Unidad 1

Lenguajes de marcas.

XML

Apuntes realizados para la asignatura de FP Grado Superior:

Lenguajes de Marcas y Sistemas de Gestiónde Información

del ciclo Administración de Sistemas Informáticos en Red

Autor: Jorge Sánchez Asenjo (www.jorgesanchez.net)

Versión del documento: 2.1, Año 2012

Atribución-NoComercial-CompartirIgual 3.0 Unported (CC BY-NC-SA 3.0)

Esto es un resumen fácilmente legible del **Texto Legal (la licencia completa)**.

http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

Usted es libre de:

Compartir - copiar, distribuir, ejecutar y comunicar públicamente la obra

hacer obras derivadas

Bajo las condiciones siguientes:

Atribución — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciante (pero no de una manera que sugiera que tiene su apoyo o que apoyan el uso que hace de su obra).

No Comercial — No puede utilizar esta obra para fines comerciales.

Compartir bajo la Misma Licencia — Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Entendiendo que:

Renuncia — Alguna de estas condiciones puede **no aplicarse** si se obtiene el permiso del titular de los derechos de autor

Dominio Público — Cuando la obra o alguno de sus elementos se halle en el **dominio público** según la ley vigente aplicable, esta situación no quedará afectada por la licencia.

Otros derechos — Los derechos siguientes no quedan afectados por la licencia de ninguna manera:

- Los derechos derivados de **usos legítimos** u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
- Los derechos **morales** del auto;
- Derechos que pueden ostentar otras personas sobre la propia obra o su uso, como por ejemplo **derechos de imagen** o de privacidad.

índice

(1) lenguajes de marcas. XML

(1.1) ordenador e información

(1.1.1) formas de representar información en el ordenador

El ordenador es una máquina digital, por lo tanto sólo es capaz de representar información utilizando el sistema binario de numeración. Esto obliga a que, para poder almacenar información en un ordenador, previamente haya que codificarla en forma de números binarios.

El problema de los números binarios es que están muy alejados del ser humano; es decir, que las personas no estamos capacitadas para manejar información en binario. Nosotros usamos sistema decimal para los números y formas de representación mucho más complejas para otra información (como el texto, las imágenes, la música,…)

Sin embargo actualmente un ordenador es capaz de manejar información de todo tipo: música, imágenes, texto,…. Esto es posible porque se ha conseguido que casi cualquier tipo de información sea codificable en binario.

Los seres humanos tenemos la capacidad de diferenciar claramente lo que es un texto de una imagen, lo que es un número de una canción,… Pero en un ordenador todo es más complicado, porque todo es binario.

Desde los inicios de la informática la **codificación** (el paso de información humana a información digital) ha sido problemática debido a la falta de acuerdo en la representación. Pero hoy día ya tenemos numerosos estándares.

Fundamentalmente la información que un ordenador maneja son **Números** y **Texto**. Pero curiosamente a nivel formal se consideran **datos binarios** a cualquier tipo de información representable en el ordenador, que no es texto (imagen, sonido, vídeo,…), aunque como ya hemos comentado, en realidad toda la información que maneja un ordenador es binaria, incluido el texto.

(1.1.2) datos en forma de texto y datos binarios

datos binarios

Cualquier dato que no sea texto, se considera dato binario. Por ejemplo: música, vídeo, imagen, un archivo Excel, un programa,…

La forma de codificar ese tipo de datos a su forma binaria es muy variable. Por ejemplo en el caso de las imágenes, cada punto (**píxel**) de la imagen se codifica utilizando su nivel de rojo, verde y azul. De modo que una sola imagen produce millones de dígitos binarios.

En cualquier caso sea cual sea la información que estamos codificando en binario, para poder acceder a dicha información, el ordenador necesita el software que sepa como **decodificar** la misma, es decir saber qué significa cada dígito binario para traducirle a una forma más humana. Eso sólo es posible utilizando el mismo software con el que se codificó o bien otro software pero que sea capaz de entender la información codificada.

texto

El texto es quizá la forma más humana de representar información. Antes de la llegada del ordenador, la información se transmitía mediante documentos o libros en papel. Esa forma de transmitir es milenaria y sigue siendo la forma más habitual de transmitir información entre humanos; incluso con la tecnología actual aplicaciones como **twitter**, **whassap**,… siguen usando el texto como formato fundamental para transmitir información.

En cuanto apareció la informática como una ciencia digital, apareció también el problema de cómo codificar texto en forma de dígitos binarios para hacerlo representable en el ordenador. La forma habitual ha sido codificar cada carácter en una serie de números binarios. De modo que por ejemplo el carácter ***A*** fuera por ejemplo ***01000001*** y la ***B*** el ***01000010***.

El problema surgió por la falta de estandarización, la letra A se podía codificar distinto en diferentes ordenadores y así nos encontrábamos con un problema al querer pasar datos de un ordenador a otro. Poco a poco aparecieron estándares para intentar que todo el hardware y software codificara los caracteres igual.

el código ASCII

El problema de la codificación de texto que hacía incompatibles los documentos de texto entre diferentes sistemas, se palió cuando se ideó en 1967 un código estándar por parte de la **ANSI**, la agencia de estándares norteamericana, dicho código es el llamado ASCII (***American Standard Code for Information Interchange***, código estándar americano para el intercambio de información). El código utiliza el alfabeto inglés (que utiliza caracteres latinos) y para codificar todos los posibles caracteres necesarios para escribir en inglés se ideó un sistema de 7 bits (con 7 bits se pueden representar 128 símbolos, suficientes para todas las letras del alfabeto inglés, en minúsculas y mayúsculas, caracteres de puntuación, símbolos especiales e incluso símbolos de control).

