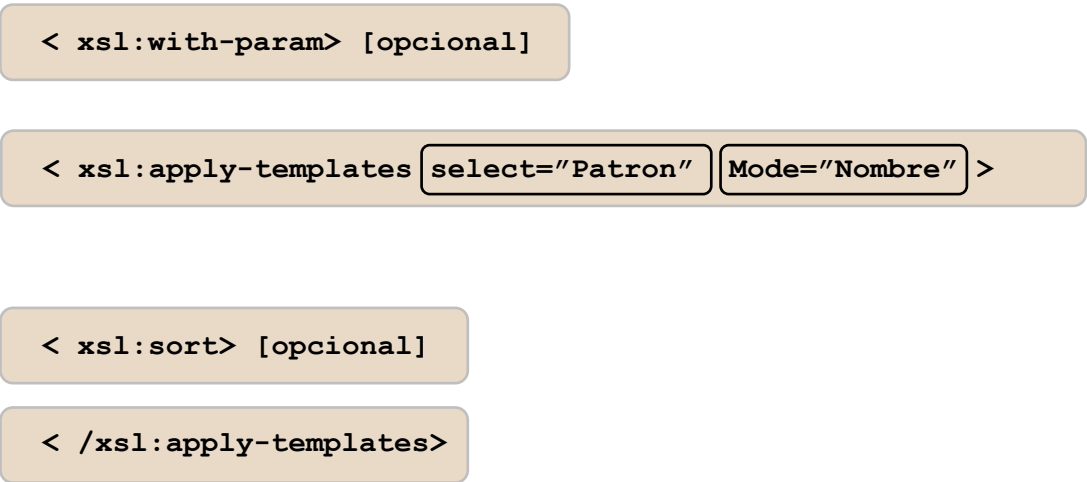


Figura 2. Sintaxis de apply-templates.

En el ejemplo siguiente se genera la etiqueta `<html>`, además de unos párrafos con los nombres de los autores.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="/"> <html> <h1>Autores</h1> <xsl:apply-templates /> </html> </xsl:template> <xsl:template match="libro"> <p><xsl:value-of select="autor"/></p> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <html><h1>Autores</h1> <p>Eduardo Mendoza</p> <p>Victoria Prego</p> <p>Stephen King</p> </html></pre>

Tabla 4. Ejemplo de uso de `apply-templates`.

Este ejemplo sí que se puede ver en el navegador, ya que se interpreta como HTML.



IMPORTANTE

Al transformar un documento, los procesadores XSLT incorporan saltos de línea y espacios en blanco en el resultado, pero no lo hacen de manera uniforme.

3.3. Instrucciones

Los elementos hijos del elemento raíz son los llamados elementos de nivel superior, que en realidad son estructuras cuyo contenido son instrucciones. Ya se ha trabajado con alguno de ellos. Ahora seguimos con los más importantes:

A. `xsl:value-of`

Calcula el valor de una expresión XPath dada y lo inserta en el árbol de resultados del documento de salida.

En XSLT podemos “filtrar” o indicar qué instancias de un elemento queremos procesar, tomando como criterio de selección el valor de los atributos que acompañan a los elementos.

Para hacer esto en un elemento `xsl:value-of`, podemos recuperar el valor de un atributo mediante la expresión `@nombreAtributo`. Por ejemplo:

```
<xsl:template match="vuelo">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
```

```

<td><xsl:value-of select="@origen" /></td>
<td><xsl:value-of select="@destino" /></td>
<td><xsl:value-of select="@hora" /></td>
</tr>
</xsl:template>

```

En el ejemplo siguiente, el documento final contiene los autores de los libros porque la plantilla los genera con la instrucción `<xsl:value-of>`. Como se ha aplicado una plantilla al nodo `<libro>`, sus hijos (`<titulo>`, `<autor>` y `<fechaPublicacion>`) no se recorren. Por eso los títulos de los libros no aparecen en el documento final.

XSLT	Resultado aplicación plantilla
<pre> <?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="libro"> <xsl:value-of select="autor"/> </xsl:template> </xsl:stylesheet> </pre>	<pre> <?xml version="1.0" encoding="UTF-8"?> Eduardo Mendoza Victoria Prego Stephen King </pre>

Tabla 5. Ejemplo a) de uso de `value-of`.

En este otro caso, los autores se obtienen a través de la regla que extrae el contenido del nodo (el "." hace referencia al propio elemento) y los títulos se obtienen porque, al no haber reglas para ese nodo, se extrae el contenido.

XSLT	Resultado aplicación plantilla
<pre> <?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="autor"> <xsl:value-of select="."/> </xsl:template> </xsl:stylesheet> </pre>	<pre> <?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Eduardo Mendoza Pequeña historia de la Transición Victoria Prego Después Stephen King </pre>

Tabla 6. Ejemplo b) de uso de `value-of`.

B. `xsl:attribute`

Añade un atributo en el documento resultado de la transformación. Lleva el atributo `name` con el nombre del atributo, y puede llevar un atributo `namespace` con la URL del espacio de nombres del cual procede el atributo.

Por ejemplo, para generar un atributo en una etiqueta `<img>` el siguiente código, incluido en una plantilla, la crea y el valor del atributo `src` sea el contenido de la etiqueta `<imagen>` existente en el archivo XML original.

```
<p>
  <img>
    <xsl:attribute name="src">
      <xsl:value-of select="imagen" />
    </xsl:attribute>
  </img>
</p>
```

C. `xsl:if`

Permite decidir si se va a procesar o no una parte del documento XSL en función de una condición. Lleva un atributo `test` que contiene la condición. Si la condición se cumple para el elemento que se está procesando, la regla se ejecutará.

```
<xsl:if test="@destino='Manises'">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
    <td><xsl:value-of select="@origen" /></td>
    <td><xsl:value-of select="@destino" /></td>
    <td><xsl:value-of select="@hora" /></td>
  </tr>
</xsl:if>
```

D. `xsl:choose`, `xsl:when` y `xsl:otherwise`

Indican qué transformación se debe realizar en el caso de que se cumpla una condición, y en el resto de casos. Se utilizan de forma conjunta. El elemento `xsl:choose` contendrá uno o más elementos `xsl:when` y un elemento `xsl:otherwise`.

El elemento `xsl:when` incluye un atributo `test` que tomará como valor la expresión que se evaluará. Si se cumple, se ejecutará el código escrito entre las etiquetas de inicio y de fin del elemento `xsl:when`.

El elemento `xsl:otherwise` contendrá el código que se ejecutará si no se cumplen las expresiones indicadas en los atributos `test` de los elementos `xsl:when`.

La estructura de trabajo conjunta de estos elementos es la siguiente:

```
<xsl:choose>
  <xsl:when test="expresión1">
    .....
  </xsl:when>
  <xsl:when test="expresión2">
    .....
  </xsl:when>
  <xsl:otherwise>
    .....
  </xsl:otherwise>
</xsl:choose>
```

E. xsl:import

Permite incluir la referencia a una hoja de estilos XSLT localizada en una URI dada. Debe llevar el elemento `href`, cuyo valor será dicha URI. Se debe incluir justo a continuación del elemento raíz y antes de cualquier otro elemento.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="hojaEstilo.xsl" />
```

F. xsl:output

Define el tipo de salida que se generará como resultado. En el ejemplo se genera un archivo XML.

```
<xsl:output method="xml"/>
```

G. xsl:sort

Las reglas se ejecutan en el orden en el que se van encontrando los elementos en el documento. Este comportamiento por defecto puede cambiarse en las hojas de estilo XSLT.

Para ordenar los contenidos se utiliza el elemento `xsl:sort`, que es un elemento hijo de `xsl:apply-templates` y puede llevar dos atributos:

- `select`: toma como valor el nombre del elemento que se va a utilizar como criterio de ordenación.
- `order`: indica si se debe utilizar un orden ascendente o descendente.

```
<xsl:apply-templates select="//aeropuerto ">
<xsl:sort select="aeropuerto" order="descending" />
</xsl:apply-templates>
```

- `xsl:strip-space` especifica cuáles son los elementos del documento XML que tienen espacios en blanco eliminados antes de la transformación.

H. xsl:variable

Se utiliza para declarar una variable. Las variables permiten hacer operaciones con los datos del documento XML para luego mostrar el resultado en el documento final. Cuando se asigna valor a la variable ya no se puede cambiar.

```
<xsl:variable name="var" select="15" />
```

3. XSLT

XSLT (Transformaciones XSL) es un lenguaje de programación declarativo que permite generar documentos en diferentes formatos a partir de documentos XML.

Algunas de sus principales características son las siguientes:

- El documento XML es el documento inicial al que se aplican las transformaciones.
- La hoja de estilo XSLT es el documento que contiene las reglas de transformación que se van a aplicar al documento inicial XML. Estas hojas de estilo son también archivos XML.
- El procesador XSLT es el programa que aplica al documento inicial XML las reglas de transformación incluidas en la hoja de estilo XSLT y genera el documento final.
- El resultado de la ejecución del programa es un nuevo documento (que puede ser un documento XML o no).

La figura siguiente ilustra este proceso.

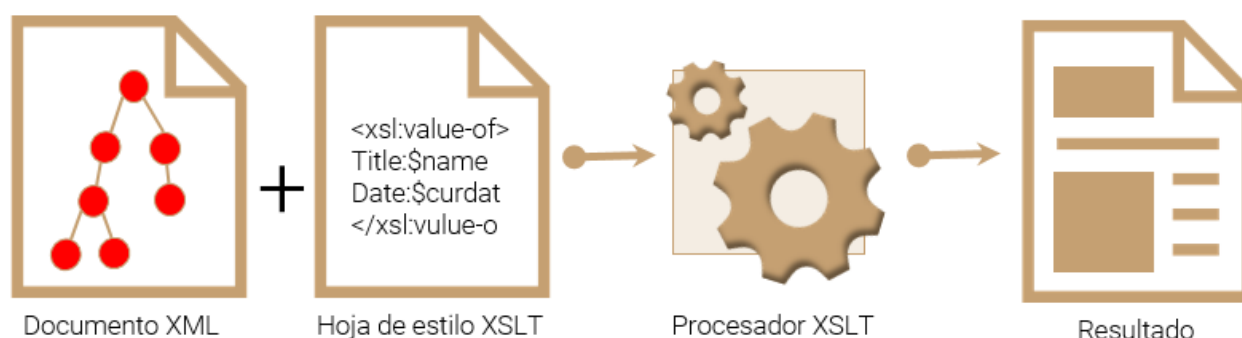


Figura 3.1. Proceso de transformación XSL.

El lenguaje XSLT está estandarizado por el W3C y ha publicado varias versiones, siendo la última la versión 2.0 de marzo de 2021, accesible desde este [enlace](#).

3.1. Estructura de una hoja de estilo o plantilla

Una hoja de estilo o plantilla XSLT es también un documento XML y debe estar bien formado. Las hojas de estilo se guardan siempre en archivos independientes con extensión .xsl, y deben comenzar con la declaración XML:

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

Los elementos básicos que puede contener una plantilla son los siguientes:

- **Elementos XSLT:** están precedidos por el prefijo xsl:, pertenecen al espacio de nombres xsl, están definidos en el estándar del lenguaje y son interpretados por cualquier procesador XSLT.
- **Elementos LRE (elemento de resultado literal):** son cualquier elemento que no sea instrucción. No pertenecen a XSLT, sino que se repiten en la salida sin más. En una plantilla todo lo que no empiece por 'xsl:' se considera un elemento literal y se transcribe exactamente a la salida.

El elemento raíz de la hoja de estilo XSLT es `stylesheet` (también puede utilizarse `transform`, ambas son equivalentes). Este elemento contendrá todos los demás, debe ir precedido por el prefijo `xsl` y el correspondiente al espacio de nombres para hojas de estilo XSLT. Los nombres de los elementos XSLT proceden de un mismo espacio de nombres, deben escribirse precedidos por el prefijo `xmlns` y la URL: `http://www.w3.org/1999/XSL/Transform`.

El elemento raíz queda así:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
.....
</xsl:stylesheet>
```

Entre las marcas de inicio y fin del elemento raíz `xsl:stylesheet` se escriben las reglas de transformación definidas mediante el elemento `xsl:template`. Cada regla indica qué instancias de los elementos del documento XML se van a transformar y cómo lo hará cada una de ellas.



EJEMPLO 4

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="//nombre">
<h2>
<xsl:value-of select="." />
</h2>
</xsl:template>
</xsl:stylesheet>
```

En este ejemplo, la regla se aplicará a todas las instancias del elemento *nombre*. Esto se indica mediante el atributo `match` que acompaña al elemento `xsl:template`. Entre las etiquetas de inicio y fin del elemento `xsl:template` se escribe la transformación que se debe realizar, es decir, qué texto y qué marcas se escribirán en el documento resultado de la transformación cada vez que se encuentre una instancia del elemento *nombre* en el documento origen.

`<xsl:value-of...>` recupera y escribe en el documento resultado el valor del elemento que está siendo procesado. Se estudia en el siguiente subapartado.

Dentro del elemento raíz `<xsl:stylesheet>`, además de las plantillas, se pueden encontrar los llamados elementos de alto nivel, como `<xsl:output>`



EJEMPLO 5

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="xml"/>
  <xsl:template match="/">
  </xsl:template>
</xsl:stylesheet>
```

En este código encontramos:

- El elemento de alto nivel `<xsl:output>` indica el tipo de salida producida.
- La instrucción `<xsl:template>` es una plantilla.
 - El atributo `match` indica los elementos afectados por la plantilla y contiene una expresión XPath.
 - El contenido de la instrucción define la transformación a aplicar (si la instrucción no contiene nada, como en el ejemplo anterior, sustituirá el nodo por nada, es decir, eliminará el nodo, aunque conservará el texto contenido en el elemento).

Para asociar un archivo de plantilla XSLT a un documento XML hay que añadir, después del prólogo, la línea siguiente:

```
<?xml-stylesheet type="text/xsl" href="ruta_del_archivo_xsl.xsl"?>
```

Con ella se indica al procesador el lugar donde se encuentra la hoja de estilos con la que dar formato a los datos del documento XML.

3.2. Utilización de plantillas. Plantillas múltiples

El elemento `xsl:template` permite controlar el formato de salida que se aplica a ciertos datos de entrada. Dentro de la plantilla tenemos las instrucciones, que son también elementos XSL.



IMPORTANTE

Para probar los ejemplos no es necesario abrir los archivos en el navegador. Basta con comprobar el resultado en el editor que se esté utilizando.

Si se utiliza el navegador web hay que tener en cuenta que los navegadores no respetan los saltos de línea ni los espacios en blanco, ni muestran las etiquetas.

Retomamos el código del Ejemplo 1.

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <titulo>Transbordo en Moscú</titulo>
    <autor>Eduardo Mendoza</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pequeña historia de la Transición</titulo>
    <autor fechaNacimiento="11/11/1948">Victoria Prego</autor>
    <fechaPublicacion año="2021"/>
  </libro>
  <libro>
    <titulo>Después</titulo>
    <autor fechaNacimiento="21/09/1947">Stephen King</autor>
    <fechaPublicacion año="2021"/>
  </libro>
</biblioteca>
```

Veamos en primer lugar el caso en el que no exista plantilla o esté vacía. Si no hay plantilla, el procesador recorre todos los nodos y extrae el texto contenido en cada nodo.

En el siguiente ejemplo, el resultado incluye el contenido de los nodos `<título>` y `<autor>`, ya que no hay ninguna plantilla.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Eduardo Mendoza Pequeña historia de la transición Victoria Prego Después Stephen King</pre>

Tabla 1. Ejemplo plantilla inexistente.

Si existe plantilla pero está vacía, el procesador no genera ningún resultado en el documento final ni recorre los nodos hijos.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="autor"> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Pequeña historia de la transición Después</pre>

Tabla 2. Ejemplo plantilla vacía.

En este caso, el resultado incluye el contenido de los nodos <titulo>, ya que no hay regla para ellos, pero los de <autor> se pierden porque la plantilla está vacía.

Si una plantilla vacía se aplica al nodo raíz, el procesador no genera ningún resultado en el documento final ni recorre ningún nodo hijo. Cuando existe más de una plantilla (en el ejemplo siguiente hay dos) en el documento XSL hay que tener en cuenta que, si ambas actúan sobre el mismo nodo, el procesador solo aplica una regla a cada nodo, la última.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="libro"> <p><xsl:value-of select="autor"/></p> </xsl:template> <xsl:template match="libro"> <p><xsl:value-of select="titulo"/></p> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <p>Transbordo en Moscú</p> <p>Pequeña historia de la transición</p> <p>Después</p></pre>

Tabla 3. Ejemplo de varias plantillas.

Observa que se ha generado texto adicional, como son los párrafos.

La instrucción `<xsl:apply-templates>` permite que se apliquen a los subelementos o subnodos las reglas que les correspondan. La figura siguiente muestra la estructura sintáctica del uso de `apply-templates`.