El código ASCII es el siguiente:

Núm.	Significado	¿Control?	Núm.	Sign.	¿?	Núm	Num.	¿?	Núm	Num.	¿Control?
0	Carácter nulo	Sí	32	Espacio	No	64	@	No	96	`	No
1	Inicio de Encabezado	Sí, ctrl-A	33	!	No	65	A	No	97	a	No

Núm.	Significado	¿Control?	Núm.	Sign.	¿?	Núm	Num.	¿?	Núm	Num.	¿Control?
2	Inicio de Texto	Sí, ctrl-B	34	"	No	66	B	No	98	b	No
3	Fin de Texto	Sí, ctrl-C	35	#	No	67	C	No	99	c	No
4	Fin de Transmisión	Sí, ctrl-D	36	$	No	68	D	No	100	d	No
5	Petición	Sí, ctrl-E	37	%	No	69	E	No	101	e	No
6	Confirmación	Sí, ctrl-F	38	&	No	70	F	No	102	f	No
7	Timbre	Sí, ctrl-G	39	'	No	71	G	No	103	g	No
8	Retroceso	Sí, ctrl-H	40	(	No	72	H	No	104	h	No
9	Tabulación horizontal	Sí, ctrl-I	41	)	No	73	I	No	105	i	No
10	Alimentación de línea	Sí, ctrl-J	42	*	No	74	J	No	106	j	No
11	Tabulación Vertical	Sí, ctrl-K	43	+	No	75	K	No	107	k	No
12	Alimentación de carro	Sí, ctrl-L	44	,	No	76	L	No	108	l	No
13	Retorno de carro	Sí, ctrl-M	45	-	No	77	M	No	109	m	No
14	Quitar mayúsculas	Sí, ctrl-N	46	.	No	78	N	No	110	n	No
15	Poner mayúsculas	Sí, ctrl-O	47	/	No	79	O	No	111	o	No
16	Data Link Escape	Sí, ctrl-P	48	0	No	80	P	No	112	p	No
17	Control Disp-1	Sí, ctrl-Q	49	1	No	81	Q	No	113	q	No
18	Control Disp-2	Sí, ctrl-R	50	2	No	82	R	No	114	r	No
19	Control Disp-3	Sí, ctrl-S	51	3	No	83	S	No	115	s	No
20	Control Disp-4	Sí, ctrl-T	52	4	No	84	T	No	116	t	No
21	Confirmación negativa	Sí, ctrl-U	53	5	No	85	U	No	117	u	No
22	Idle síncrono	Sí, ctrl-V	54	6	No	86	V	No	118	v	No
23	Fin de bloque de transmisión	Sí, ctrl-W	55	7	No	87	W	No	119	w	No
24	Cancelar	Sí, ctrl-X	56	8	No	88	X	No	120	x	No
25	Fin de mitad	Sí, ctrl-Y	57	9	No	89	Y	No	121	y	No
26	Sustituto	Sí, ctrl-Z	58	:	No	90	Z	No	122	z	No
27	Escape	Sí, ctrl-[	59	;	No	91	[	No	123	{	No
28	EOF	Sí, ctrl-\	60	<	No	92	\	No	124	|	No
29	Separador de Grupo	Sí, ctrl-]	61	=	No	93	]	No	125	}	No
30	Separador de registro	Sí, ctrl-^	62	>	No	94	^	No	126	~	No
31	Separador de unidad	Sí, ctrl-_	63	?	No	95	_	No	127	Borrado	No

Pero, en países con lenguas distintas del inglés, surgió el problema de que parte de los símbolos de sus alfabetos quedaban fuera del ASCII (como la letra eñe)-

Por ello se diseñaron códigos de 8 bits que añadían 128 símbolos más y así aparecieron los llamados códigos **ASCII extendidos**. En ellos, los 128 símbolos primeros son los mismos de la tabla ASCII original y los 128 siguientes se corresponden a símbolos extra. Así por ejemplo el sistema MS-DOS utilizaba el llamado código ***437*** que incluía símbolos y caracteres de otras lenguas de Europa Occidental y caracteres que permitían hacer marcos y bordes en pantallas de texto, entre otros símbolos.

Sin embargo 8 bits siguen siendo insuficientes para codificar todos los alfabetos del planeta. Por lo que cada zona usaba su propia tabla ASCII extendida. Ante el caos consiguiente, la ISO decidió normalizar dichas tablas de códigos para conseguir versiones estándares de los mismos. Lo hizo mediante las siguientes normas (cada una

de las cuales definía una tabla de 256 caracteres, siempre los 128 primeros son el ASCII original)

- **8859-1**. ASCII extendido para Europa Occidental (incluye símbolos como ñ o ß)
- **8859-2**. ASCII extendido para Europa Central y del Este (incluye símbolos como Ź o č)
- **8859-3**. ASCII extendido para Europa del Sur (incluye símbolos como Ġ o Ï)
- **8859-4**. ASCII extendido para Europa del Norte (incluye símbolos como ø o å)
- **8859-5**. ASCII extendido para alfabeto cirílico (incluye símbolos como Ð o Ж)
- **8859-6**. ASCII extendido para alfabeto árabe (incluye símbolos como □ o □)
- **8859-7**. ASCII extendido para alfabeto griego moderno (incluye símbolos como φ o α)
- **8859-8**. ASCII extendido para alfabeto hebreo (incluye símbolos como ץ o ן)
- **8859-9**. ASCII extendido, versión de 8859-1 que incluye símbolos turcos en lugar de otros poco utilizados
- **8859-10**. ASCII extendido, versión de 8859-4 que incluye símbolos más utilizados en las lenguas nórdicas actuales
- **8859-11**. ASCII extendido para alfabeto tailandés (incluye símbolos como ๗ o ๎)
- **8859-12**. ASCII extendido para alfabeto devanagari de India y Nepal que ya no se usa
- **8859-13**. ASCII extendido para alfabetos bálticos con símbolos que no estaban en 8859-4
- **8859-14**. ASCII extendido para alfabeto celta (incluye símbolos como ŵ o Ẃ)
- **8859-15**. ASCII extendido, versión de 8859-1 que incluye el símbolo del euro y símbolos de lenguas bálticas. Es el recomendado actualmente para Europa Occidental.
- **8859-16**. ASCII extendido, versión de 8859-1 pensada para los países del sureste de Europa
- **2022-JP**. Símbolos japoneses (parte 1)
- **2022-JP-2**. Símbolos japoneses (parte 2)
- **2022-KR**. Símbolos coreanos

Este problema sigue existiendo ahora de modo que en los documentos de texto hay que indicar el sistema de codificación utilizado (el caso más evidente son las páginas web), para saber cómo interpretar los códigos del archivo. Así en 8859_1 el código **245** es el carácter õ y en **8859_2** es el carácter ő

Unicode

La complicación de las tablas de código se intenta resolver gracias al sistema **Unicode** que ha conseguido incluir los caracteres de todas las lenguas del planeta a cambio de que cada carácter ocupe más de un byte (ocho bits). En Unicode a cada símbolo se le asigna un número (evidentemente los 128 primeros son los originales de ASCII para

mantener la compatibilidad con los textos ya codificados y de hecho los 256 primeros son la tabla ISO-8859_1).