```
< xsl:with-param> [opcional]
```

```
< xsl:apply-templates select="Patron" Mode="Nombre" >
```

```
< xsl:sort> [opcional]
```

```
< /xsl:apply-templates>
```

Figura 1. Sintaxis de `apply-templates`.

En el ejemplo siguiente se genera la etiqueta `<html>`, además de unos párrafos con los nombres de los autores.

XSLT	Resultado aplicación plantilla
<pre> <?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="/"> <html> <h1>Autores</h1> <xsl:apply-templates /> </html> </xsl:template> <xsl:template match="libro"> <p><xsl:value-of select="autor"/></p> </xsl:template> </xsl:stylesheet> </pre>	<pre> <?xml version="1.0" encoding="UTF-8"?> <html><h1>Autores</h1> <p>Eduardo Mendoza</p> <p>Victoria Prego</p> <p>Stephen King</p> </html> </pre>

Tabla 4. Ejemplo de uso de *apply-templates*.

Este ejemplo sí que se puede ver en el navegador, ya que se interpreta como HTML.



IMPORTANTE

Al transformar un documento, los procesadores XSLT incorporan saltos de línea y espacios en blanco en el resultado, pero no lo hacen de manera uniforme.

3.3. Instrucciones

Los elementos hijos del elemento raíz son los llamados elementos de nivel superior, que en realidad son estructuras cuyo contenido son instrucciones. Ya se ha trabajado con alguno de ellos. Ahora seguimos con los más importantes:

A. `xsl:value-of`

Calcula el valor de una expresión XPath dada y lo inserta en el árbol de resultados del documento de salida.

En XSLT podemos “filtrar” o indicar qué instancias de un elemento queremos procesar, tomando como criterio de selección el valor de los atributos que acompañan a los elementos.

Para hacer esto en un elemento `xsl:value-of`, podemos recuperar el valor de un atributo mediante la expresión `@nombreAtributo`. Por ejemplo:

```
<xsl:template match="vuelo">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
    <td><xsl:value-of select="@origen" /></td>
    <td><xsl:value-of select="@destino" /></td>
    <td><xsl:value-of select="@hora" /></td>
  </tr>
</xsl:template>
```

En el ejemplo siguiente, el documento final contiene los autores de los libros porque la plantilla los genera con la instrucción `<xsl:value-of>`. Como se ha aplicado una plantilla al nodo `<libro>`, sus hijos (`<titulo>`, `<autor>` y `<fechaPublicacion>`) no se recorren. Por eso los títulos de los libros no aparecen en el documento final.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="libro"> <xsl:value-of select="autor"/> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Eduardo Mendoza Victoria Prego Stephen King</pre>

Tabla 1. Ejemplo a) de uso de `value-of`.

En este otro caso, los autores se obtienen a través de la regla que extrae el contenido del nodo (el "." hace referencia al propio elemento) y los títulos se obtienen porque, al no haber reglas para ese nodo, se extrae el contenido.

XSLT	Resultado aplicación plantilla
------	--------------------------------

<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="autor"> <xsl:value-of select="."/> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> Transbordo en Moscú Eduardo Mendoza Pequeña historia de la Transición Victoria Prego Después Stephen King</pre>
--	--

Tabla 2. Ejemplo b) de uso de value-of.

B. xsl:attribute

Añade un atributo en el documento resultado de la transformación. Lleva el atributo `name` con el nombre del atributo, y puede llevar un atributo `namespace` con la URL del espacio de nombres del cual procede el atributo.

Por ejemplo, para generar un atributo en una etiqueta `<img>` el siguiente código, incluido en una plantilla, la crea y el valor del atributo `src` sea el contenido de la etiqueta `<imagen>` existente en el archivo XML original.

```
<p>
  <img>
    <xsl:attribute name="src">
      <xsl:value-of select="imagen" />
    </xsl:attribute>
  </img>
</p>
```

C. xsl:if

Permite decidir si se va a procesar o no una parte del documento XSL en función de una condición. Lleva un atributo `test` que contiene la condición. Si la condición se cumple para el elemento que se está procesando, la regla se ejecutará.

```
<xsl:if test="@destino='Manises'">
  <tr>
    <td><xsl:value-of select="@numero" /></td>
    <td><xsl:value-of select="@origen" /></td>
    <td><xsl:value-of select="@destino" /></td>
    <td><xsl:value-of select="@hora" /></td>
  </tr>
</xsl:if>
```

D. xsl:choose, xsl:when y xsl:otherwise

Indican qué transformación se debe realizar en el caso de que se cumpla una condición, y en el resto

de casos. Se utilizan de forma conjunta. El elemento `xsl:choose` contendrá uno o más elementos `xsl:when` y un elemento `xsl:otherwise`.

El elemento `xsl:when` incluye un atributo `test` que tomará como valor la expresión que se evaluará. Si se cumple, se ejecutará el código escrito entre las etiquetas de inicio y de fin del elemento `xsl:when`.

El elemento `xsl:otherwise` contendrá el código que se ejecutará si no se cumplen las expresiones indicadas en los atributos `test` de los elementos `xsl:when`.

La estructura de trabajo conjunta de estos elementos es la siguiente:

```
<xsl:choose>
<xsl:when test="expresión1">
.....
</xsl:when>
<xsl:when test="expresión2">
.....
</xsl:when>
<xsl:otherwise>
.....
</xsl:otherwise>
</xsl:choose>
```

E. xsl:import

Permite incluir la referencia a una hoja de estilos XSLT localizada en una URI dada. Debe llevar el elemento `href`, cuyo valor será dicha URI. Se debe incluir justo a continuación del elemento raíz y antes de cualquier otro elemento.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="hojaEstilo.xsl" />
```

F. xsl:output

Define el tipo de salida que se generará como resultado. En el ejemplo se genera un archivo XML.

```
<xsl:output method="xml"/>
```

G. xsl:sort

Las reglas se ejecutan en el orden en el que se van encontrando los elementos en el documento. Este comportamiento por defecto puede cambiarse en las hojas de estilo XSLT.

Para ordenar los contenidos se utiliza el elemento `xsl:sort`, que es un elemento hijo de `xsl:apply-templates` y puede llevar dos atributos:

- `select`: toma como valor el nombre del elemento que se va a utilizar como criterio de ordenación.
- `order`: indica si se debe utilizar un orden ascendente o descendente.

```
<xsl:apply-templates select="//aeropuerto ">
```

```
<xsl:sort select="aeropuerto" order="descending" />
</xsl:apply-templates>
```

- `xsl:strip-space` especifica cuáles son los elementos del documento XML que tienen espacios en blanco eliminados antes de la transformación.

H. `xsl:variable`

Se utiliza para declarar una variable. Las variables permiten hacer operaciones con los datos del documento XML para luego mostrar el resultado en el documento final. Cuando se asigna valor a la variable ya no se puede cambiar.

```
<xsl:variable name="var" select="15" />
```

4. Procesadores XSLT. Depuración

Un procesador XSLT puede procesar los siete tipos de nodos presentes en un documento XML. El procesador XSLT toma como entrada el documento XML, y de él coge el contenido. Luego toma la hoja de estilos, con la estructura y las reglas de presentación, y genera un documento de salida aplicando dichas reglas sobre el contenido del documento XML al que afecta.

El procesador XSLT sigue la estructura jerárquica del documento XML y localiza cada nodo utilizando las instrucciones de la hoja de estilo. Esto lo hace nodo por nodo. Cuando el procesador encuentra una coincidencia, al contenido del elemento se le aplican las reglas de la plantilla.

El siguiente ejemplo de plantilla solo se aplica al nodo raíz (/) y en ella solo hay elementos literales. Cuando el procesador XSLT lee el documento XML, el primer nodo que se encuentra es el nodo raíz y con la regla dada hace 'match' (coincidencia) con el elemento raíz y muestra lo que vemos en el resultado.

XSLT	Resultado aplicación plantilla
<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999 /XSL/Transform" version="1.0"> <xsl:template match="/"> <html> <head> </head> <body> </body> </html> </xsl:template> </xsl:stylesheet></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <html> <head> </head> <body> </body> </html></pre>

Tabla 1. Ejemplo de uso del procesador XSLT.

El procesador XSLT es un software que lee un documento XML y un archivo XSLT, y crea un documento de salida aplicando las instrucciones de la hoja de estilos o plantilla XSLT a la información del documento XML.

Los procesadores XSLT pueden estar integrados en el propio navegador web. Es el caso de los motores de diseño de Safari y Chrome que utilizan la biblioteca libxslt (es una biblioteca gratuita publicada bajo licencia [MIT](#)) para realizar las transformaciones XSL, pero solo para la versión 1.0.

Otros procesadores XSLT son:

- [Saxon](#): es un procesador XSLT 3.0, con versiones patentadas y de código abierto, para operaciones con Java, JavaScript y .NET.
- [Xalan](#): es un procesador XSLT 1.0 de código abierto de la Apache Software Foundation disponible para Java y C++. Utilizado para la transformación de documentos XML en documentos HTML, texto u otro documento XML.

Muchos de los editores XML incluyen herramientas para el procesamiento de documentos XSLT. Por ejemplo, están:

- Liquid Studio: editor XML basado en Windows con editor DTD, un editor gráfico de XML-Schema, y editor de esquemas JSON.
- notepad++: editor de código fuente de texto con licencia GPL basado en C++. Dispone de complementos para la validación de XML, XSD y DTD, ya que verifica el formato y la sintaxis. Es compatible con la evaluación actual de la expresión XML Path y Xpath. Hay versiones para Windows y para GNU/Linux, pero en este caso hay que ejecutarlo sobre el emulador wine.
- XMLSpy: se comercializa principalmente como un editor JSON y XML. Incluye integración de Visual Studio Code y Eclipse. Tiene un constructor, validador, evaluador y depurador XPath.

En concreto, Visual Studio Code (herramienta que se está utilizando a lo largo del curso) dispone de un conjunto de extensiones para la depuración de archivos XSL y para realizar transformaciones.

También existen validadores online para las transformaciones, como por ejemplo el disponible en este enlace y otros que ya se han referenciado en otros apartados.

Por último, se explica la función de los depuradores XSLT.

El depurador XSLT permite recorrer la transformación paso a paso, pasando por los nodos, por las instrucciones, etcétera. El depurador resalta la instrucción actual del XSTL (que se está ejecutando) y el nodo actual del archivo XML, y genera una salida para cada uno de los pasos que aparece resaltada en el panel de resultados.

Muchos de los editores de XML incorporan un depurador XSLT que muestra las plantillas que se están ejecutando y sobre qué datos del archivo XML están actuando, así como la salida generada.

Por ejemplo:

- [Altova XML Spy](#) es de los más conocidos y potente. Disponible para Windows y para GNU/Linux a través de wine. Pero es una aplicación propietaria.
- [StylusStudio](#), muy similar al anterior.
- oXygenXML, basado en Java, y por tanto multiplataforma. También es una aplicación propietaria, pero se puede conseguir una descarga de prueba de 30 días gratuita.
- EditiX, otro editor/depurador XSLT muy sencillo para Windows, GNU/Linux y Mac OS.



Unidad 7.

Almacenamiento de información



**Mc
Graw
Hill**

1. Sistemas de almacenamiento de información

El conjunto de estos elementos físicos, lógicos, datos y personas constituyen lo que se conoce con el nombre de sistema de información, y su objetivo es transformar, mediante una serie de procesos, los datos en información, y de ella obtener conocimiento. Por lo tanto, no hay que confundir datos con información.

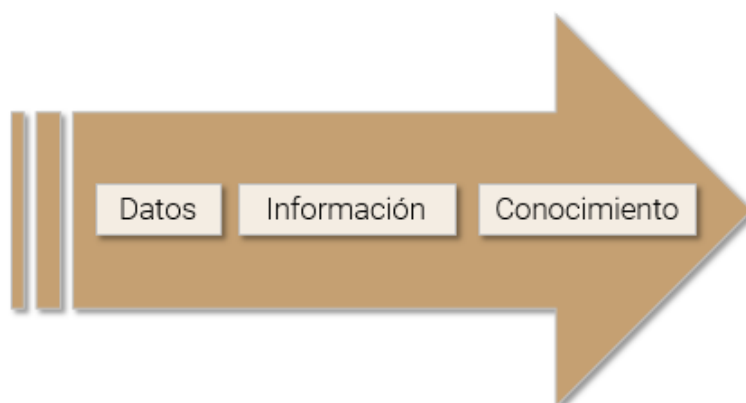


Figura 1.1. Transformación de los datos en conocimiento.

De ahí la importancia que tiene para la organización disponer de un buen sistema de información que proteja los datos, ya que son el elemento clave de todo el proceso.

Aquí es donde entra en juego el sistema de almacenamiento de los datos iniciales y de la información generada con ellos.

Una de las primeras soluciones para el almacenamiento de los datos fue el uso de archivos. Los archivos son almacenes de datos con una estructura determinada, y los llamados “programas de aplicación” accedían a estos datos, dando lugar a unos resultados. A su vez, los resultados se podían almacenar en otros archivos o se imprimían en papel. Estos resultados se pueden considerar información, ya que proporcionan conocimiento.

Esta forma de trabajar era costosa en tiempo y recursos, y poco eficiente. Los nuevos avances en las tecnologías de la información y la comunicación mejoran el escenario de trabajo con la aparición de los sistemas gestores de bases de datos (SGBD).

Con estas nuevas herramientas, las bases de datos, se implementan funcionalidades como son, por ejemplo, la posibilidad de hacer consultas y actualizaciones de los mismos datos por parte de usuarios diferentes; establecer relaciones entre diferentes archivos, tanto local como remotamente; eliminar la redundancia en los datos para mejorar su eficiencia y coherencia, etcétera.

Las bases de datos se definen como conjuntos de archivos relacionados entre sí, compartidos por diferentes procesos y usuarios simultáneamente, y con estructuras complejas.

En la actualidad, las bases de datos se han implantado como el mecanismo por excelencia de almacenamiento de datos, procesamiento de estos y obtención de información. Se utilizan bases de

datos para el almacenamiento de datos de cualquier naturaleza u origen: desde datos de clientes, proveedores, productos y empleados en una empresa hasta usos más personales y cercanos, como son el registro de gastos, consumos, libros, música, etcétera.

Con respecto a XML, el uso de las bases de datos permite integrar diferentes sistemas de información, los basados en archivos y los basados en el uso de bases de datos de tipo relacional, incluso orientadas a objetos.

- **Sistemas de información basados en documentos** (archivos): tienen estructura irregular, utilizan tipos de datos relativamente simples y en ellos el orden es importante.
- **Sistemas de información estructurados** (bases de datos relacionales): son relativamente planos, utilizan tipos de datos más complejos y en ellos el orden es poco importante.

Los desarrolladores, cuando se trata de diseñar bases de datos, normalmente deben decidir si los datos que se van a manejar se adaptan mejor al modelo XML, en algunos casos también conocido como NoSQL, o al modelo relacional o SQL. Siendo SQL el lenguaje de consulta más utilizado en las bases de datos relacionales.

Si partimos de un documento XML con su esquema asociado, veamos cuáles son sus semejanzas con una base de datos.

- XML utiliza uno o más documentos para almacenar la información.
- Define esquemas sobre la información.
- Tiene lenguajes de consulta específicos para recuperar la información requerida.
- Dispone de interfaces de programación o API, como son SAX (Simple API for XML) y DOM (Document Object Model).

Pero hay muchas diferencias que van a marcar definitivamente cuál de las dos tecnologías se va a utilizar. Estas son las principales diferencias entre el modelo XML y el modelo relacional:

- **Los datos XML son jerárquicos.** Un documento XML contiene información sobre la relación entre sus datos de forma jerárquica. En el modelo relacional, los únicos tipos de relaciones que pueden definirse son relaciones lógicas entre tablas.
- **Los datos XML son autodescriptivos.** Un documento XML contiene, además de los datos, su codificación. De esta forma, un solo documento puede contener diferentes tipos de datos. En el modelo relacional, el contenido de los datos se define en función de la columna en la que se encuentra. Todos los datos de una columna son de la misma tipología.
- **Los datos XML tienen un orden inherente.** En un documento XML se presupone que el orden de los elementos de datos es el orden del propio documento. En las bases de datos relacionales el orden de las filas no está garantizado, a no ser que se especifique.

2. Utilización de XML para el almacenamiento de información

¿Qué son las bases de datos XML?

Un documento XML es una base de datos porque:

1. Almacena información (documentos).
2. Responde a un esquema (DTD, XML Schema)
3. Tiene lenguajes de consulta (XPath, XQuery) y diferentes API de programación (SAX, DOM, JDOM, etcétera).

Ejemplos prácticos de posibles bases de datos XML:

- Archivo de configuración de una aplicación.
- Plantilla de un fax.
- Formulario para solicitar dietas de viajes.
- Temario de una asignatura.
- Todos los informes de un departamento.

En general, una base de datos XML es una base de datos almacenada o gestionada en forma de documentos XML.

¿Por qué surgen las bases de datos XML?

Por la necesidad de almacenar y recuperar datos poco estructurados con la eficiencia de las bases de datos convencionales.

Cuando se habla del almacenamiento de información utilizando XML existen dos enfoques o categorías diferenciadas de los documentos XML: un enfoque está centrado en los datos y el otro está centrado en los documentos.

2.1. Centrado en los datos:

- Este tipo de documentos, también llamados “data-centric”, suelen tener una estructura bien definida y sus datos pueden extraerse del documento e indexarse utilizando alguna base de datos convencional.
- Son documentos para el intercambio de datos (generalmente entre ordenadores). Los datos que se transmiten son como elementos atómicos bien definidos, por ese motivo estos datos se suelen llamar de “grano fino”.
- Cuando un documento XML tiene una estructura bien definida y contiene datos que pueden ser actualizados y usados de formas diferentes, el documento habitualmente está centrado en los datos.
- Suelen incluir documentos con datos científicos, órdenes de compra, facturación, publicidad, etcétera, que utilizan XML para el intercambio de información.

2.2. Centrado en los contenidos:

- Tienen una estructura irregular, lo que dificulta la ejecución de consultas, ya que estas se realizarán tanto del contenido como de la estructura del documento. De hecho, se llaman semiestructurados, ya que su formato no es muy estricto.
- Los datos se suelen llamar de “grano grueso”, ya que la unidad independiente más pequeña puede ser, incluso, un documento completo. Por ejemplo, libros o correos electrónicos.
- Contienen mucho texto e información.
- Enfocados a sistemas documentales y de gestión de contenidos.

De cualquier forma, esta clasificación de los documentos XML no es siempre sencilla y clara, y, a menudo, el contenido estará mezclado o será difícil catalogarlo dentro de un tipo u otro. Pero es una primera forma de determinar o descartar el sistema de almacenamiento XML a utilizar.

A. Almacenamiento de documentos XML

Veamos ahora qué tipos de almacenamiento permiten las bases de datos XML en general. En concreto, son tres:

1. **Almacenamiento no estructurado:** los documentos XML se almacenan directamente en formato de texto como atributo de tipo archivo y se deben proporcionar funciones adicionales para poder acceder a la información dentro de los documentos XML.
2. **Almacenamiento estructurado:** la estructura de un documento XML se convierte a un esquema de la base de datos que hay por debajo.
3. **Mapeo:** se establece una correspondencia entre los elementos XML y las estructuras de la base de datos. Si la estructura del documento XML no es compatible con la estructura de la base de datos, el documento debe ser transformado para ajustarlo a la estructura de la base de datos antes de almacenarlo.

El mapeo se puede hacer de dos formas:

Basado en tablas

- Modela el documento XML en un conjunto de tablas de un modelo relacional.
- Los atributos XML se almacenan en una columna de una tabla.
- Para recuperar un documento XML se debe realizar un proceso de serialización (conversión en una secuencia de bytes para su almacenamiento y/o transmisión).
- Distintos esquemas XML pueden generar el mismo esquema de BD relacional.

Basado en interrelaciones entre objetos.

- Cada tipo de elemento XML de más alto nivel se modela como una clase de objetos.
- Los atributos XML se modelan como atributos de las clases.

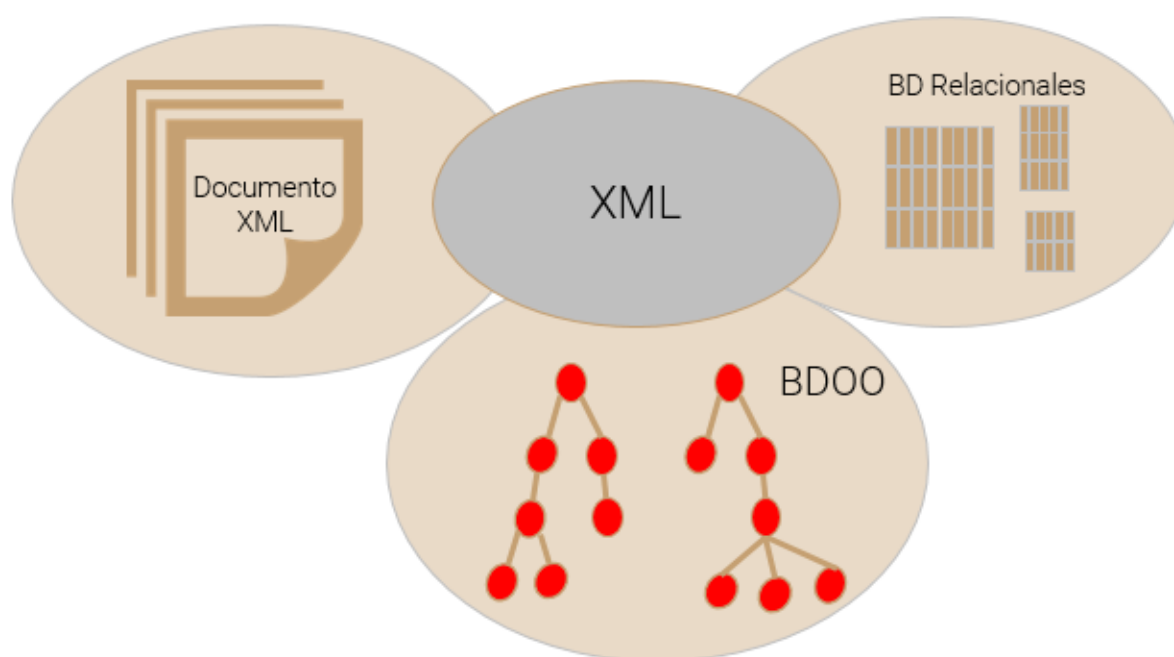


Figura 2.1. Mapeo de estructuras XML a diferentes tipos de bases de datos.

B. Tipos de bases de datos XML

Para clasificar los tipos de bases de datos XML que existen hay que tener en cuenta el tipo de almacenamiento y gestión que hacen de los documentos XML.

Se llaman bases de datos **XML-enabled** (bases de datos que permiten XML) aquellas bases de datos que mapean el esquema del documento en un esquema de base de datos y transfieren los datos según este mapeo.

Se llaman bases de datos **nativas XML** aquellas que utilizan un conjunto de estructuras fijas que permiten almacenar cualquier documento XML de forma original o nativa.

Las bases de datos llamadas XML-enabled suelen funcionar mejor cuando se necesita un enfoque basado en los datos.

Las bases de datos nativas XML funcionan mejor cuando el enfoque está centrado en el contenido.



IMPORTANTE

Ya se ha utilizado en numerosas ocasiones el parser o analizador XML, y es conocida su función como herramienta de validación del documento XML. Para ello, el parser se compone de módulos o programas que transforman el documento XML en una representación interna. Entre esos analizadores o parsers son especialmente importantes:

- SAX (Simple API for XML): se encarga de recorrer la estructura del documento generando eventos que corresponden a los elementos que se va encontrando.
- DOM (Document Object Model): representa el archivo en una estructura tipo árbol que usará para leer el documento.

Bases de datos XML-enabled

Utilizan el almacenamiento basado en el modelo, que consiste en definir un modelo de datos lógico (por ejemplo, DOM) para la estructura jerárquica de los documentos XML y almacenar el modelo binario del documento en un almacén existente o bien específico. En este caso, las posibilidades que existen son:

- Traducir el DOM a tablas relacionales como elementos, atributos, entidades, etcétera.
- Traducir el DOM a objetos en una base de datos orientada a objetos.
- Alguno de estos modelos son:
- Bases de datos relacionales: se basan en el uso de tablas bidimensionales (bases de datos relaciones) como único medio para representar los datos del mundo real.
- Bases de datos orientadas a objetos: soportan un modelo de objetos puro, no basado en extensiones del modelo relacional.

Se utilizan para publicar datos existentes, como XML, o importar datos desde un documento XML en una base de datos existente, no para almacenar documentos XML completos.

Requieren tiempo de diseño para el mapeo de esquemas.

No pueden almacenar documentos cuyo esquema no sea conocido en el tiempo de diseño.



C. XML y las bases de datos relacionales.

La base de datos relacional se basa, como su nombre indica, en las relaciones, y se puede definir como un conjunto de tablas interrelacionadas. Y una relación representa un conjunto de entidades con las mismas propiedades. Cada relación se compone de una serie de filas o registros (las llamadas tuplas), cuyos valores dependen de los atributos (columnas) definidos en el esquema relacional.

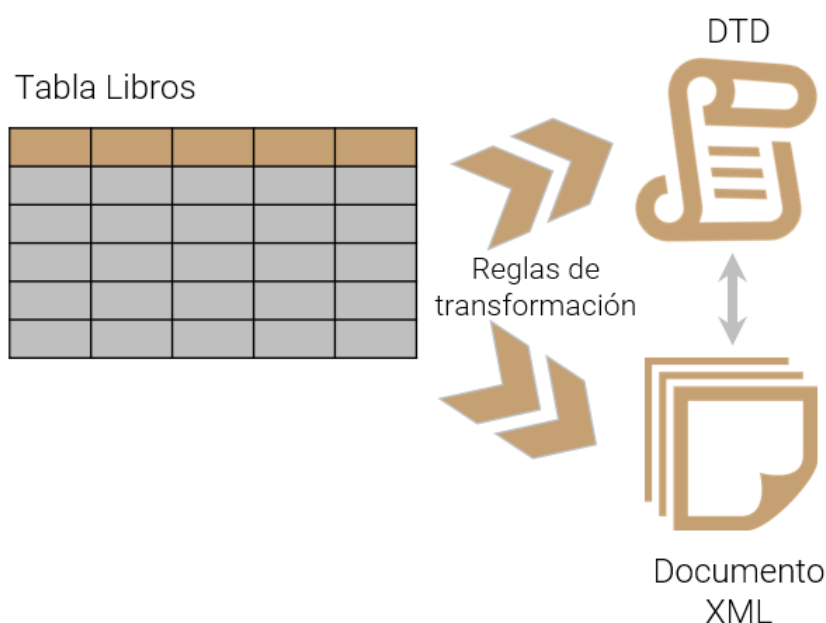


Figura 2.2. Transformación de BD relacional a XML.

Este tipo de bases de datos son una opción para el almacenamiento XML, pero no están optimizadas para almacenar estructuras de tipo jerárquico como son los documentos XML. Los motivos son los siguientes:

1. Las bases de datos relacionales tienen una estructura regular frente al carácter heterogéneo de los documentos XML.
2. Los documentos XML suelen contener muchos niveles de anidamiento, mientras que los datos relacionales son planos.
3. Los documentos XML tienen un orden intrínseco, mientras que los datos relacionales son no ordenados.
4. Los datos relacionales son generalmente densos (cada columna tiene un valor), mientras que los datos XML son dispersos, es decir, pueden representar la carencia de información mediante la ausencia del elemento.

La lógica de intercambio de información entre aplicaciones (Middleware) se encarga de la transferencia de información entre estructuras XML y bases de datos relacionales.

En general, esta lógica es un software que ayuda a una aplicación para interactuar o comunicarse con otras aplicaciones, redes, sistemas operativos o hardware, abstrayéndole de la complejidad de estas capas inferiores.

D. XML y las bases de datos noSQL

Constituyen una alternativa a las bases de datos que siguen el modelo entidad-relación.



Figura 2.3. Acrónimo NoSQL.

El detalle más significativo de ellas es el que indica su propio nombre: no utilizan el lenguaje de consulta SQL. Además, los datos no se almacenan en tablas, no soportan las operaciones típicas de JOIN (combinar filas de dos o más tablas), pero sin embargo son bases de datos que permiten el “escalado” horizontal de estas muy bien. El “escalado” horizontal de las bases de datos hace referencia a la posibilidad de mejorar la disponibilidad y el comportamiento de esta cuando se tenga que soportar una mayor carga de trabajo (más memoria, mayor capacidad de almacenamiento, mayor velocidad de proceso, etcétera).

Han tomado especial relevancia por ser las utilizadas por diferentes empresas como Google y Twitter, entre otras.

En realidad las bases de datos noSQL engloban varios tipos de bases de datos: bases de datos clave-valor, bases de datos documentales, bases de datos en grafo, bases de datos tabular, bases de datos multivalor y orientadas a objetos. Como se observa, el espectro abarcado por este tipo de bases de datos es muy amplio, pero de ellas las más conocidas son las documentales. La mayoría de las bases de datos XML son documentales.

Veamos cuáles son las ventajas que proporciona el uso de bases de datos noSQL:

- Dispone de diferentes tipos de bases de datos noSQL en función del tipo de proyecto.
- Permite el manejo de una gran cantidad de datos, dado que utiliza una estructura distribuida.
- Se ejecutan en máquinas con pocos recursos, no requieren de apenas computación.
- No genera cuellos de botella, en contraposición a los sistemas SQL, que pueden ralentizar el sistema por el mecanismo de análisis y ejecución de las diferentes sentencias.
- Escalabilidad horizontal, a base de incrementar el número de nodos disponibles.
- Permite almacenar estructuras de datos muy complejas con un rendimiento muy bueno.
- Se suelen utilizar cuando los datos son complejos.

Veamos cuáles son las desventajas de las bases de datos noSQL:

- El modelo relacional está muy extendido. Existen marcas comerciales y de software libre que lo respaldan. Este detalle es de gran importancia para las empresas, porque les da más confianza y seguridad.
- Estas bases de datos noSQL están desarrolladas por pequeñas empresas, y existe el temor a que puedan dejar el desarrollo y la actualización en cualquier momento.
- Existen limitaciones en el uso de herramientas de creación de informes.

3. Lenguajes de consulta y manipulación en documentos

Sin embargo, aunque XQuery y SQL puedan considerarse similares, el modelo de datos de XQuery es muy distinto del modelo de datos relacional en el que se basa SQL. La razón es que XML incluye conceptos como el de jerarquía y orden de los datos, que no están incluidos en el modelo relacional.

Por ejemplo, el orden de los datos. En XQuery es muy importante y determinante el orden en que se encuentren los datos, ya que no es lo mismo buscar una etiqueta `nombre` dentro de una etiqueta `autor` que todas las etiquetas `nombre` del documento (que pueden estar anidadas dentro de una etiqueta `autor` o fuera).

La definición de XQuery, tal como figura en su [documentación oficial](#), es la siguiente:

"XQuery es un lenguaje estandarizado para combinar documentos, bases de datos, páginas web y casi cualquier otra cosa. Está muy ampliamente implementado. Es poderoso y fácil de aprender."

La última versión disponible, [XQuery 3.1](#), es de marzo de 2017. Hay que tener en cuenta que el grupo de trabajo XQuery del W3C fue cerrado en octubre de 2018.

Las principales características de XQuery son las siguientes:

- Está basado en el lenguaje XPath, que se estudió en la unidad anterior, y lo utiliza para navegar a través de documentos XML.
- Es un lenguaje declarativo, lo que significa que, en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL.
- XQuery es compatible con todas las principales bases de datos.

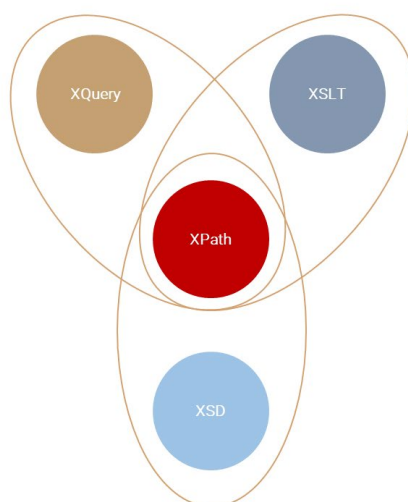


Figura 3.1. Relación entre tecnologías XML

Veamos ahora cuáles son los requerimientos técnicos de XQuery más relevantes:

- Debe ser independiente del protocolo de acceso a la colección de datos utilizado. Esto significa que la consulta XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos o al consultar un archivo XML en un servidor web.