Para ello el organismo también llamado Unicode participado por numerosas e influyentes empresas informáticas y coordinado por la propia ISO, ha definido tres formas de codificar los caracteres:

- **UTF-8.** Es la más utilizada (y la más compleja de usar para el ordenador). Utiliza para cada carácter de uno a cuatro caracteres, de forma que:
  - Utilizan uno los que pertenecen al código ASCII original
  - Dos los pertenecientes a lenguas latinas, cirílicas, griegas, árabes, hebreas y otras de Europa, Asia Menor y Egipto
  - Tres para símbolos fuera de los alfabetos anteriores como el chino o el japonés
  - Cuatro para otros símbolos: por ejemplo los matemáticos y símbolos de lenguas muertas como el fenicio o el asirio o símbolos asiáticos de uso poco frecuente.
- **UTF-16.** Utiliza para cada carácter dos (para los dos primeros grupos del punto anterior) o cuatro caracteres (para el resto). Es más sencillo que el anterior
- **UTF-32.** La más sencilla de todas. Cada carácter independientemente del grupo al que pertenezca ocupa 4 caracteres. No se utiliza.

archivos binarios y archivos de texto

ventajas de los archivos binarios

(1) Ocupan menos espacio que los archivos de texto, ya que optimizan mejor su codificación a binario (por ejemplo el número *213* ocupa un solo byte y no tres como ocurriría si fuera un texto).

(2) Son más rápidos de manipular por parte del ordenador (se parecen más al lenguaje nativo del ordenador)

(3) Permiten el acceso directo a los datos. Los archivos de texto siempre se manejan de forma secuencial, más lenta

(4) En cierto modo permiten cifrar el contenido que de otra forma sería totalmente visible por cualquier aplicación capaz de entender textos (como el bloc de notas). Es decir los datos no son fácilmente entendibles

ventajas de los archivos de texto

(1) Son ideales para almacenar datos para exportar e importar información a cualquier dispositivo electrónico ya que cualquier es capaz de interpretar texto

(2) Son directamente modificables, sin tener que acudir a software específico

(3) Su manipulación es más sencilla que la de los archivos binarios

(4) Son directamente transportables y entendibles por todo tipo de redes

(1.2) exportar/importar datos

(1.2.1) el problema de compartir datos

Los problemas relacionados con el intercambio de información entre aplicaciones y máquinas informáticas es tan viejo como la propia informática.

El problema parte del hecho de haber realizado un determinado trabajo con un software en un ordenador concreto y después querer pasar dicho trabajo a otro software en ese u otro ordenador.

Los archivos binarios tienen la complicación de que para hacer ese proceso, el origen y el destino de los datos deben comprender cómo codificar y decodificar la información. Eso, en muchos casos, ha sido un gran problema que ha obligado a que todos los trabajadores y trabajadoras hayan tenido que adaptarse al software de la empresa y por supuesto en toda la empresa utilizar dicho software.

En la informática actual eso es aún más problema al tener una necesidad de disponibilidad global del trabajo y además la posibilidad de ver dicho trabajo en dispositivos de todo tipo como mini ordenadores, PDA o incluso teléfonos móviles.

Por ello poco a poco han aparecido formatos binarios de archivo que han sido estándares de facto (aunque no han sido reconocidos por ningún organismo de estándares) como por ejemplo el formato documental **PDF**, el formato de imagen **JPEG**, la música **MP3** o el formato **MPEG** de vídeo.

Pero sigue habiendo empresas que utilizan formato propio por la idea de que sus formatos de archivo están directamente relacionados con la calidad de su software es decir razonan que el software que fabrican es muy potente y necesitan un formato binario propio compatible con esa potencia. De ahí que muchas veces la opción para exportar e importar datos sea utilizar conversores, capaces de convertir los datos de un formato a otro (por ejemplo de **Word** a **Open Office**; de **MP3** a **MOV** de Apple, etc.).

(1.2.2) el texto como el formato más versátil

Sin embargo hay un formato de archivo que cualquier dispositivo es capaz de entender. El texto. La cuestión es que para los archivos llamados de texto, sólo son capaces de almacenar **texto plano**; es decir sólo texto sin indicar ningún formato o añadir información no textual.

Debido a la facilidad de ser leído con cualquier aparato, se intenta que el propio texto sirva para almacenar otros datos. Evidentemente no es posible usar texto para almacenar por ejemplo imágenes, pero sí otras cosas. Para ello dentro del archivo habrá contenido que no se interpretará como texto sin más que simplemente se debe mostrar, sino que hay texto en el archivo que se marca de manera especial haciendo que signifique otra cosa. Desde hace muchos añoso hay dos campos en los que está idea ha funcionado bien: en las bases de datos y en los procesadores de texto. Actualmente el éxito de Internet ha permitido espolear esta tecnología a otros campos.

Hay un problema con el texto, puesto que al ser formato tan universal, y ser su contenido siempre visible; es peligroso como fuente para almacenar datos confidenciales, ya que quedaría expuesto a cualquier persona.

Los datos binarios no son del todo seguros, pero como requieren del software que entienda el formato binario concreto hacen que su contenido quede menos expuesto.

(1.3) lenguajes de marcas

(1.3.1) introducción histórica

aparición de los lenguajes de marcas

Como se ha comentado en el punto anterior, el problema de la exportación de datos ha puesto en entredicho a los archivos binarios como fuente para exportar e importar información.

En su lugar parece que los archivos de texto poseen menos problemas (excepto el del cifrado de su información, que queda demasiado descubierta). Por ello se ha intentado que los archivos de **texto plano** (archivos que sólo contienen texto y no otros datos binarios) pudieran servir para almacenar otros datos como por ejemplo detalles sobre el formato del propio texto u otras indicaciones.

Los **procesadores de texto** fueron el primer software en encontrarse con este dilema. Puesto que son programas que sirven para escribir texto parecía que lo lógico era que sus datos se almacenaran como texto. Pero necesitan guardar datos referidos al formato del texto, tamaño de la página, márgenes, etc. La solución clásica ha sido guardar la información de formato de forma binaria, lo que provoca los ya comentados problemas.

Algunos procesadores de texto optaron por guardar toda la información como texto, haciendo que las indicaciones de formato no se almacenen de forma binaria sino textual. Dichas indicaciones son caracteres **marcados** de manera especial para que así un programa adecuado pueda traducir dichos caracteres no como texto sino como operaciones que finalmente producirán mostrar el texto del documento de forma adecuada..

La idea del marcado procede del inglés ***marking up*** término con el que se referían a la técnica de marcar manuscritos con lápiz de color para hacer anotaciones como por ejemplo la tipografía a emplear en las imprentas. Este mismo término se ha utilizado para los documentos de texto que contienen comandos u anotaciones.

Las posibles anotaciones o indicaciones incluidos en los documentos de texto han dado lugar a ***lenguajes*** (entendiendo que en realidad son formatos de documento y no lenguajes en el sentido de los lenguajes de programación de aplicaciones) llamados **lenguajes de marcas**, **lenguajes de marcado** o **lenguajes de etiquetas**.