- Las consultas y los resultados deben respetar el modelo de datos XML.
- Las consultas y los resultados deben ofrecer soporte para los espacios de nombres.
- Debe soportar las DTD y los XML-Schema, pero también debe ser capaz de trabajar sin ellos.
- Debe funcionar sin conocer la estructura del documento y, por lo tanto, independiente de ella.

3.1. Aplicaciones

Las principales aplicaciones de XQuery son las siguientes:

1. Recuperar información a partir de conjuntos de datos XML.
2. Transformar unas estructuras de datos XML en otras estructuras que organizan la información de forma diferente.
3. Ofrecer una alternativa a XSLT para realizar transformaciones de datos en XML a otro tipo de representaciones, como HTML o PDF.

Ventajas del uso de XQuery:

- Se pueden recuperar datos jerárquicos y tabulares.
- Se puede usar para consultar estructuras de árbol y gráficas.
- Se puede usar directamente para consultar páginas web.
- Se puede usar directamente para crear páginas web.
- Se puede usar para transformar documentos XML.
- Es ideal para bases de datos basadas en XML.

Los motores XQuery más importantes son:

- **Qexo:** escrito en Java y con licencia GPL que se distribuye integrado dentro del paquete Kawa.
- **Saxon:** escrito en Java y distribuido en dos paquetes:
 - Saxon-B: es software libre bajo licencia GPL.
 - Saxon-SA: contiene un procesador completo XSLT y XQuery con licencia propietaria, pero permite la evaluación de 30 días.
- **Qizx/open:** escrito en Java con todas las características del lenguaje excepto la importación y validación de XML-Schemas. Es uno de los motores con licencia GPL más completo que existe.

3.2. Modelo de datos.

El modelo de datos en que se sustenta XQuery es el modelo de datos de XPath.

XPath modela un documento XML como una estructura jerárquica en forma de árbol. El árbol está formado por nodos, y hay siete tipos de nodos: raíz, elemento, texto, atributo, espacio de nombres, instrucción de procesamiento y comentario.

Los principales nodos de la estructura jerárquica o en árbol en un documento XML son:

- **Nodo raíz o /:** es el primer nodo del documento.
- **Nodo elemento:** cualquier elemento de un documento XML. Cada nodo elemento posee un padre y puede o no tener hijos. En el caso de que no tenga hijos, es un nodo hoja.
- **Nodo texto:** cualquier elemento del documento que no esté marcado con una etiqueta de la DTD del documento XML.
- **Nodo atributo:** un nodo elemento puede tener etiquetas que complementen la información de ese elemento. Esto sería un nodo atributo.

Algunas reglas básicas de sintaxis:

- XQuery es sensible a mayúsculas.
- XQuery elementos, atributos y variables deben ser nombres XML válidos.
- Un valor de cadena XQuery puede estar entre comillas simples o dobles.
- Una variable XQuery se define con un \$ seguido de un nombre; por ejemplo, \$ librería.
- Los comentarios están delimitados por (: and :) , por ejemplo: (: XQuery Comment :)

Las principales características de este modelo de datos son:

- Se basa en la definición de **secuencia**, como una colección ordenada de cero o más ítems. Estas pueden ser heterogéneas, es decir, pueden contener varios tipos de nodos y valores atómicos. Sin embargo, una secuencia nunca puede ser un ítem de otra secuencia.
- Orden del documento: corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato XML.
- Contempla un valor especial llamado "error value", que es el resultado de evaluar una expresión que contiene un error.

3.3. Rutas de localización

La forma de localizar cada uno de los nodos del árbol XML es mediante una expresión XPath conocida como camino o ruta de localización.

Una ruta de localización:

- Selecciona un conjunto de nodos relativo al nodo de contexto.
- Puede contener recursivamente expresiones utilizadas para filtrar conjuntos de nodos.
- Al ser evaluado, devuelve el conjunto de nodos seleccionados por la ruta de localización.
- Se construye siguiendo unas reglas de sintaxis y semántica, vistas en la unidad 6.

Hay dos tipos de rutas de localización:

- **Rutas relativas:** son una secuencia de uno o más pasos de localización separados por /. Los pasos se componen de izquierda a derecha.
- **Rutas absolutas:** comienza con / seguido, opcionalmente, por una ruta de localización relativa. Una / por sí misma selecciona el nodo raíz del documento que contiene el nodo contextual.

Repasamos algunos ejemplos teóricos de rutas de localización:

- `libro` selecciona los elementos `libro` hijos del nodo contextual.
- `libro//titulo` selecciona los elementos `titulo` descendientes de los elementos `libro` hijos del nodo contextual.
- `*` selecciona todos los elementos hijos del nodo contextual.
- `@año` selecciona el atributo `año` del nodo contextual.
- `@*` selecciona todos los atributos del nodo contextual.
- `libro[1]` selecciona el primer hijo `libro` del nodo contextual.
- `libro[@año=2021]` selecciona todos los hijos `libro` del nodo contextual que tengan un atributo `año` con valor 2021.

Conceptos sobre consultas XQuery

Una consulta XQuery es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML, donde:

- Una secuencia es un conjunto ordenado de cero o más ítems.
- Un ítem es cualquier tipo de nodo del árbol XML o un valor atómico.

Las funciones que se pueden utilizar para referenciar colecciones y documentos dentro de la base de datos son las siguientes:

- `collection` (camino de la colección).
- `doc` (camino del documento).

Por ejemplo:

- La consulta `collection(/libros)`: devuelve el contenido de la colección de ruta absoluta `/libros`.
- La consulta `doc(/biblioteca.xml)`: devuelve el documento `/biblioteca.xml` completo.
- La consulta `doc(/biblioteca.xml)//nombre` devuelve todos los nodos `nombre` del documento `/biblioteca.xml`.
- La consulta `collection(/libros)//libro/titulo` devuelve los nodos `titulo` de todos los libros de la colección `/libros`.

3.4. Expresiones y cláusulas

Una consulta en XQuery es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML.

Los principales tipos de expresiones de XQuery son:

- Expresiones XPath para navegar por los documentos.
- Expresiones FLWOR (For, Let, Where, Order by, Return) para iterar por los elementos de un conjunto de datos.

Algunas características de las expresiones XQuery:

- La mayoría de las expresiones están compuestas por la combinación de expresiones más simples unidas mediante operadores y palabras reservadas.
- Al estudiar XPath se explicó que era un lenguaje declarativo utilizado para la localización de nodos en árboles XML. Esto implica que toda expresión XPath es también una consulta XQuery válida.
- Las expresiones a evaluar están delimitadas por las llaves { }.
- Las expresiones condicionales tipo if-then-else siguen la misma estructura y significado que en otros lenguajes de programación.

Una consulta XQuery está formada por dos partes:

- Prólogo: donde se declaran los espacios de nombres, de funciones, variables, etcétera.
- Expresión: es la consulta propiamente dicha.

Expresiones FLWOR

Las consultas XQuery pueden estar formadas por hasta cinco tipos de cláusulas diferentes y siguen la norma FLWOR (son las siglas de For, Let, Where, Order by y Return). Estas cláusulas son los bloques principales del XQuery y equivalen a las cláusulas de SQL: select, from, where, group by, having, order by y limit.

Las cláusulas FLWOR que pueden formar parte de una sentencia XQuery son las siguientes:

- **FOR**: permite seleccionar los nodos que se quieren consultar, guardando su valor en una variable (identificador que comienza por \$). Al conjunto de valores de la variable se le llama **tupla**. Si en la consulta aparece más de una cláusula FOR (o más de una variable en una cláusula FOR), el resultado es el producto cartesiano de dichas variables.
- **LET** (opcional): asocia valores a variables. Si esta cláusula aparece en una sentencia en la que ya hay al menos una cláusula FOR, los valores de la variable vinculada por la cláusula LET se añaden a cada una de las tuplas generadas por la cláusula FOR.
- **WHERE** (opcional): filtra tuplas producidas por las cláusulas FOR y LET, quedando solo aquellas que cumplen con la condición. Contiene una expresión que es evaluada para cada

tupla. Si su evaluación es false, esa tupla es descartada.

- **ORDER BY** (opcional): se ejecuta una vez para cada tupla generadas por FOR y LET después de que han sido filtradas por la cláusula WHERE. Es evaluada antes del return y los resultados son concatenados en una secuencia que es el resultado de FLWOR. Por defecto, el orden es ascendente, pero se puede usar el modificador *descending* para cambiar el sentido del orden.
- **RETURN**: construye el resultado de la expresión FLWOR para una tupla dada. Esta cláusula es ejecutada una vez para cada tupla retenida por la cláusula where. Los resultados de estas ejecuciones son concatenados en una secuencia que sirve como resultado de la expresión FLWOR

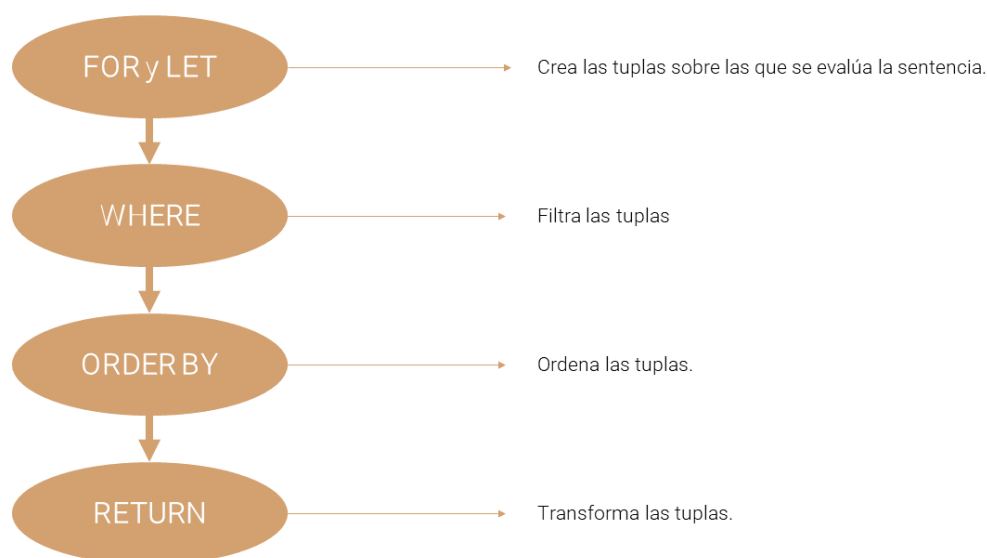


Figura 1.2. Expresión FLWOR. UD7_7.png.

Una expresión FLWOR vincula variables a valores con las cláusulas FOR y LET, y utiliza esos vínculos para crear nuevas estructuras de datos XML.

En una sentencia FLWOR al menos ha de existir una cláusula FOR o una LET; el resto, si existen, han de respetar escrupulosamente el orden dado por el nombre, FLWOR.



EJEMPLO 1

Para todos los ejemplos se toma como código de base el archivo XML gimnasio.xml (disponible en la sección [Ejemplos para trabajar y analizar](#), al inicio de esta misma unidad). Se trata de la oferta de diferentes disciplinas impartidas en un gimnasio.

Para hacer las pruebas se utiliza la herramienta Oxygen. Hay que crear una carpeta con el archivo XML gimnasio.xml y cada una de las consultas que se vayan generando. Luego crear un nuevo proyecto para estos ejemplos. Crear un nuevo escenario indicando los archivos implicados (el archivo XML y la consulta XQuery). Por último, aplicar la transformación XQuery. Las consultas serán archivos con extensión .xq. Los códigos de las diferentes consultas están disponibles para descargar también en la mencionada sección de archivos de los ejemplos, situada al inicio de la unidad.

```
<?xml version="1.0" encoding="utf-8"?>
<gimnasio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <disciplina id="1">
    <nombre>Pilates </nombre>
    <precio cuota="mensual" moneda="euro">35</precio>
    <plazas>20</plazas>
    <comienzo>1/10/2021</comienzo>
    <fin>1/07/2022</fin>
    <profesor>Profesor_Pilates</profesor>
    <sala>1</sala>
  </disciplina>
  <disciplina id="2">
    <nombre>Crossfit </nombre>
    <precio cuota="trimestral" moneda="euro">100</precio>
    <plazas>18</plazas>
    <comienzo>1/07/2021</comienzo>
    <fin>1/10/2021</fin>
    <profesor>Profesora_Crossfit</profesor>
    <sala>2</sala>
  </disciplina>
  <disciplina id="3">
    <nombre>BodyPump </nombre>
    <precio cuota="mensual" moneda="euro">30</precio>
    <plazas>15</plazas>
    <comienzo>1/01/2021</comienzo>
    <fin>1/12/2021</fin>
    <profesor>Profesor_BodyPump</profesor>
    <sala>1</sala>
  </disciplina>
  <disciplina id="4">
    <nombre>Zumba </nombre>
    <precio cuota="trimestral" moneda="dolares">80</precio>
    <plazas>12</plazas>
    <comienzo>1/01/2021</comienzo>
    <fin>1/12/2021</fin>
    <profesor>Profesora_Zumba</profesor>
```

```

        <sala>2</sala>
    </disciplina>
    <disciplina id="5">
        <nombre>Spinning </nombre>
        <precio cuota="mensual" moneda="dolares">40</precio>
        <plazas>10</plazas>
        <comienzo>1/10/2021</comienzo>
        <fin>1/07/2022</fin>
        <profesor>Profesor_Spinning</profesor>
        <sala>2</sala>
    </disciplina>
    <disciplina id="6">
        <nombre>Step </nombre>
        <precio cuota="anual" moneda="euro">300</precio>
        <plazas>10</plazas>
        <comienzo>1/01/2021</comienzo>
        <fin>31/12/2021</fin>
        <profesor>Porfesora_Zumba</profesor>
        <sala>1</sala>
    </disciplina>
</gimnasio>

```

a) Obtener el nombre de todas las disciplinas:

Consulta XQuery a aplicar:

```

for $a in doc("gimnasio.xml")//gimnasio/disciplina/nombre
return $a

```

Resultado obtenido:

Pilates Crossfit BodyPump Zumba Spinning Step

b) Obtener el nombre y precio de todas las disciplinas.

Consulta XQuery a aplicar:

```

for $b in doc("gimnasio.xml")//gimnasio/disciplina
return <li>{data($b/nombre)}. Precio: {data($b/precio)}</li>

```

Resultado obtenido:

Observar que se han añadido etiquetas que organizan la salida.

```

<?xml version="1.0" encoding="UTF-8"?>
<li>Pilates . Precio: 35</li>
<li>Crossfit . Precio: 100</li>
<li>BodyPump . Precio: 30</li>
<li>Zumba . Precio: 80</li>
<li>Spinning . Precio: 40</li>
<li>Step . Precio: 300</li>

```

c) Obtener el nombre y precio de todas las disciplinas cuyo precio es > 40.

Consulta XQuery a aplicar:

```
for $c in doc("gimnasio.xml")//gimnasio/disciplina
where $c/number(precio) > 40
return <li>{data($c/nombre)}</li>
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<li>Crossfit </li>
<li>Zumba </li>
<li>Step </li>
```

d) Obtener los profesores/as que den disciplinas con cuota mensual.

Consulta XQuery a aplicar:

```
for $d in doc("gimnasio.xml")//gimnasio/disciplina
where $d/precio/@cuota = "mensual"
return <li>{data($d/profesor)}</li>
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<li>Profesor_Pilates</li>
<li>Profesor_BodyPump</li>
<li>Profesor_Spinning </li>
```

3.5. Funciones

Ya se han estudiado algunas de las diferentes cláusulas de XQuery y cómo se utilizan. Estudiamos ahora las funciones soportadas por XQuery.

XQuery dispone de un conjunto de funciones propias, pero permite, además, la creación de nuevas funciones por parte del usuario. Están agrupadas en: funciones numéricas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana.

De ellas se van a estudiar las más importantes.

Funciones numéricas	Descripción
floor()	Devuelve el valor numérico inferior más próximo al dado.
ceiling()	Devuelve el valor numérico superior más próximo al dado.
round()	Redondea el valor dado al más próximo.
count()	Determina el número de ítems en una colección.
min() max()	Devuelven, respectivamente, el mínimo y el máximo de los valores de los nodos dados.
avg()	Calcula el valor medio de los valores dados.
sum()	Calcula la suma total de una cantidad de ítems dados.

Tabla 1. XQuery: funciones numéricas.

Funciones de cadena de texto	Descripción
concat()	Devuelve una cadena construida por la unión de dos cadenas dadas.
string-length()	Devuelve la cantidad de caracteres que forman una cadena.
starts-with() ends-with()	Determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
upper-case() lower-case()	Devuelven, respectivamente, la cadena dada en mayúsculas o minúsculas.

Tabla 2. XQuery: funciones de cadena de texto.

Funciones de uso general	Descripción
empty()	Devuelve "true" cuando la secuencia dada no contiene ningún elemento.
exists()	Devuelve "true" cuando una secuencia contiene, al menos, un elemento.
distinct-values()	Extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

Tabla 3. XQuery: funciones de uso general.

Cuantificadores existenciales	Descripción
Some every	Permiten definir consultas que devuelven alguno o todos los elementos que verifiquen la condición dada.

Tabla 4. XQuery: cuantificadores existenciales.

Si el usuario necesita crear una nueva función, la sintaxis es la siguiente:

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as tipo_dato2,...
$paramN as tipo_datoN)
as tipo_dato_devuelto
{
...CÓDIGO DE LA FUNCIÓN...
}
```

Una llamada a una función puede aparecer donde puede aparecer una expresión. Por ejemplo:

- En un elemento:
`<nombre>{upper-case($disciplina)}</nombre>`
- En el predicado de una expresión de ruta:
`doc("gimnasio.xml")/disciplina/nombre[stringlength($titulo)]`
- En una cláusula let: `let $var := (substring($nombre,1,3))`



EJEMPLO 2

Tomando como referencia el archivo del ejemplo 1, gimnasio.xml, vamos a hacer diferentes consultas.

Utilizar la herramienta Oxygen para realizar las pruebas siguiendo el procedimiento indicado en el Ejemplo 1.

a) Obtener cuántas disciplinas hay y añadiendo una etiqueta **total**.

Consulta XQuery a aplicar:

```
let $numDisciplina := count(/gimnasio/disciplina)
return <total>{$numDisciplina}</total>
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<total>6</total>
```

b) Obtener el nombre de la disciplina ordenados de forma descendente.

```
for $b in /gimnasio/disciplina
order by $b/nombre descending
return data($b/nombre)
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
Zumba Step Spinning Pilates Crossfit BodyPump
```

c) Obtener el nombre de las disciplinas de la sala 1, que se pague con euros y el precio sea menor a 40.

```
for $c in doc("gimnasio.xml")//gimnasio/disciplina
where $c/sala/text() = "1" and $c/precio/@moneda = "euro" and
$c/number(precio) < 40
return $c/nombre/text()
```

Resultado obtenido:

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

d) Obtener los profesores que impartan disciplinas con cuota mensual.

```
for $d in doc("gimnasio.xml")//gimnasio/disciplina
where $d/precio/@cuota = "mensual"
return <li>{data($d/profesor)}</li>
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<li>Profesor_Pilates </li>
<li>Profesor_BodyPump </li>
<li>Profesor_Spinning </li>
```

3.6. Operadores.

Las siguientes tablas muestran los operadores más importantes agrupados según su funcionalidad.

Comparación de valores: comparan dos valores escalares y producen un error si alguno de los operandos es una secuencia de longitud mayor de 1.

Operadores	Descripción
eq	Igual
ne	No igual
lt	Menor que
le	Menor o igual que
gt	Mayor que
ge	Mayor o igual que

Tabla 5. XQuery: operadores comparación de valores.

Comparación de generales: comparan operandos que sean secuencias.

Operadores	Descripción
=	Igual
!=	Distinto
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Tabla 6. XQuery: operadores comparación generales.

Comparación de nodos: comparan la identidad de dos nodos.

Operadores	Descripción
is	Devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
is not	Devuelve true si las dos variables no están ligadas al mismo nodo.

Tabla 7. XQuery: operadores comparación de nodos.

Secuencias de nodos: devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.

Operadores	Descripción
Unión	Devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
Intersect	Devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
Except	Devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el segundo.

Tabla 8. XQuery: operadores secuencia de nodos.

Otros operadores son:

Comparación de órdenes de los nodos: <<

Compara la posición de dos nodos. Devuelve "true" si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.

Lógicos: and y or

Se utilizan para combinar condiciones lógicas dentro de un predicado.

Aritméticos: +, -, *, div y mod

Devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados.



EJEMPLO 3

Tomando como referencia el archivo del ejemplo 1, gimnasio.xml, vamos a hacer diferentes consultas.

Utilizar la herramienta Oxygen para realizar las pruebas siguiendo el procedimiento indicado en el Ejemplo 1.

a) Mostrar las disciplinas cuyo precio sea menor que 100.

```
for $a in /gimnasio/disciplina
where $a/precio < 100
return data($a/nombre)
```

Resultado obtenido:

```
<?xml version="1.0" encoding="UTF-8"?>
Pilates  BodyPump  Zumba  Spinning
```

b) Intentamos sumar todos los precios de las diferentes disciplinas del gimnasio.

```
let $total := 0
for $b in //gimnasio/disciplina
let $total := $total + $b/precio
return $total
```

Resultado obtenido: observar que muestra cada uno de los precios pero no el total.

```
<?xml version="1.0" encoding="UTF-8"?>35 100 30 80 40 30
```

c) Mostrar el precio suma de todas las disciplinas. De esta forma sí que se obtiene la suma.

```
sum(//gimnasio/disciplina/precio)
```

Resultado obtenido

```
<?xml version="1.0" encoding="UTF-8"?>585
```

3.7. XQuery Update Facility.

El lenguaje XQuery solo proporciona expresiones para la realización de consultas sobre documentos XML, pero no su actualización (inserción, modificación o eliminación de nodos).

XQuery Update Facility es una extensión de XQuery que permite la actualización de documentos mediante las cláusulas insert, delete, replace y rename.

El funcionamiento de las cláusulas de XQuery Update es el siguiente:

Claúsulas	Descripción
insert	Permite la inserción de uno o varios nodos antes (before) o después (after) del nodo indicado. También se puede insertar al principio (as first into) o al final del documento (as last into).
Delete	Elimina uno o varios nodos del documento.
Replace	Tiene dos funciones: • Modificar el valor del nodo. • Modificar el nodo completo.
Rename	Renombra un nodo (elemento, atributo o instrucciones de proceso) sin afectar a su contenido.

Tabla 9. XQuery Update Facility. Cláusulas.

Por ejemplo, si se quiere eliminar la disciplina de id=2 en el documento gimnasio.xml habría que escribir:

```
delete node doc(gimnasio.xml)//disciplina[@id=2]
```

4. Almacenamiento XML nativo

Las bases de datos XML nativas son una herramienta que facilita el almacenamiento y la manipulación de los documentos XML.

Su unidad de almacenamiento lógico es el documento XML. Estos documentos se pueden almacenar y recuperar de manera intacta, es decir, con todos sus componentes, por lo que se deben tener en cuenta los elementos, los atributos, el texto y las secciones CDATA, además de preservar el orden en el documento XML.

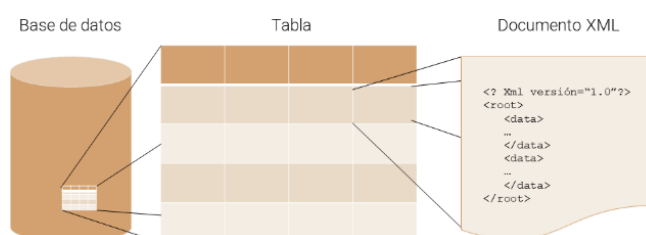


Figura 4.1. Almacenamiento BD nativas XML.

No requieren ningún modelo de almacenamiento físico determinado. Pueden construirse sobre bases de datos relacionales, jerárquicas u orientadas a objetos.

Las principales características de las BBDD XML nativas son:

- Utilizan la colección como elemento de agrupación de almacenamiento de documentos XML. La colección sería el equivalente a la tabla en las bases de datos relacionales. A su vez, las colecciones pueden contener otras colecciones.
- La agrupación en una colección se realiza en base a la información contenida en los documentos XML.
- Respecto a la validación, no siempre se realiza. Es decir, no necesitan un DTD o un XML Schema (W3C) para almacenar los documentos, es suficiente que los documentos XML estén bien formados.
- Pueden utilizar uno o varios lenguajes de consulta, de los cuales el más conocido es XQuery.
- Permite la creación de índices (Indexación XML) para acelerar la ejecución de las consultas.
- Soportan también la creación de identificadores únicos. De esta forma, a cada documento XML se le asocia un identificador único.
- Permiten las operaciones típicas sobre una base de datos: inserción, actualización y borrado. De ellas, la actualización Update XQuery es la más utilizada.
- Soportan transacciones, accesos concurrentes, control de accesos y backup, como cualquier otro sistema de bases de datos.

- Se define un modelo (lógico) para el documento XML, no para los datos.
- Respetan la estructura de documento, que es una estructura en árbol. Con este modelo es posible almacenar y recuperar el documento, tal y como fue insertado, de acuerdo a ese modelo.
- Almacenan los documentos XML en formato nativo. La unidad mínima de almacenamiento es un documento XML.
- No traducen el documento XML a estructuras relacionales u objetos.

El uso de las bases de datos nativas presenta las siguientes ventajas:

- No necesitan mapeos adicionales.
- Conservan la integridad de los documentos.
- Permiten almacenar documentos heterogéneos en la misma colección.
- Permite trabajar con grandes volúmenes de datos estructurados, semiestructurados y no estructurados que cambian a menudo.

Y suelen utilizarse:

- Documentos con anidamientos profundos.
- Cuando es importante preservar la integridad de los documentos.
- En sistemas con XML orientado a los documentos.
- Búsquedas de contenido.

Las principales áreas de aplicación son, entre otras:

- Portales de información corporativa.
- Catálogos de datos.
- Almacenamiento de información médica.
- BD de personalización.

Algunas bases de datos nativas XML son:

- **BaseX**: con licencia BSD. La página oficial es <http://basex.org>. Se estudia en el apartado 5.
- **MarkLogic**: con licencia propietaria. Basada en documentos. Es una base datos noSQL, digna competidora de Oracle. La página oficial es <https://www.marklogic.com/product/getting-started/>.
- **Sedna**: de código abierto. Especialmente diseñada para almacenar gran cantidad de datos XML. Almacenamiento y consultas muy ágiles.
- **eXist-db**: con licencia LGPL. La página oficial es <http://exist-db.org>. Multiplataforma. Con soporte para distintos lenguajes de consultas XML como XQuery, XPath y XSLT, indexación de documentos y soporte para la actualización de los datos.

5. Herramientas de tratamiento y almacenamiento de información en formato XML

BaseX es una base de datos XML nativa multiplataforma (GNU/Linux, Windows y Mac OS) y de software libre. Está incluida en el grupo de las llamadas bases de datos noSQL y es catalogada como base de datos documental.

Su sitio oficial es <https://basex.org>, está desarrollada por la Universidad de Konstanz (Alemania) y cuenta con el respaldo del W3C.

Está desarrollada en Java y dispone de procesadores XPath y XQuery. BaseX almacena las estructuras en árbol de los documentos XML utilizando tablas. La base de datos podrá almacenar un documento único o una colección de ellos. Dispone de varios tipos de índices que mejoran el rendimiento en diferentes operaciones, como son el seguimiento, las comparaciones de atributos de texto y búsquedas de texto completo, entre otras.

Con BaseX se pueden crear bases de datos y tablas, importar y exportar datos, hacer consultas y copias de seguridad con facilidad, entre otras funcionalidades.

En general, se puede decir que BaseX es una herramienta muy versátil, ligera, estable y compacta. Y, además, tiene una interfaz gráfica muy sencilla e intuitiva.

Una de las ventajas más importantes que ofrece BaseX es que puede tratar archivos XML aunque no sean estructurados o solo sean semiestructurados. No es indispensable que un nodo o atributo esté ubicado de la misma forma dentro de la base de datos, es decir, no es obligatorio que esté en la misma columna (como en las bases de datos relacionales). Se procesará igualmente. Esto facilita el análisis de los datos, ya que no requieren de un tratamiento previo.

BaseX es compatible con archivos XML, HTML, JSON y CSV, entre otros.

Sus principales características son:

- Gran rendimiento en el almacenamiento de texto, atributos y rutas. Con soporte para documentos XML de gran extensión.
- Cumple las recomendaciones del W3C XPath/XQuery.
- Arquitectura cliente/servidor, con soporte de transacciones seguras, gestión de usuarios y autenticación.
- Editor XQuery en tiempo real disponible, con resaltado de sintaxis y feedback de errores.
- Amplia gama de interfaces: REST/RESTXQ, WebDAV, XQJ, XML:DB.
- Permite buscar de forma sencilla, explorar y analizar datos, y evaluar expresiones XPath/XQuery en tiempo real.

La página de descargas es <https://basex.org/download/>. Seleccionar la opción correspondiente al sistema operativo y proceder a su instalación.

En el caso concreto de disponer de Ubuntu GNU/Linux, la herramienta está disponible en su pool de aplicaciones, y simplemente hay que seleccionarlo desde Ubuntu Software o ejecutar en la terminal la orden:

```
$sudo apt install basex
```


A continuación se muestra la interfaz gráfica de BaseX y se explican brevemente sus elementos más significativos.

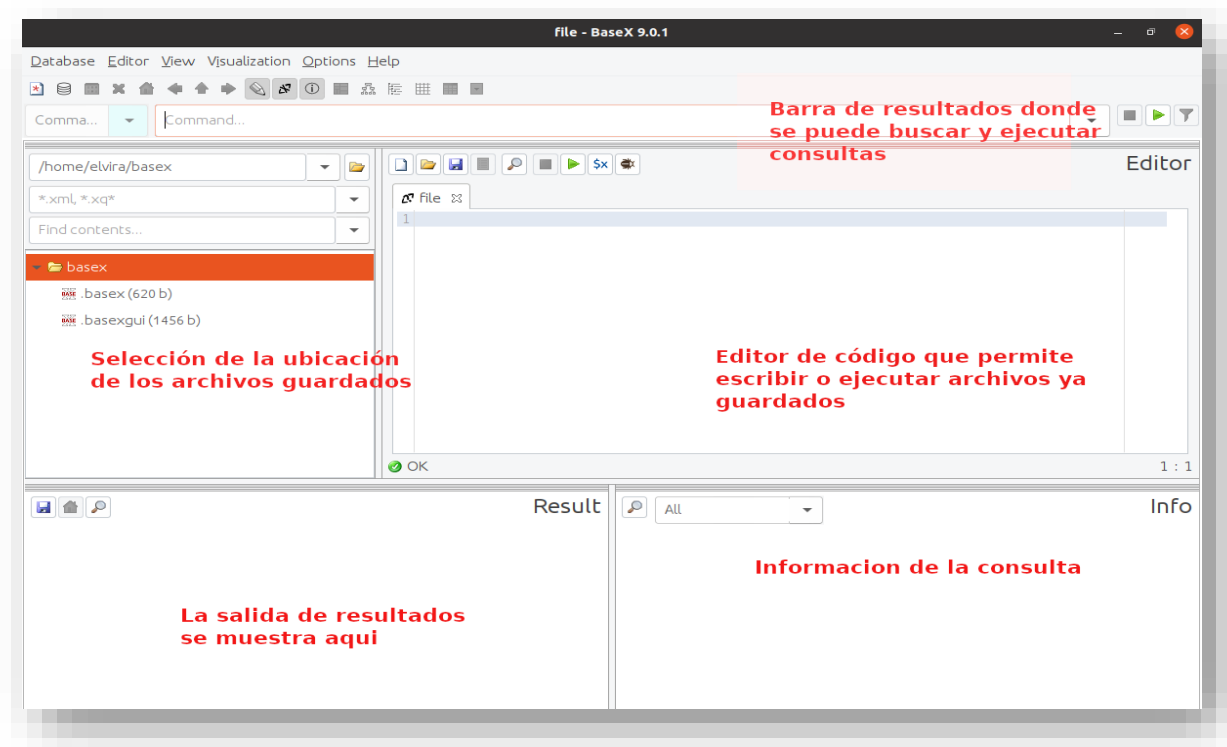


Figura 1. Interfaz de BaseX.

En la imagen se pueden ver las principales características de la interfaz gráfica de BaseX:

- En la parte superior está la barra de menús.
- Barra de herramientas (botones) en la parte inferior de la barra de menús.
- Línea de comando o barra de resultados en la parte inferior.
- Editor de consultas a la derecha.
- Diferentes pantallas de datos; específicamente en modo de árbol, modo de mapa o modo de texto. Se verán al ir trabajando con la herramienta.
- Información de la consulta.

Hay que comenzar creando una base de datos XML nativa. Para ello, ir a la opción de menú *Database* (Base de datos) y seleccionar *New* (Nueva). Muestra una ventana en la que se podrá localizar un archivo.

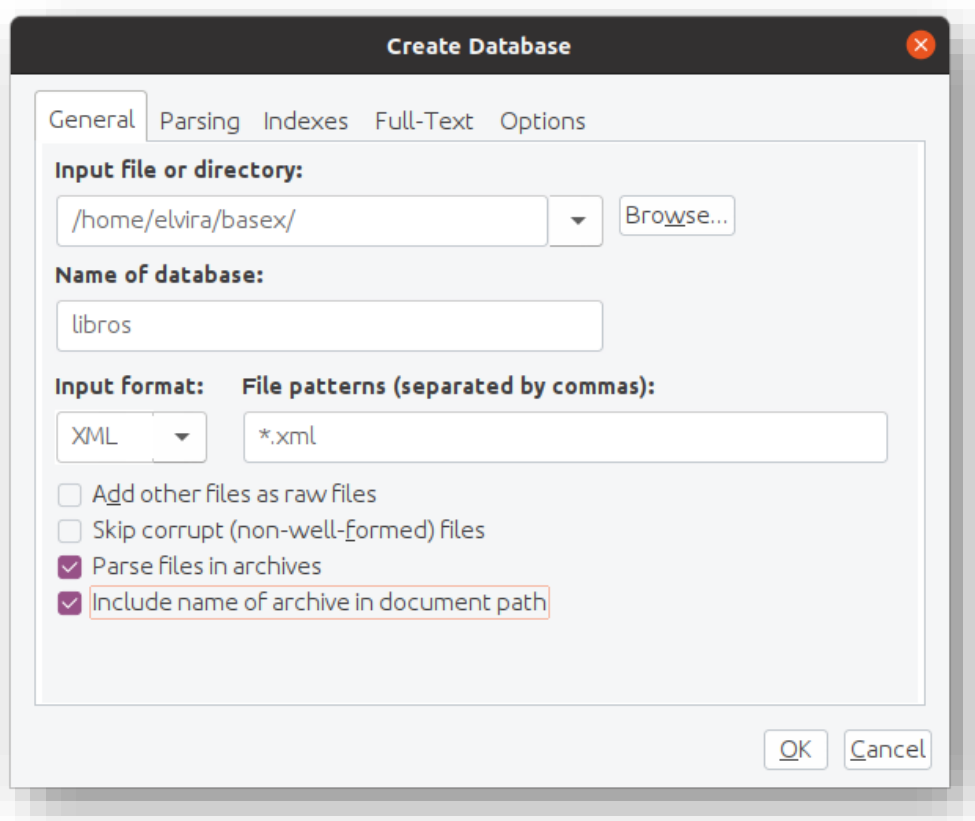


Figura 2. Creación de la base de datos.

Hay que dejar activada la opción *Parse files in archives* y marcar la opción *Include name of archive in document path*.

BaseX puede crear bases de datos en las que estén implicados uno o más archivos XML.

En la línea de comandos se pueden escribir comandos relacionados con la gestión de la base de datos, como son `CREATE DB`, `OPEN`, `CREATE INDEX`, `CREATE USER`, `ADD`, `DELETE`, `REPLACE`, `CLOSE`, `DROP DB`, etcétera.

En nuestro caso, la creación de la base de datos se podría haber hecho igualmente con `CREATE DB`, pero habría que haber proporcionado todos los datos para la creación (nombre de la base de datos y nombre del archivo XML implicado). Por defecto se ha almacenado en la carpeta personal del usuario, dentro de una nueva carpeta llamada `basex`.

Otras operaciones que permite BaseX es crear colecciones de documentos, borrar documentos, borrar colecciones, obtener información del servidor (listar bases de datos, documentos, usuarios, etcétera) y hacer copias de seguridad.



Unidad 8.

Sistemas de gestión empresarial



Mc
Graw
Hill

1. Aplicaciones de gestión empresarial. Tipos

La implantación y el uso de los sistemas de gestión empresarial ofrece muchas más ventajas a las empresas. Entre ellas, un sistema de gestión empresarial provee de:

- Almacenamiento de grandes cantidades de información de forma centralizada.
- Mecanismos para impedir la pérdida de información.
- Análisis de la información almacenada para realizar previsiones de futuro y tomar decisiones en consecuencia.
- Gestión de la empresa de forma centralizada, ya que cualquier miembro de la organización utilizará la misma herramienta para su trabajo diario.
- Automatización de procesos y tareas repetitivas.
- Mecanismos de adaptación al tamaño de la organización.
- Mecanismos para salvaguardar la integridad en los datos, que ayudará en el proceso de toma de decisiones.
- Mecanismos de generación de informes transparentes sobre cualquier área de la organización.

Y todas estas ventajas en realidad se pueden resumir en un ahorro tiempo y costes, así como en un aumento de la productividad.

Pero también existen algunos inconvenientes o desventajas. El primero de ellos es el precio. Normalmente, estas herramientas tienen un coste elevado, sobre todo cuando la organización requiere de una adaptación de la misma a sus necesidades. Es verdad que existen sistemas de gestión empresarial que cubren la mayor parte de las funciones que se necesitan, y también los hay especializados en diferentes ámbitos de trabajo. Pero aun así es muy normal solicitar adaptaciones a la empresa que comercializa o distribuye la herramienta.

En la actualidad ya no se realiza “programación a medida” de la organización, por su excesivo precio y porque los procesos internos cada vez más se tiende a normalizarlos, estandarizarlos. Pero incluso, aunque se decida utilizar software libre para cubrir estas necesidades, siempre hay módulos específicos que quedan fuera de la gratuidad, cuyo coste es elevado. El negocio de las empresas distribuidoras reside en la posibilidad de poder retocar el código libre y realizar estas adaptaciones.

Otra desventaja importante es el tiempo necesario para implementar estas adaptaciones funcionales y llevar a cabo el proceso de adiestramiento en su uso de todos los empleados de la organización. Abandonar las rutinas de trabajo establecidas durante años y cambiarlas a un sistema que lleva todo el control de la organización de forma digital y centralizada no es una tarea simple, de hoy para mañana. Es todo un proceso de transformación y adaptación muy importante que lleva su tiempo.

Tipos

Los principales tipos de aplicaciones para la gestión empresarial son los siguientes:

- **ERP** (Enterprise Resource Planning): constituye el software de gestión empresarial por excelencia. Este programa integra y maneja múltiples tareas relacionadas con la producción y la distribución.
- **CRM** (Customer Relationship Management): es otro gran sistema de gestión empresarial en la actualidad. Se trata de una herramienta enfocada a la gestión de las ventas y la atención a los clientes.
- **SRM** (Supplier Relationship Management): este software trabaja con la finalidad de mejorar el contacto con los proveedores y compartir una metodología de trabajo que afiance las relaciones comerciales.
- **BI** (*Business Intelligence*): la inteligencia empresarial es un software creado para estructurar en información útil todos los datos que genera una organización, ya sean provenientes de las transacciones que realice o de las campañas de marketing que lleve a cabo.

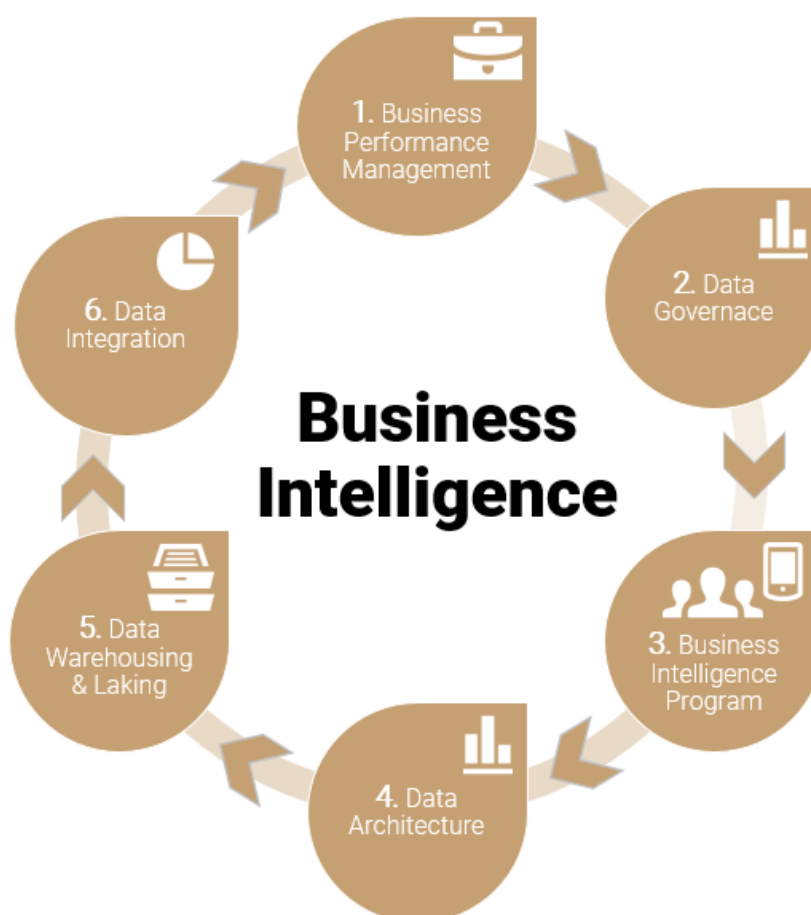


Figura 1.1. Inteligencia empresarial.

La elección entre uno u otro sistema dependerá de las necesidades de la empresa y de las áreas de actividad que tenga. En algunos casos, incluso, se puede implantar varias de ellas o pueden venir integradas. Por ejemplo, existen muchos ERP que integran su propio CRM.

2. Planificación de recursos empresariales (ERP)

Un ERP es un sistema de gestión de información que permite a las empresas controlar todos los flujos de información que se generan en cada ámbito de la organización y, en consecuencia, integran todos sus departamentos.

Una de las características más importantes que marcan la diferencia entre un ERP y cualquier otro software empresarial es que deben ser sistemas integrales, modulares y adaptables.

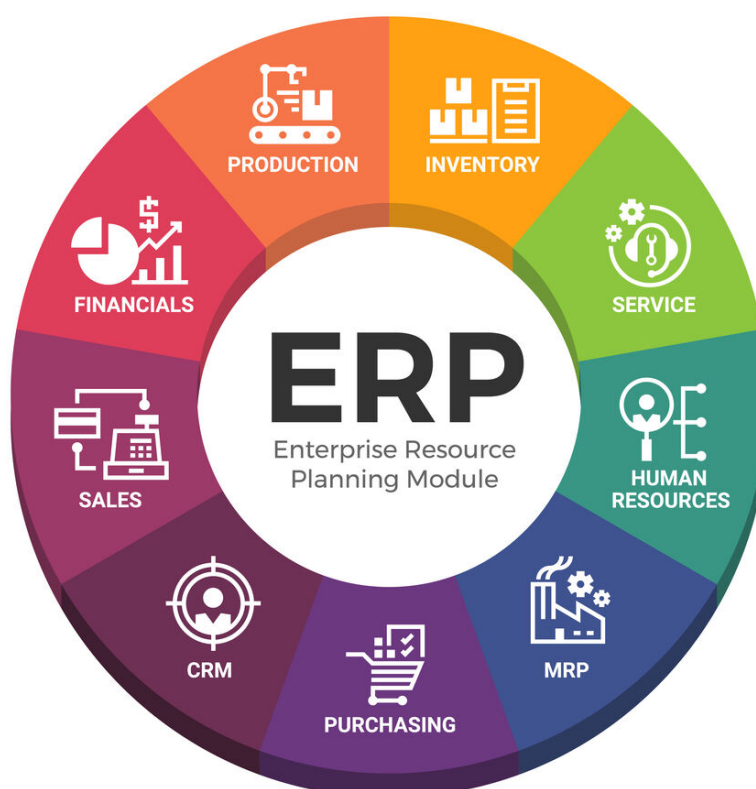


Figura 2.1. Diagrama de funcionalidades de un ERP.

Un sistema ERP tiene los siguientes objetivos:

- Optimizar los procesos empresariales.
- Acceder a la información de manera fiable y precisa.
- Permitir compartir información entre los departamentos o áreas de la organización.
- Eliminar datos y procesos innecesarios.

Además, un ERP se caracteriza por:

- Ser un programa o conjunto de programas con acceso a una base de datos.

- Sus componentes interactúan entre sí.
- Los datos deben ser consistentes y completos.

En la actualidad, y debido a la amplia implantación de las intranets en las empresas, la mayoría de los sistemas ERP tienen interfaz web, lo que aporta la ventaja de permitir el acceso al ERP a través del navegador web.

En líneas generales hemos dicho cuál es la utilidad de un ERP. Veamos ahora sus diferentes funcionalidades sabiendo que no todos los ERP las soportan, depende de las características de cada uno de ellos:

1. **Gestión de facturación:** puede personalizar y programar el envío de facturas a los clientes para las fechas indicadas.
2. **Gestión de albaranes:** permite gestionar albaranes de compra y venta, con todos los datos almacenados en la base de datos.
3. **Creación de presupuestos:** esta opción puede ser muy útil para empresas que ofrezcan servicios.
4. **Gestión de clientes y proveedores:** gestiona la lista de clientes y proveedores, con todos los datos de contacto.
5. **Gestión de compras:** las compras a proveedores pueden gestionarse desde un ERP, controlando los precios de compra o incluso gestionando la relación con los proveedores.
6. **Gestión de ventas:** controla las ventas de la empresa, qué productos se han vendido más o menos, cuánto se ha vendido cada mes, cuáles son los ingresos, etcétera.
7. **Contabilidad e impuestos:** hay ERP que permiten llevar el control fiscal de la empresa permitiendo cumplimentar los modelos de Hacienda y enviarlos directamente. Los datos se calculan a partir de las facturas y datos que se han incluido en el ERP.
8. **Control de inventario y almacén:** gestión completa de almacén, lotes, fechas de caducidad, etiquetas, control de stock, stock mínimo y máximo, etcétera.
9. **TPV (terminal punto de venta):** existen ERP que permiten disponer de un TPV para cobrar a los clientes, y los datos van directos a la base de datos.
10. **Tienda online:** sincroniza la tienda online con los productos: precios, stock, productos, facturas.
11. **CRM (Customer Relationship Management):** permite gestionar las relaciones con los clientes, realiza campañas de marketing vía correo electrónico y permite llevar un control eficaz de las comunicaciones y de la atención al cliente.
12. **Gestión del cuadro de mando:** en general, todos los ERP disponen de una herramienta de visualización que permite controlar los principales indicadores de rendimiento mediante gráficos e informes que ayudan a ver la evolución de la empresa.

Veamos ahora los tipos de ERP y sus diferentes criterios de clasificación. El dilema vendrá en el momento de decidir cuál es el más apropiado para la empresa.

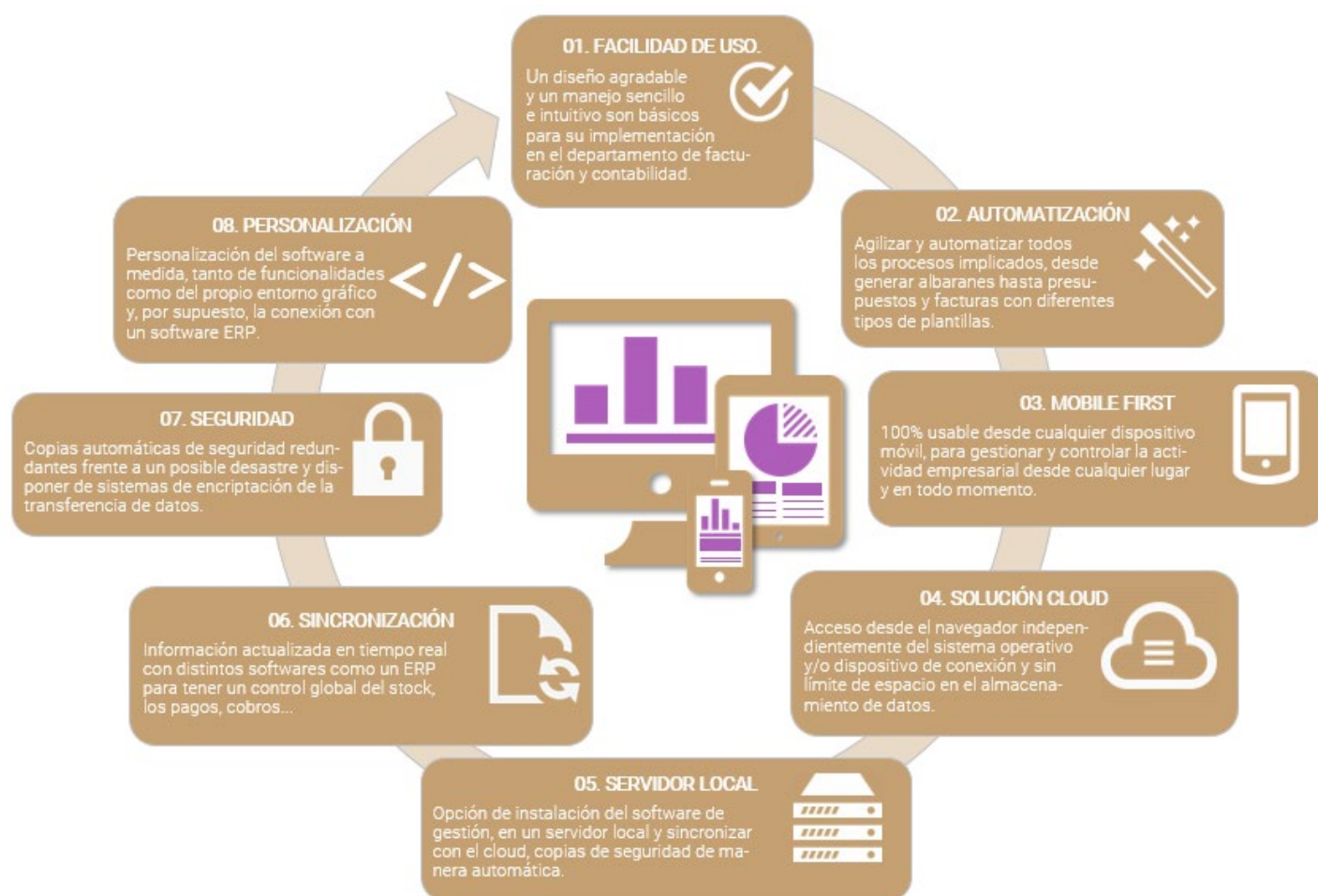


Figura 2.2. Factores decisivos para elegir un ERP.

Tipos de ERP

Los ERP se pueden clasificar atendiendo a diferentes criterios.

1. Clasificación por tamaño

El tamaño o volumen de la empresa influye en las necesidades de la misma. No es lo mismo atender a los requerimientos de una gran corporación que a los de una pyme. Son planteamientos diferentes que requieren soluciones diferentes.

Dentro de este grupo de ERP están:

- **ERP para autónomos, pequeñas y medianas empresas**