Goldfarb

Se considera a **Charles Goldfarb** como al padre de los lenguajes de marcas. Se trata de un investigador de IBM que propuso ideas para que los documentos de texto tuvieran la posibilidad de indicar el formato del mismo. Al final ayudó a realizar el lenguaje **GML** de **IBM** el cual puso los cimientos del futuro **SGML** ideado por el propio Goldfarb.

TeX y LaTeX

En la década de los 70 **Donald Knuth** (uno de los ingenieros informáticos más importantes de la historia, padre del análisis de algoritmos) creó TEX para producir documentos científicos utilizando una tipografía y capacidades que fueran iguales en cualquier computadora, asegurando además una gran calidad en los resultados.

Para ello apoyó a TeX con tipografía especial (fuentes **Modern Computer**) y un lenguaje de definición de tipos (**METAFONT**). TeX ha tenido cierto éxito en la comunidad científica gracias a sus 300 comandos que permiten crear documentos con

tipos de gran calidad, para ello se necesita un programa capaz de convertir el archivo TeX a un formato de impresión.

El éxito de TeX produjo numerosos derivados de los cuales el más popular es LaTeX (**LaTeX**). Se trata de un lenguaje que intenta simplificar a TeX, fue definido en 1984 por **Leslie Lamport**, aunque después ha sido numerosas veces revisado. Al utilizar comandos de TeX y toda su estructura tipográfica, adquirió rápidamente notoriedad y sigue siendo utilizado para producir documentos con expresiones científicas, de gran calidad. La idea es que los científicos se centren en el contenido y no en la presentación. Ejemplo de código LaTeX:

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\title{\Ejemplo}
\begin{document}
Este es el texto ejemplo de \LaTeX{}
Con datos en \emph{cursiva} o \textbf{negrita}.
Ejemplo de f\'ormula
\begin{align}
E &= mc^2
\end{align}
\end{document}
```

Que con un traductor daría lugar al resultado:

Este es el texto ejemplo de LaTeX
Con datos en *cursiva* o **negrita**. Ejemplo de fórmula

$$E = mc^2 \qquad (1)$$

RTF

RTF es el acrónimo de **Rich Text Format** (***Formato de Texto Enriquecido***) un lenguaje ideado por **Microsoft** en 1987 para producir documentos de texto que incluyan anotaciones de formato.

Actualmente se trata de un formato aceptado como texto con formato y en ambiente Windows es muy utilizado como formato de intercambio entre distintos procesadores por su potencia. El procesador de texto **Word Pad** incorporado por Windows lo utiliza como formato nativo. Ejemplo:

```
{\rtf1\ansi\ansicpg1252\deff0\deflang3082{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
\viewkind4\uc1\pard\sa200\sl276\slmult1\lang10\f0\fs22 soy \i cursiva\i0\par
}
```

Produce el resultado:

soy *cursiva*

SGML

Se trata de la versión de GML que estandarizaba el lenguaje de marcado y que fue definida finalmente por ISO como estándar mundial en documentos de texto con etiquetas de marcado. La estandarización la hace el subcomité SC24 que forma parte del comité JTC1 del organismo IEC de ISO que se encarga de los estándares electrónicos e informáticos (en definitiva se trata de una norma ISO/IEC JTC1/SC24, concretamente la 8879).

Su importancia radica en que es el padre del lenguaje XML y la base sobre la que se sostiene el lenguaje HTML.

En SGML las etiquetas que contienen indicaciones para el texto se colocan entre símbolos < y >. Las etiquetas se cierran con el signo /. Es decir las reglas fundamentales de los lenguajes de etiquetas actuales ya las había definido SGML.

En realidad (como XML) no es un lenguaje con unas etiquetas concretas, sino que se trata de un lenguaje que sirve para definir lenguajes de etiquetas; o más exactamente es un lenguaje de marcado que sirve para definir formatos de documentos de texto con marcas. Entre los formatos definidos mediante SGML, sin duda HTML es el más popular.

PostScript

Se trata de un lenguaje de descripción de páginas. De hecho es el más popular. Permite crear documentos en los que se dan indicaciones potentísimas sobre como mostrar información en el dispositivo final. Se inició su desarrollo en 1976 por John Warnock y dos años más tarde se continuo con la empresa Xerox, hasta que en 1985 el propio Warnock funda Adobe Systems y desde esa empresa se continua su desarrollo.

Es en realidad todo un lenguaje de programación que indica la forma en que se debe mostrar la información que puede incluir texto y el tipo de letra del mismo, píxeles individuales y formas vectoriales (líneas, curvas). Sus posibilidades son muy amplias. Ejemplo[1]:

```
%!PS
/Courier                    % Elige el tipo de letra
20 selectfont               % Establece el tamaño de la letra y
                            % la toma como el tipo de letra en uso
72 500 moveto               % Coloca el cursor en las coordenadas
                            % 72, 500 (contando los píxeles desde
                            % la esquina izquierda de la página)
(Hola mundo!) show          % Escribe el texto entre paréntesis,
showpage                    % Imprime el resultado
```

HTML

Tim Bernes Lee utilizó SGML para definir un nuevo lenguaje de etiquetas que llamó Hypertext Markup Language (lenguaje de marcado de hipertexto) para crear documentos transportables a través de Internet en los que fuera posible el hipertexto; es decir la posibilidad que determinadas palabras marcadas de forma especial permitieran abrir un documento relacionado con ellas.

[1] Tomado de http://en.wikipedia.org/wiki/PostScript

A pesar de tardar en ser aceptado, HTML fue un éxito rotundo y la causa indudable del éxito de Internet. Hoy en día casi todo en Internet se ve a través de documentos HTML, que popularmente se denominan **páginas web**.

Inicialmente estos documentos se veían con ayuda de intérpretes de texto (como por ejemplo el **Lynx** de **Unix**) que simplemente coloreaban el texto y remarcaban el hipertexto. Después el software se mejoró y aparecieron navegadores con capacidad más gráfica para mostrar formatos más avanzados y visuales.

XML

Se trata de un subconjunto de SGML ideado para mejorar el propio SGML y con él definir lenguajes de marcado con sintaxis más estricta, pero más entendibles. Su popularidad le ha convertido en el lenguaje de marcado más importante de la actualidad y en el formato de documentos para exportación e importación más exitoso.

JSON

Abreviatura de ***JavaScript Object Notation***, Se trata de una notación de datos procedente del lenguaje JavaScript estándar (concretamente ECMA Script de 1999). En el año 2002 se le daba soporte desde muchos de los navegadores y su fama ha sido tal que ahora se ha convertido en una notación independiente de JavaScript que compite claramente con XML.

Se trata de una notación que realmente no se considera lenguaje de marcas, ya que no hay diferencia en el texto a través de etiquetas, sino que se basa en que el texto se divide en dato y metadato. De modo que el símbolo de los dos puntos separa el metadato del dato. Por otro lado los símbolos de llave y corchete permiten agrupar de manera correcta los datos. Ejemplo de JSON:

```
{
  "nombre": "Jorge",
  "apellido1": "Sánchez",
  "dirección": {
    "calle": "C/ Falsa nº 0",
    "localidad": "Palencia",
    "código Postal": "34001",
    "país":"España"
  },
  "teléfonos": [
    {
      "tipo": "fijo",
      "número": "999 999 999"
    },
    {
      "tipo": "móvil",
      "number": "666 666 666"
    }
  ]
}
```

(1.3.2) tipos de lenguajes de marcas

- **Orientados a la presentación**. En ellos al texto común se añaden palabras encerradas en símbolos especiales que contienen indicaciones de formato que permiten a los traductores de este tipo de documentos generar un documento final en el que el texto aparece con el formato indicado. Es el caso de HTML en el que se indica cómo debe presentarse el texto (y no por ejemplo lo que significa el mismo) también se considera así los archivos generados por los procesadores de texto tradicionales en los que al texto del documento se le acompaña de indicaciones de formato (como negrita, cursiva,…)
- **Orientados a la descripción**. En ellos las marcas especiales permiten dar significado al texto pero no indican cómo se debe presentar en pantalla el mismo. Sería el caso de **XML** (o de **SGML**) y JSON en el que la presentación nunca se indica en el documento; simplemente se indica una semántica de contenido que lo hace ideal para almacenar datos (por ejemplo si el texto es un nombre de persona o un número de identificación fiscal).
- **Orientados a procedimientos**. Se trata de documentos en los que hay texto marcado especialmente que en realidad se interpreta como órdenes a seguir y así el archivo en realidad contiene instrucciones a realizar con el texto. Es el caso de **LaTeX** o **PostScript** donde por ejemplo se puede indicar una fórmula matemática.

(1.4) ¿qué es XML?

(1.4.1) objetivos de XML

XML es un lenguaje de marcas que se ha estandarizado y se ha convertido en uno de los formatos más populares para intercambiar información.

Se trata de un formato de archivos de texto con marcado que deriva del original **SGML**, pero que le ha superado añadiendo otro tipo de reglas y de forma de trabajar.

La realidad es que XML siempre ha estado muy ligado al éxito de HTML. Se planteó por los problemas crecientes que se fueron observando en las páginas web.

HTML tiene estos problemas como formato de intercambio de información:

- La mayoría de etiquetas HTML no son semánticas; es decir, no sirven para decir el tipo de contenido que tenemos sino para indicar el formato. Por ejemplo la etiqueta ***H1*** sí es semántica ya que indica que el texto que contiene es un encabezado de nivel principal. Mientras que la etiqueta HTML clásica ***font*** sirve para colorear o cambiar el tipo de letra, sin indicar qué tipo de texto tenemos (se aplica a cualquiera)
- HTML es un lenguaje rígido, no es extensible. Es decir no podemos añadir etiquetas ya que ningún navegador las reconocerá. Cada vez que se decide añadir hay que cambiar el estándar y los navegadores se deben de adaptar a los cambios.
- Requiere de arreglos “extraños” para añadir potencia y funcionalidad, por lo que los diseñadores tienden a incrustar dentro del código HTML código de lenguajes como **PHP** o **Javascript** que dificultan su legibilidad y comprensión.

Por ello al crear XML se plantearon estos objetivos:

(1) Debía de ser similar a HTML (de hecho se basa en el lenguaje SGML base para el formato HTML)

(2) Debía de ser extensible, es decir que sea posible añadir nuevas etiquetas sin problemas. Esa es la base del lenguaje XML.

(3) Debía de tener unas reglas concisas y fáciles, además de estrictas.

(4) Debía de ser fácil de implantar en todo tipo de sistemas. XML nace con una vocación multiplataforma, como base de intercambio de información entre sistemas de toda índole.

(5) Debía ser fácil de leer por los humanos y fácil crear procesadores XML software (llamados **parsers**)

(1.4.2) XML y SGML

Básicamente XML es SGML, es decir una persona que sepa SGML no tendrá ningún problema en aprender XML, las bases son las mismas pero XML elimina gran parte de su complejidad. De hecho se dice que XML es un subconjunto de SGML; es SGML pero restringiendo o eliminando ciertas normas.

Por otro lado XML sirve para lo mismo que SGML: para diseñar lenguajes de mediante marcas. Es decir, XML define tipos de documentos de forma que junto a la información del documento hay una serie de **etiquetas** que sirven para clarificar lo que significa la información.

De esa manera XML (como ya hizo SGML) es un lenguaje que permite especificar otros lenguajes. Entendiendo como lenguajes en este caso, al conjunto de etiquetas se pueden utilizar en un documento; no solo qué etiquetas sino en qué orden y de qué forma se pueden utilizar.

(1.4.3) lenguajes basados en XML

Algunos de los lenguajes estándares de marcado basados en XML son:

- **RSS**. (*Really Simple Syndication*, aunque hay otras interpretaciones de los acrónimos) Para producir contenidos sindicables, se utiliza fundamentalmente para producir noticias. Es una de las aplicaciones XML más utilizada
- **Atom**. Formato semejante al anterior, pensado también para distribuir información desde una web. Muchas veces complementa a RSS
- **ePUB**. Formato de libro digital que se ha convertido en un estándar de facto por la gran implantación que está teniendo en todos los dispositivos de lectura digitales. En realidad es un archivo comprimido (ZIP) que contiene tres documentos XML que son los que especifican la estructura y contenido del documento.
- **DITA**. *Darwin Information Typing Architecture*, se utiliza para producir documentos técnicos y está siendo cada vez más popular. Tiene capacidad para incluir numerosos metadatos y para poder adaptarse a las necesidades de las entidades que utilicen el lenguaje. Permite dividir en temas reutilizables los documentos.

- **MathML**. Pensado para representar documentos con expresiones matemáticas.
- **ODF** u **OD**. ***Open Document for Office Applications***. Es un formato abierto que pretende ser un estándar como formato de intercambio entre documentos de oficina (hojas de cálculo, procesadores de texto, presentaciones, diagramas,…). Su popularidad aumentó notablemente cuando fue adoptado por el software **Open Office**, actualmente perteneciente a la empresa **Oracle**.
- **OSDF**. ***Open Software Description Format***, basado en especificaciones de las empresas **Marimba** y **Microsoft**, sirve para describir la tecnología y los componentes con los que se ha desarrollado un determinado software, a fin de facilitar el proceso de instalación.
- **OOXML. *Office Open XML***. Formato rival del anterior, propiedad de Microsoft y utilizado como formato XML para el software ***Microsoft Office***. Pretende también ser un estándar.
- **RDF**, ***Resource Description Format,*** Sirve para desarrollar documentos que describan recursos. Se trata de un proyecto ya antiguo para definir modelos de metadatos. Se basa en los modelos conceptuales como el modelo **entidad/relación** o los **diagramas de clases**, aunque actualmente se utiliza fundamentalmente para describir recursos web.
- **SMIL**. ***Synchronized Multimedia Integration Language***, Lenguaje sincronizado de integración multimedia. Utilizado para producir presentaciones de TV en la web, fundamentalmente.
- **SOAP**. ***Simple Object Access Protocol***. Protocolo estándar de comunicación entre objetos utilizado para comunicar con Servicios Web.
- **SVG**. ***Scalable Vector Graphics,*** gráficos de vectores escalables. Permite definir imágenes vectoriales pensadas para ser publicadas en una página web.
- **VoiceXML**. Se utiliza para representar diálogos vocales.
- **WSDL**. ***Web Services Description Language***. Lenguaje para definir Servicios Web.
- **XHTML**. Versión del lenguaje de creación de páginas web, **HTML**, que es compatible con las normas XML.

(1.4.4) usos de XML

contenido web

XML podría pasar a reemplazar a HTML como el formato en el que se escriben las páginas web. Es de hecho una de sus vocaciones ya que ofrece una sintaxis más rígida (que es ventajosa ya que facilita su aprendizaje), una escalabilidad (que permite que jamás se quede obsoleto el lenguaje) y una serie de tecnologías relacionadas más poderosas.

La razón del fracaso de HTML como estándar se debe a que cada navegador impone etiquetas propias, es decir que no hay un estándar real. Con XML no es posible esta situación, ya que la propia naturaleza del lenguaje no la hace posible ya que todo XML tendrá un documento de validación conocido por los navegadores que especifica exactamente qué elementos y de qué forma se pueden utilizar en el documento.

intercambio de información entre aplicaciones

El hecho de que XML almacene información mediante documentos de texto plano, facilita que se utilice como estándar, ya que no se requiere software especial para leer su contenido: es texto y es entendible por cualquier software.

computación distribuida

Se trata de la posibilidad de utilizar XML para intercambiar información entre diferentes computadoras a través de las redes. Las ventajas de XML están relacionadas con el hecho de que con él se crean documentos inocuos (no pueden contener código maligno como virus o espías), con lo que la seguridad de esos sistemas es total.

Es decir un documento XML no puede contener virus o software espía. Las máquinas enrutadoras de las redes no tienen ningún problema en encaminar archivos XML al ser texto. Entienden que no es una amenaza.

información empresarial

XML es un formato que tiene cada vez más importancia para generar documentos empresariales por la facilidad de estructurar los datos de la forma más apropiada para la empresa. Un documento XML se parece mucho a una pequeña base de datos, con la ventaja de que es muy fácil darle formato de salida por pantalla o impresión.

(1.4.5) tecnologías relacionadas con XML

XML posee un gran número de tecnologías para dar funcionalidad, presentación o integración con otros lenguajes. Las más importantes son:

- **DTD**. ***Document Type Definition***, definición de tipo de documento. Es un lenguaje que permite especificar documentos cuyas reglas han de cumplir los documentos XML a los que se asocien. Es decir, permite crear **documentos de validación** para archivos XML.
- **XML Schema**. La función que cumple esta tecnología es la misma que la anterior, la diferencia está en que los documentos XML Schema poseen una sintaxis 100% XML, por lo que es un formato orientado a suplir al anterior.
- **Relax NG**. Otro formato de definición de validaciones para documentos XML. Es una alternativa a las dos anteriores y la que tiene un formato más sencillo.
- **Namespacing, espacios de nombres**. Permite conseguir nombres de elementos que carecen de ambigüedad: es decir nombres únicos dentro de los documentos XML.
- **XPath.** Lenguaje de consulta que permite seleccionar o acceder a partes de un documento XML.
- **CSS.** ***Cascade StyleSheet***. Hojas de estilo en cascada. Permiten dar formato a los documentos XML o HTML.
- **XSLT.** Sirve para lo mismo que CSS, dar formato a un documento XML. Tene muchas más posibilidades que CSS.
- **XQuery.** Permite consultar datos de los documentos XML, manejándole como si fuera una base de datos.
- **DOM.** ***Document Object Model***, permite acceder a la estructura jerárquica del documento normalmente para utilizarla dentro de un lenguaje de programación

- SAX. *Simple API for XML*, permite el uso de herramientas para acceder a la estructura jerárquica del documento XML a través de otro lenguaje; se usa mucho en Java.
- XForms. Formato de formularios de introducción de datos.
- XLink. Permite crear hipervínculos en un documento XML.
- XPointer. Semejante al anterior, especifica enlaces a elementos externos al documento XML.

(1.5) software para producir XML

En principio XML se puede escribir desde cualquier editor de texto plano (como el bloc de notas de Windows o el editor vi de Linux). Pero es más interesante hacerlo con un editor que reconozca el lenguaje y que además marque los errores en el mismo.

De hecho el software necesario es el siguiente:

(1) Un editor de texto plano para escribir el código XML. Bastaría un editor como el bloc de notas de Windows o el clásico vi de Linux; o las opciones de editores capaces de colorear el código como emacs, Notepad++ o SublimeText.

(2) Un analizador sintáctico o parser, programa capaz de entender y validar el lenguaje XML. Apache Xerces es quizá el más popular validador de documentos XML.

(3) Un procesador XML que sea capaz de producir un resultado visual sobre el documento XML. Un simple navegador puede hacer esta función, pero cuando se aplican formatos visuales sobre el documento XML (como los creados mediante XSL) entonces hace falta un software especial que convierta los datos a la forma final visible por el usuario. Apache Xalan y Saxon son los dos procesadores más conocidos

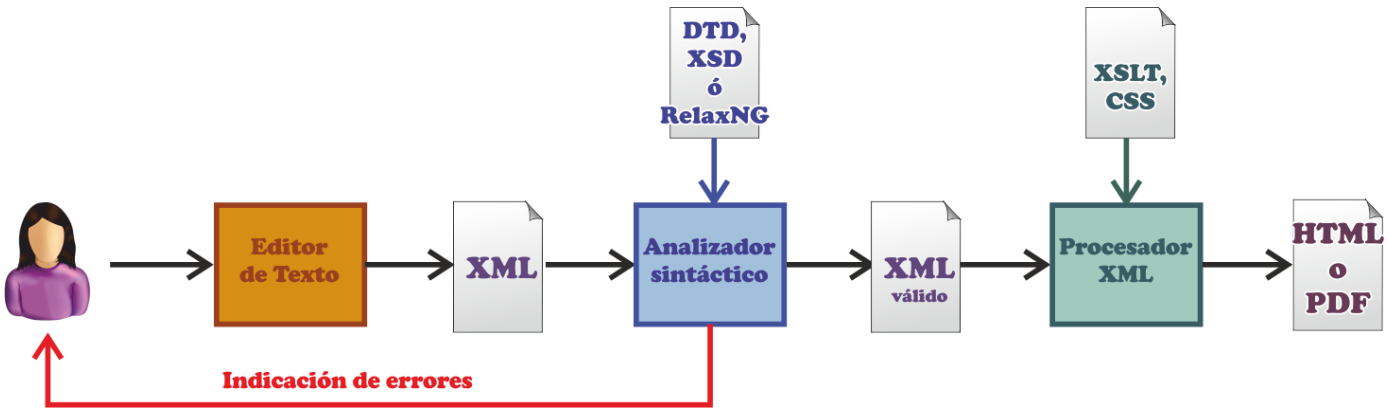

Ilustración 1, Proceso completo de funcionamiento productivo de un XML

Hay entornos de trabajo que incluyen todas esas prestaciones dentro del mismo paquete software.

(1.6) funcionamiento de XML

(1.6.1) cuestiones básicas

Los documentos XML en definitiva son documentos de lenguajes de marcas, donde hay texto normal y etiquetas (marcas) que permiten clasificar dicho texto indicando su significado.

Las etiquetas en XML se deciden a voluntad, no hay una serie de etiquetas que se pueden utilizar; de hecho la función de XML es definir tipos de documentos etiquetados.

Ejemplo de XML:

```
<persona>
   <nombre>Jorge</nombre>
   <apellido>Sánchez</apellido>
</persona>
```

El código es similar a HTML, sólo que las etiquetas se deciden según nos interese. Pero el funcionamiento el mismo:

- Las etiquetas tienen cierre que se indica con el signo / antes del nombre de la etiqueta
- Las etiquetas afectan al texto (y otras etiquetas) que están entre la apertura y el cierre

Por otro lado XML llama **elemento** a las etiquetas; mejor dicho, elemento sería tanto la etiqueta como lo que contiene.

En cualquier caso las normas son:

- Las etiquetas sirven para indicar elementos. El nombre de la etiqueta se indica entre los símbolos < y >
- Las etiquetas se cierran indicando </ seguido del nombre de la etiqueta que se está cerrando
- XML distingue entre mayúsculas y minúsculas siendo buena práctica escribir las etiquetas en minúsculas
- Se pueden espaciar y tabular las etiquetas a voluntad. Es buena práctica que un elemento interno a otro aparezca en el código con una sangría mayor (por eso en el ejemplo anterior *nombre*, que está dentro de *persona*, aparece sangrado).
- Los comentarios en el código se inician con los símbolos <!-- y terminan con -->
- Según la **W3C** (organismo de estandarización de XML), el texto en un documento XML debe de estar codificado en **Unicode** (normalmente **UTF-8**)

(1.6.2) estructura de un documento XML

Los documentos XML se dividen en:

- **Prólogo**. Se trata de la primera zona del documento y sirve para describir qué tipo de documento es. Es similar al apartado **head** de **HTML**. Puede contener
  - **Declaración del documento**, que permite indicar el tipo de documento XML que es.
  - **Instrucciones** para el procesado del documento
  - **Comentarios**
  - **Indicación del documento DTD, XSD o Relax NG** para comprobar si el mismo es válido según las reglas impuestas por dicho documento.
  - **Indicación de otros documentos que afectan al actual**, como por ejemplo los documentos **XSLT** que permiten especificar la forma en la que el documento se debe mostrar en pantalla.
- **Elemento raíz**. Todo el contenido del documento debe de estar incluido en el llamado elemento raíz, se trata de un elemento obligatorio que se abre tras el prólogo y se debe cerrar justo al final. De este modo cualquier elemento está dentro del elemento raíz. Contiene:
  - **Más elementos**
  - **Atributos**
  - **Texto normal**
  - **Entidades**
  - **Comentarios**

(1.6.3) reglas para los nombres

En XML los elementos, atributos,…. tienen un nombre (más correctamente llamado **identificador**), el cual debe cumplir estas reglas

- En XML se distingue entre mayúsculas y minúsculas, por lo que hay que tener cuidado al utilizar el nombre desde otro punto del documento
- Deben comenzar por una letra, y después le seguirán más letras, números o el signo de subrayado o guión bajo.
- No pueden empezar con la palabra XML ni en mayúsculas ni en minúsculas ni en ninguna combinación de mayúsculas ni minúsculas.

(1.6.4) elementos del prólogo

declaración XML

Se trata de la primera línea de un documento XML e indica el tipo de documento XML que es (y así poder validar el mismo). En realidad es opcional, pero es muy recomendable. Es:

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

Indica la versión XML del documento y la codificación (utf-8 es la habitual).

instrucciones de procesamiento

Un documento XML puede incluir instrucciones de este tipo para indicar un documento para validar el XML, darle formato,... u otras funciones. Por ejemplo:

```
<?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>
```

Esta instrucción asocia un documento **xsl** al documento XML para poder darle un formato de salida (para especificar la forma en la que los datos se muestran por pantalla por ejemplo).

(1.6.5) comentarios

Como se ha indicado antes comienzan con el símbolo **<!--** y terminan con **-->**. Dentro puede haber cualquier texto que se utiliza con fines explicativos o de documentación del código.

Los comentarios no pueden meterse dentro de la etiqueta de un elemento, ni tampoco puede contener etiquetas tanto de apertura como de cierre.

(1.6.6) elementos

Son la base del documento XML. Sirven para dar significado al texto o a otros elementos o también para definir relaciones entre distintos elementos y datos.

Hay una confusión entre lo que es un elemento y lo que es una etiqueta. En este caso por ejemplo:

```
<nombre>Jorge</nombre>
```

- `<nombre>` Es un etiqueta de apertura
- `</nombre>` Es una etiqueta de cierre
- `<nombre>Jorge</nombre>` Es un elemento (el elemento ***nombre***)
- `Jorge` es el contenido del elemento

El contenido de un elemento puede contener simplemente texto:

```
<descripción>
    Producto con precio rebajado debido a su escasa demanda
</descripción>
```

O puede contener otros elementos (o ambas cosas). En este el elemento *persona* consta de un elemento *nombre* y otro *apellido*.