Los ERP diseñados para compañías pequeñas o medianas se caracterizan porque son útiles a lo largo de la vida de la empresa. Se adaptan a cada etapa de su crecimiento. Su coste es mínimo, para que cualquier pequeño empresario se lo pueda permitir desde el inicio de su actividad. Además, provee las funcionalidades para que sigan cubriendo sus necesidades en la medida que la empresa vaya creciendo.

Ejemplos: Odoo Community (Open Source), SAP Business One (para pymes de tamaño medio), Business Central (entornos Microsoft), eV4 (conectividad con tiendas online), entre muchos otros.

- **ERP para grandes empresas**

En este caso estamos hablando de ERP muy potentes, con una gran cantidad de módulos funcionales que abarcan todas las necesidades de empresas de tamaño considerable. Por supuesto, con un coste muy elevado y unos requerimientos de formación del personal que lo utiliza muy alto.

Ejemplos: Oracle Fusion Cloud, SAP S/4HANA, IFS ERP, etcétera.

2. Clasificación por instalación

- **ERP de escritorio local (on premise)**

Si la empresa decide disponer de un ERP en local eso implica que lo va a instalar en su propio servidor. Para ello requiere de espacio físico en las instalaciones de la empresa en el que poder ubicar el servidor de forma segura y que esté conectado con el resto de equipos informáticos.

Trabajar de esta forma tiene la ventaja de que es la misma empresa la que se encarga de preservar la seguridad, tanto de la máquina como de los datos almacenados, así como del acceso controlado a estos. Pero, por otro lado, la empresa necesita disponer de personal formado para garantizar el buen funcionamiento del servidor, la gestión de las copias de seguridad de los datos y la resolución de los problemas que pudieran surgir.

Ejemplos: Holded, webERP, Factusol, etcétera.

- **ERP en la nube (ERP cloud)**

Este tipo de ERP está alojado en la nube, entendiendo por la nube un conjunto de servidores (físicamente alojados en la Tierra) que proveen determinadas empresas especializadas en estos servicios. Por ejemplo, están Google Cloud Platform, Azure (Microsoft), Oracle Cloud, entre otros.

La principal ventaja que ofrece este tipo de alojamiento para un ERP es que la herramienta está disponible desde cualquier lugar y de forma segura. El proveedor del servicio en la nube es el que garantiza la disponibilidad y seguridad.

Otra ventaja es el mantenimiento, ya que esta responsabilidad recae también el proveedor. La empresa se libera de esta tarea.



Figura 2.3. ERP en la nube.

3. Clasificación por áreas de competencia

- ERP vertical

Estos ERP ofrecen funcionalidades muy concretas y se adaptan bien a las empresas que trabajan en áreas de negocio o sectores muy determinados. Es el caso, por ejemplo, de un ERP basado en software SAT. Este tipo de ERP tiene su ámbito de uso en empresas que ofrecen asistencia técnica, y sus funcionalidades están orientadas a la gestión de contratos de mantenimiento, avisos y órdenes de trabajo.

En este caso, los ERP son muy variados en función del sector productivo que abarque. Los hay para la gestión de hoteles, farmacias, fabricación de muebles, clínicas, transporte, comercio, etcétera. Para cada área existen varias marcas comerciales de ERP.

- ERP horizontal

Los ERP horizontales abarcan todas las necesidades comunes de las empresas (facturación, presupuestos, almacén, TPV, tienda online, etcétera). Es útil para empresas de cualquier sector, pero no incluyen funcionalidades específicas.

En algunos casos, los proveedores de la herramienta pueden adaptar un ERP horizontal a algunas necesidades específicas de la empresa. Siempre que el código sea de su propiedad o, en su defecto, su licencia lo permita.

Ejemplos: SAP Business One, Microsoft Dynamics, Netsuite Oracle, entre otros.

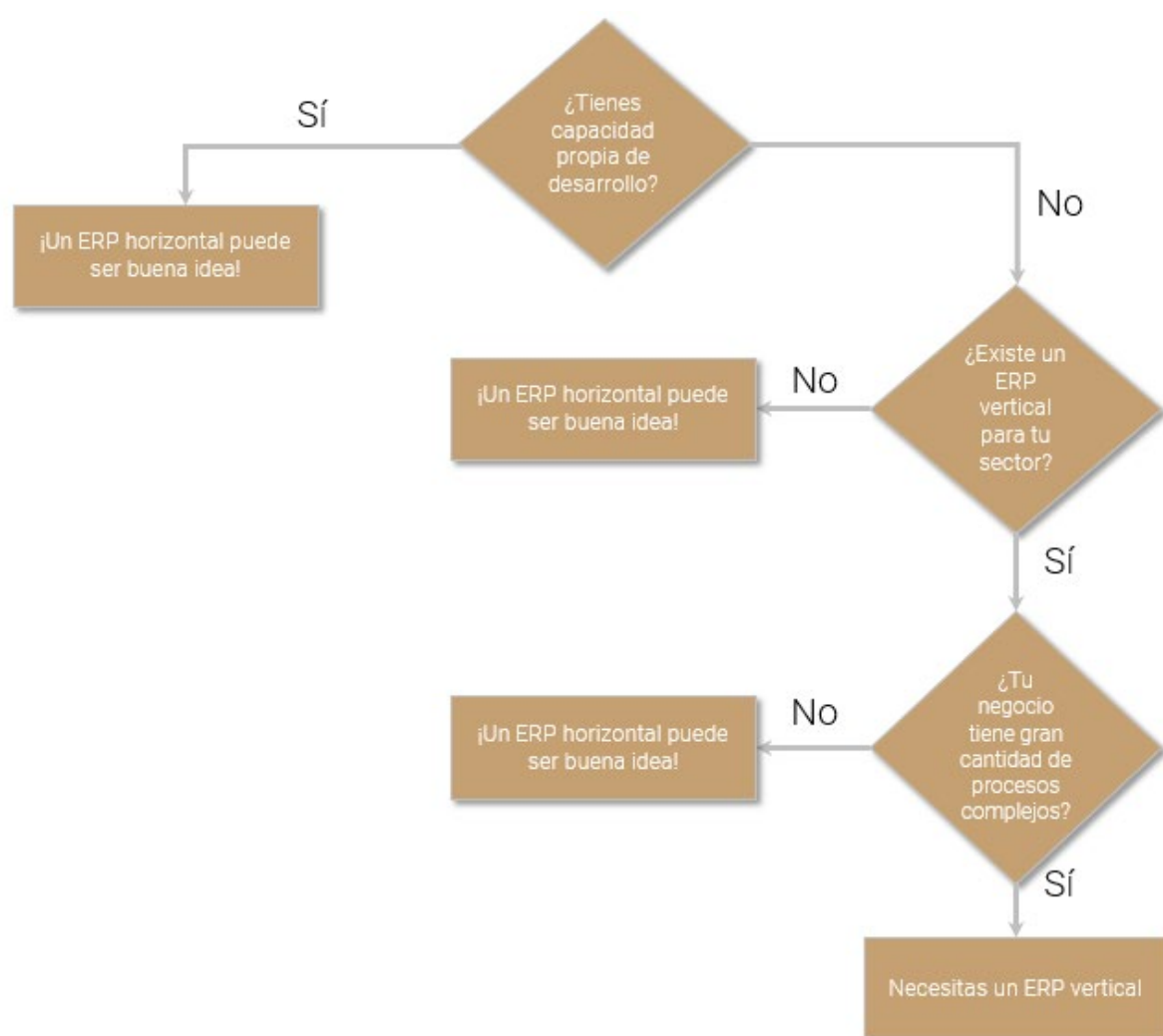


Figura 2.4. ERP vertical vs. ERP horizontal, ¿cuál elegir?.



IMPORTANTE

El apartado 2 ha explicado brevemente las características de los ERP sin entrar en el detalle de ninguno de ellos. Hay que tener en cuenta que el ciclo DAM dispone, en el segundo curso, de un módulo profesional dedicado exclusivamente a este tema. Su nombre es sistemas de gestión empresarial.

3. Gestión de relación con el cliente (CRM)

Un CRM (en inglés, Customer Relationship Management, o gestión de las relaciones con clientes) es una aplicación que permite centralizar en una única base de datos todas las interacciones entre la empresa y sus clientes.

El CRM recopila toda la información de las gestiones comerciales con sus clientes, manteniendo un histórico detallado. De esta forma, el CRM dispone de un gran conocimiento de un cliente dado para entender sus necesidades y anticiparse a ellas.

Por otra parte, el CRM permite dirigir y gestionar de forma sencilla las campañas de captación de clientes y de fidelización. Controla el conjunto de acciones realizadas sobre los clientes actuales o clientes potenciales y gestiona las acciones comerciales a programar en base a la situación y evolución de la empresa.

Las empresas que utilizan software CRM generan más oportunidades de venta, agilizando la gestión, con presupuestos actualizados en tiempo real y procesos de ventas optimizados. Del mismo modo, las compañías que usan CRM disponen de la información necesaria para ofrecer un servicio de atención al cliente y posventa de nivel superior.

Como resumen, diremos que un CRM sirve principalmente para tres cosas:

1. Se mejora la comunicación interna de la empresa. Las tareas relacionadas con los clientes se asignan a los empleados y se hace el seguimiento de su marcha y evolución.
2. Se mejora la productividad en el trabajo, ya que se evitan gran cantidad de correos electrónicos, reuniones y llamadas telefónicas solicitando o enviando información sobre los clientes a otros compañeros de trabajo.
3. Los comerciales venden más gracias a que llevan el seguimiento de todas sus potenciales ventas de forma personalizada y les permite gestionar más ventas a la vez.



Figura 3.1. Funciones del CRM.

3.1. Tipos de CRM

Existen diferentes criterios para clasificar los CRM.

1. Clasificación según su funcionalidad y objetivos.

- **CRM operativo**

El CRM operativo agiliza los procesos de la empresa mediante la automatización de las ventas, el marketing y los servicios. El objetivo principal es generar clientes potenciales, convertirlos en contactos, registrar todos los detalles necesarios y dar servicio a lo largo del ciclo de vida del cliente.

Existen muchos sistemas CRM operativos con diferentes características y plataformas (nube, local, híbrido). Es tarea de la empresa analizar cuál es el que se adapta mejor a sus necesidades.

Ejemplos: Vtiger (pymes), SugarCRM (grandes empresas), Salesforce (cualquier tamaño).

- **CRM analítico**

Un CRM analítico estudia, de forma colaborativa, entre gerencia, marketing, ventas y soporte cuál es la mejor manera de prestar servicios a los clientes. La función principal de este tipo de CRM es el análisis de datos. Analiza los datos de los clientes procedentes de distintas fuentes para obtener una imagen fiable del estado del negocio. Permite a la gerencia tomar mejores decisiones, a marketing entender la efectividad de las campañas, a ventas incrementar los beneficios y al personal de soporte mejorar la atención y construir una relación sólida con el cliente.

Ejemplos: Hubspot (cualquier tamaño), Pipedrive (pymes), SumaCRM (cualquier tamaño).

- **CRM colaborativo**

El CRM colaborativo comparte la información de los clientes con varias áreas de la compañía, como el departamento de ventas, marketing, el equipo técnico y soporte. El CRM recopila el trabajo de las diferentes unidades del negocio para conseguir un objetivo: utilizar toda la información para mejorar la calidad del servicio al cliente, fidelizarlo y captar nuevos clientes para aumentar las ventas.

Ejemplos: Zoho CRM (grandes empresas), Bitrix24 (cualquier tamaño), Nelis (pymes).

2. Clasificación por instalación

- **CRM local (on-premise)**

Este CRM se instala en local, en el servidor de la empresa, y se desarrolla desde cero para uso interno de la organización. Tiene la ventaja de ser altamente personalizado, ajustándose a los objetivos y las necesidades del negocio.

Pero al igual que ocurría con los ERP on-premise, son costosos, ya que la organización debe invertir mucho tiempo y dinero en su desarrollo y formación de los empleados en su uso, así como en el mantenimiento de una infraestructura informática propia.

- **CRM en la nube**

El software CRM alojado en la nube no requiere su descarga e instalación en los equipos, el acceso es a través del navegador web. La base de datos utilizada también está en la nube y no requiere espacio en el centro de datos de la organización. Se paga al proveedor del servicio por su uso, o una suscripción que puede ser mensual o anual.

En general, utilizar este tipo de CRM suele ser más económico, ya que su almacenamiento, funcionamiento, mantenimiento y protección de datos corre a cargo del proveedor.

3.2. CRM en la nube

Cuando la empresa toma la decisión de implementar un CRM online debe tener claros y definidos los objetivos a alcanzar para que su implementación sea válida y poder analizar la rentabilidad.

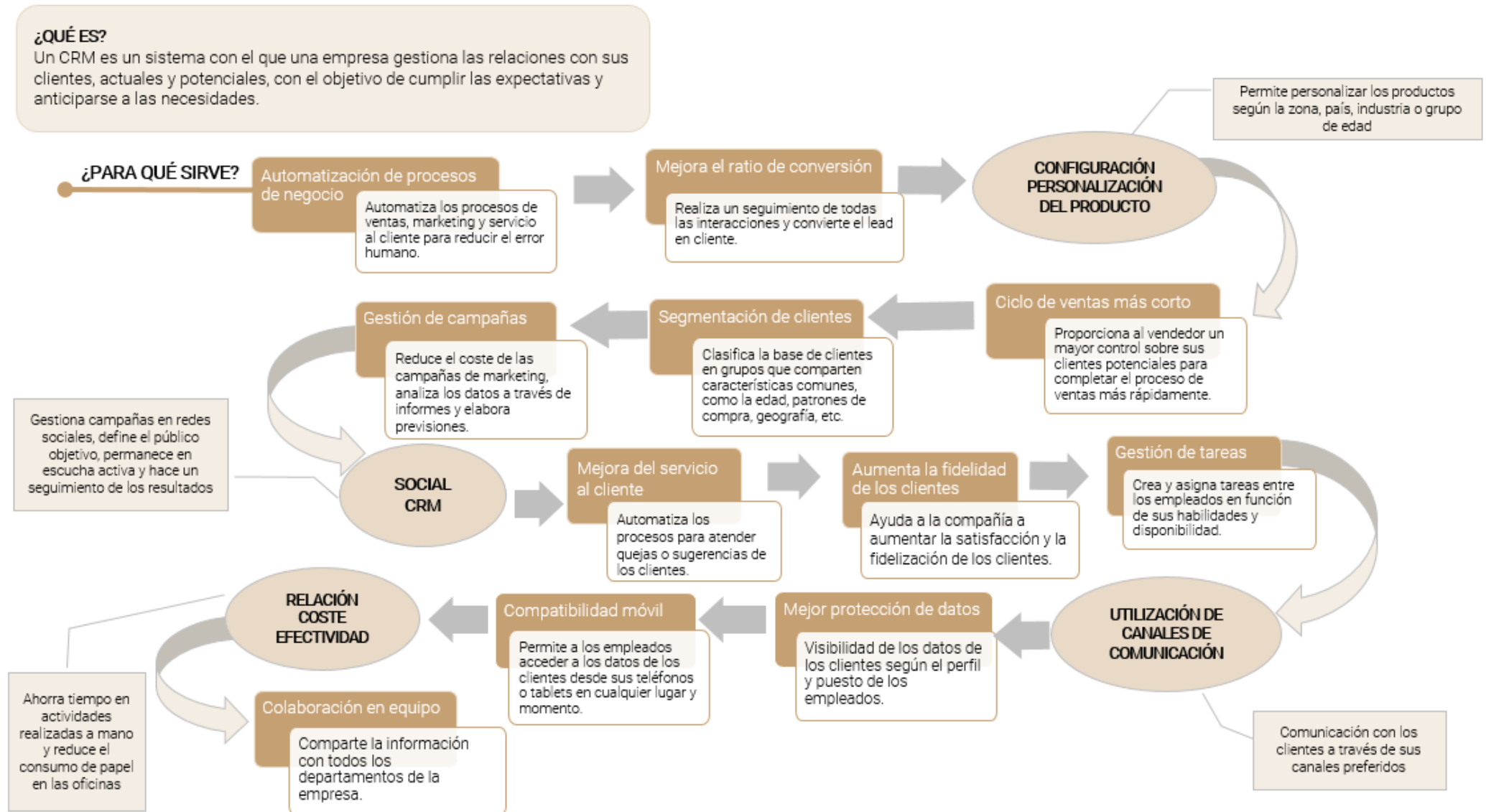


Figura 3.2. Infografía: qué es y para qué sirve un CRM.

Veamos algunos de estos posibles objetivos:

- **Automatizar el flujo de trabajo.**

Prácticamente todos los CRM en la nube disponen de funciones que permiten agilizar todo tipo de tareas, ya sean simples o complejas: desde envíos masivos de correos electrónicos hasta llevar a cabo tareas de análisis de la información de ventas y de clientes para mejorar los resultados de la empresa.

- **Captar nuevos clientes y fidelizar los ya existentes.**

Es un objetivo fundamental cuando el negocio de la empresa depende de los clientes. El CRM está diseñado para mejorar la forma en que se gestiona a cada cliente, permitiendo abarcar una mayor cantidad de ellos y de una manera más personalizada. Si se amplía la cartera actual de clientes de la empresa muy probablemente se incrementará el número de ventas efectivas.

Es cierta la frase “un cliente satisfecho es un cliente fidelizado”, y eso repercute en ingresos para la empresa. Además, es mucho más barato (en dedicación) vender a un cliente existente que adquirir uno nuevo.

- **Mejorar el trabajo del equipo de ventas.**

Si se disminuyen los tiempos de respuesta y se agiliza todo el proceso, el comercial podrá cerrar más ventas. Y como además muchos CRM muestran la evolución de las ventas, cada miembro del equipo puede utilizarlo como guía para establecer de forma precisa su siguiente movimiento y optimizar así su trabajo.

- **Establecer una estrategia de marketing eficaz.**

Las campañas de marketing pueden ayudar a mejorar los resultados de las empresas si están bien planteadas. Disponer de CRM puede contribuir a que estas campañas de marketing sean eficaces, contribuyan a **generar más ventas y a fidelizar los clientes con la empresa.**

Otros beneficios que puede proporcionar un CRM online son los siguientes:

1. Permite la movilidad técnica y el acceso a la información desde cualquier dispositivo.
2. Optimiza los pagos y el proceso de facturación.
3. Vincula las funciones de marketing y ventas para lograr una mejor estrategia.

Como resumen, podemos decir que disponer de CRM online proporciona a la empresa una mayor productividad de los empleados, un mejor conocimiento de los clientes y la capacidad de ahorrar tiempo y recursos con una menor inversión tanto a corto como a largo plazo. Con la ventaja de que el CRM puede crecer en función de la evolución de la empresa.

Para comprobar los beneficios de un CRM online se ha decidido probar vTiger en su versión en la nube. Tiene la ventaja de que es gratuito para un funcionamiento bastante completo y no requiere la descarga de ningún paquete software ni instalación alguna en el ordenador. Existen también versiones de pago, entendemos que más completas, pero las herramientas que ofrece la versión completamente gratuita son suficientes para tener un conocimiento de su funcionamiento y utilidad. Lógicamente, al ser gratuito tiene sus limitaciones, y en este caso la más importante es que solo admite un usuario.

Para disponer de un CRM en la nube vTiger hay que acceder a la página oficial <https://www.nube-crm.es/> del proveedor e ir a “Tu Nube-CRM Gratis” y pulsar en *Empezar*. Los pasos son muy sencillos y todo el proceso está documentado.

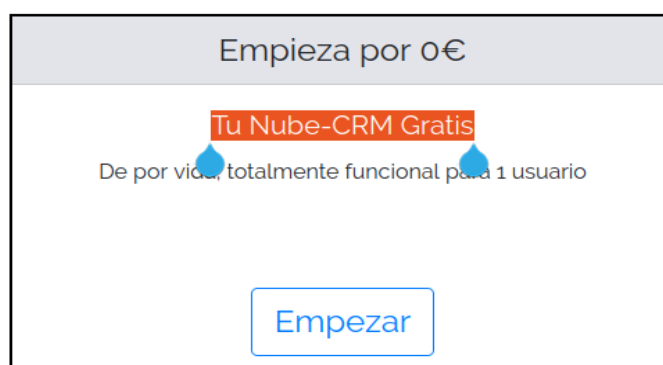


Figura 3.3. Acceso a la versión libre.

Una vez se dispone del CRM en la nube operativo se debe proceder a la personalización y a hacer los ajustes necesarios para la empresa que va a utilizarlo. Este proceso incluye aspectos como incorporar el logo de la compañía o dar de alta a los usuarios con los permisos adecuados (en nuestro caso es el CEO), además de configurar el sistema de avisos que proporciona la aplicación, o la compatibilidad con herramientas de correo, etcétera.

La figura siguiente muestra el panel completo de servicios que ofrece.

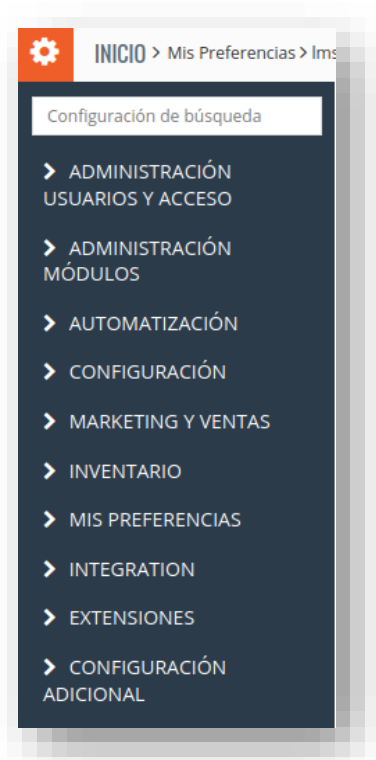


Figura 3.4. Panel de administración.

Los sistemas CRM, en general, permiten la incorporación de distintos tipos de módulos predefinidos que facilitan la personalización del paquete. La integración de estos módulos, que complementan la aplicación base, puede hacerse tanto en el momento de arranque inicial del servicio como en otro momento en el que se requiera ampliar su funcionalidad.

En nuestro caso, el CRM ofrece gran cantidad de módulos, como muestra la siguiente figura.

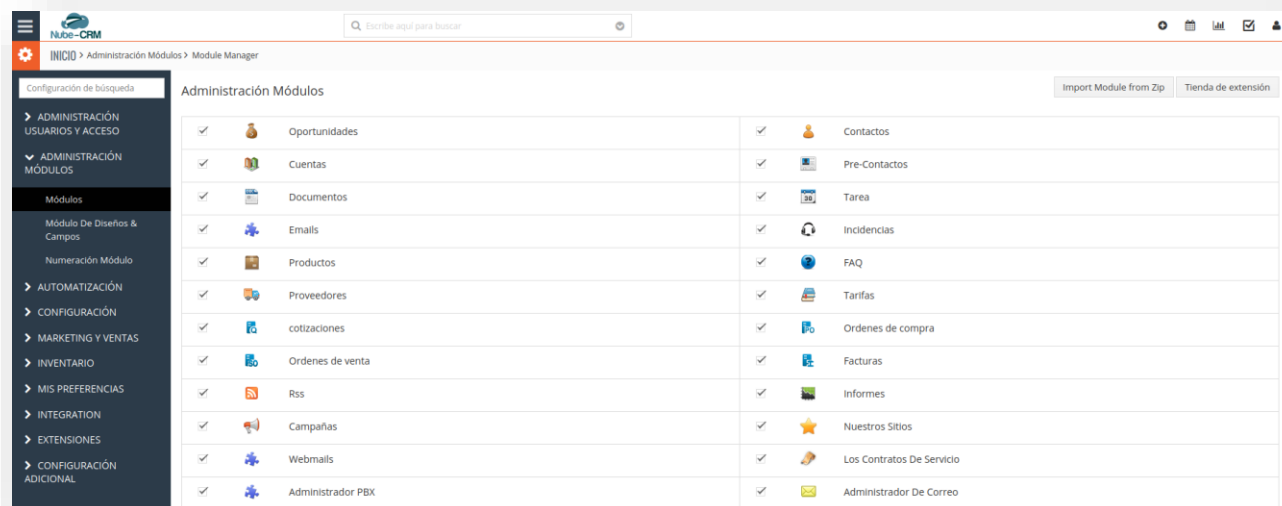


Figura 3.5. Módulos disponibles.

4. Seguridad. Elaboración de informes

A continuación vamos a enumerar los riesgos de seguridad más importantes que tiene un ERP/CRM y una breve descripción de ellos.

1. El ERP/CRM está obsoleto, ya no tiene actualizaciones ni soporte.

Cuando un ERP/CRM se queda obsoleto y ya no tiene soporte por parte del proveedor se pueden presentar problemas de compatibilidad con el resto de las aplicaciones de la empresa, porque es posible que ya no se integre el ERP/CRM con ellas. Pueden aparecer problemas en los servidores, en los navegadores utilizados e incluso con el resto de equipamiento informático.

2. Pérdida de control sobre los datos.

Si en ERP/CRM no dispone de módulos para la generación de informes y hay que realizarlos a través de herramientas externas, e incluso empresas externas, se puede perder el control sobre los datos proporcionados. Por ese motivo es tan importante que, al tomar la decisión sobre qué herramienta utilizar, esta incluya siempre la generación de todos los informes que la empresa vaya a necesitar. Los datos y los accesos a los datos siempre bajo control son una buena medida de seguridad.

3. Personal técnico y desarrolladores externos.

No siempre las amenazas vienen del exterior. A menudo están dentro de la propia empresa o provienen de personal técnico especializado y externo que accede, local o remotamente, para hacer cambios, actualizaciones, etcétera, sobre el ERP/CRM. Se pueden instalar puertas traseras o realizar cambios en la configuración de la herramienta que permitan comportamientos no autorizados o conocidos por la empresa.

4. Actualizaciones de la herramienta.

Una de las medidas de seguridad más importantes y sencillas de llevar a cabo es la constante actualización del código a las últimas versiones estables de la herramienta. Además, también se deben aplicar los necesarios parches de seguridad para evitar cualquier vulnerabilidad del servicio. A menos que sea una vulnerabilidad de "día cero" (no se tiene todavía disponible la solución) siempre hay que tener el propio sistema operativo y las aplicaciones utilizadas, entre ellas el ERP/CRM, actualizadas y/o parcheadas.

5. Incumplimiento de las normas de seguridad.

Además de las políticas de seguridad de la empresa, que son de obligado cumplimiento para todo el personal en todos los niveles, existen unas normas específicas para el correcto uso del ERP/CRM. Por ejemplo, un riesgo muy importante es no cumplir con el estándar de seguridad de datos de la industria de tarjetas de pago (PCI DSS). Este estándar obliga a almacenar los números de las tarjetas de crédito de los clientes siempre encriptadas.

Otras medidas de seguridad que deben estar previstas en las políticas de la empresa son disponer de un cortafuegos, software antimalware actualizado y funcionando de forma proactiva, control estricto sobre el acceso a los datos, copias de seguridad, etcétera.

Estos serían los riesgos potenciales a los que se enfrentaría la empresa. Veamos ahora cuáles son las medidas de seguridad que aportan los propios ERP/CRM. Son las siguientes:

- **Niveles de acceso configurables para los usuarios según su rol.**

El usuario, en función de las tareas que deba realizar, debe contar con los permisos adecuados que le permitan acceder a los datos que necesita. En la medida que tenga un perfil con más responsabilidad en la toma de decisiones tendrá privilegios mayores.

- **Auditoría de cada transacción.**

Proporcionan un control para cada envío de datos, lo que garantiza las operaciones realizadas.

- **Conexión segura HTTPS.**

Para garantizar la seguridad de la comunicación entre los equipos cliente y el servidor en el que está instalada la herramienta se utiliza un protocolo de comunicación seguro que cifre al menos la operación de autenticación del usuario (HTTPS). Aunque sería preferible que todo el canal de comunicación fuera cifrado, por ejemplo, utilizando VPN (red privada virtual).

4.1. Roles de usuario

Los ERP/CRM basan la seguridad que ofrecen, en gran medida, en la asignación de roles a los usuarios que los utilizan. Cada usuario solo tiene acceso a la información almacenada que le hace falta para llevar a cabo su tarea. Por encima de su perfil habrá otro/s perfiles con mayores privilegios que podrán auditar su trabajo.

Los paquetes básicos de los ERP y CRM suelen tener varios tipos de usuarios posibles. El nombre que se les asigne podrá variar de una marca a otra.

- **Administrador:** usuario con todos los privilegios, tanto en lo referente al acceso a la información como a las tareas de gestión del sistema, como son la creación de usuarios, la integración/activación de módulos, la modificación de la configuración y el aspecto del sistema, etcétera.
- **Usuario normal:** usuario sin privilegios de administración, pero tiene todos los privilegios respecto a la información almacenada.
- **Usuario de grupo,** se crean para recibir el correo entrante para su distribución.
- **Usuario de portal,** permite al usuario acceder a los portales creados en el entorno pero no a la aplicación.

Hay aplicaciones en las que los usuarios no pueden eliminarse directamente; hay que hacerlo desde la base de datos, aunque siempre permiten desactivarlos.

Además de los perfiles de usuarios es interesante conocer los diferentes roles que hay definidos para los usuarios de una aplicación.

Un rol define los privilegios del usuario para llevar a cabo tareas específicas.

Las características de los roles son:

- Un rol es un grupo concreto de privilegios.
- Un rol solo tiene validez cuando está asignado a algún usuario.
- Un usuario puede tener asignados varios roles, en ese caso prevalece el rol más restrictivo.
- Los cambios realizados en los roles no son efectivos hasta que no se inicia una nueva sesión.
- Cuando un rol niega el acceso a un módulo se pierde la posibilidad de ver cualquier panel del mismo.

4.2. Informes

Los sistemas ERP/CRM suelen disponer de herramientas que facilitan la elaboración de informes que ayudarán en la toma de decisiones estratégicas para mejorar estos tres aspectos:

1. La planificación.
2. La gestión.
3. El control.

Estas herramientas permiten, a cualquier usuario sin conocimientos técnicos hacer informes dinámicos, flexibles e interactivos, para que el usuario no esté limitado por listados predefinidos.

También hacen una integración entre todos los sistemas/departamentos de la empresa, garantizando la calidad y la integración de los datos dentro de la misma.

SuiteCRM es el CRM local que se va a utilizar en las prácticas. Dispone del módulo de generación de informes y se trabajará con él.