```
<persona>
    <nombre>Jorge</nombre>
    <apellido>Sánchez</apellido>
</persona>
```

Los elementos se deben abrir y cerrar con la etiqueta que sirve para definir el elemento; siempre se debe cerrar el último elemento que se abrió. Es decir es un error:

```
<persona>
    <nombre>Jorge</nombre>
    <apellido>Sánchez</persona>
</apellido>
```

Puede haber incluso elementos vacíos:

```
<casado></casado>
```

En este caso se pueden cerrar en la propia etiqueta de apertura:

```
<casado />
```

(1.6.7) atributos

Se definen dentro de las etiquetas de apertura de los elementos. Se indica su nombre seguido del signo = y del valor (entre comillas) que se le da al atributo. Ejemplo:

```
<persona complejidad=”alta”>
    <nombre>Jorge</nombre>
    <apellido>Sánchez</apellido>
</persona>
```

Un elemento puede contener varios atributos:

```
<persona privacidad=”alta” tipo=”autor”>
    <nombre>Jorge</nombre>
    <apellido>Sánchez</apellido>
</persona>
```

(1.6.8) texto

El texto como se comentó antes está siempre entre una etiqueta de apertura y una de cierre. Eso significa que todo texto es parte de un elemento XML.

Se puede escribir cualquier carácter Unicode en el texto, pero no es válido utilizar caracteres que podrían dar lugar a confusión como los signos separadores < y > por ejemplo

(1.6.9) CDATA

Existe la posibilidad de marcar texto para que no sea procesado como parte de XML, eso se consigue colocándolo dentro de un elemento CDATA. Formato:

- <! [CDATA [texto no procesable...]]>

Esto permite utilizar los caracteres < y > por ejemplo y no serán considerados como separadores de etiquetas.

Ejemplo:

```
<?xml version="1.0"?>
<documento>
   <título>Prueba</título>
   <ejemplo>
         <![CDATA[
               En HTML la negrita se escribe: <strong>
         ]]>
   </ejemplo>
</documento>
```

En el ejemplo, los símbolos < y > no se toman como una etiqueta XML, sino como texto normal.

Otro uso de CDATA es colocar dentro de este elemento código de lenguajes de scripts como Javascript para que no sean interpretados como parte de XML.

(1.6.10) entidades

Las entidades representan caracteres individuales. Se utilizan para poder representar caracteres especiales o bien caracteres inexistentes en el teclado habitual. Se trata de códigos que empiezan con el signo & al que sigue el nombre de la entidad o el número Unicode del carácter que deseamos representar.

En XML hay definidas cinco entidades:

- > Símbolo >
- < Símbolo <
- & Símbolo &
- " Símbolo "
- ' Símbolo '

También podemos representar caracteres mediante entidades con número. De modo que el ñ representa a la letra ñ (suponiendo que codificamos en Unicode, que es lo habitual). El número puede ser hexadecimal por ejemplo para la eñe de nuevo, sería ñ

(1.6.11) jerarquía XML

Los elementos de un documento XML establecen una jerarquía que estructura el contenido del mismo. Esa jerarquía se puede representar en forma de árbol.

Así por ejemplo el archivo XML:

```
<?xml version="1.0"?>
<documento>
   <título>Apuntes de XML</título>
   <autor>Jorge Sánchez</autor>
   <fecha_pub>
         <día>18</día>
         <mes>Enero</mes>
         <año>2009</año>
   </fecha_pub>
</documento>
```

Gráficamente de forma jerárquica se podría expresar así:

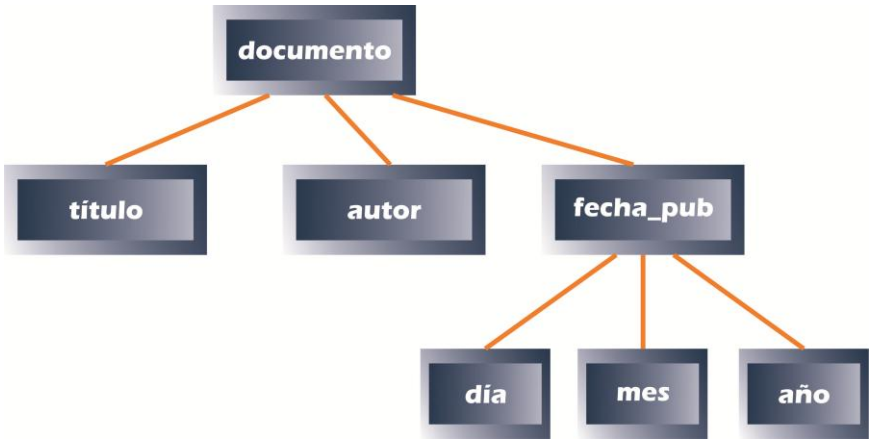

Ilustración 2, Estructura jerárquica de un documento XML

(1.7) XML bien formado

Se habla de XML *bien formado* (**well formed**) cuando cumple reglas que facilitan su legibilidad y además permiten cumplir los objetivos formales de la edición de documentos XML.

Debemos tomar las reglas para producir XML bien formado como reglas obligadas, de hecho los analizadores (*parsers*) de XML indicarían error en caso de no cumplirlas. Es decir, un documento XML bien formado es un **documento analizable** (pero no significa que sea **válido** que es un concepto que se explicará en la siguiente unidad).

(1.7.1) reglas generales

- Dentro del texto común no se pueden utilizar los símbolos de mayor (**>**), menor (**<**), ampersand (**&**) ni las comillas simples o dobles. Se deben de utilizar entidades o deben estar incluidos en una sección CDATA
- En principio en el texto normal, los símbolos de separación de caracteres como espacios en blanco, tabuladores y saltos de línea, no se tienen en cuenta, se ignoran Pero sí es posible que sean significativos en algunos elementos (si así se indica en su documento de validación). No tener en cuenta los espacios, significa que en el texto contenido en un elemento aunque dejemos 25 espacios, se entenderá que hemos dejado uno solo.

  En cualquier caso todos los caracteres escritos en el documento XML forman parte del mismo, será una cuestión posterior si se tienen en cuenta o no para presentar los datos del documento XML en pantalla o impresión.

(1.7.2) reglas para los elementos

- Se deben cerrar primero las etiquetas de los últimos elementos abiertos.
- Todos los elementos poseen etiquetas de apertura y de cierre; es decir toda etiqueta que se abra se debe de cerrar. Si la etiqueta no tiene contenido, se debe cerrar en la propia etiqueta al estilo:

```
<br />
```

- Todos los documentos XML deben de tener un único elemento raíz
- Los nombres de los elementos comienzan con letras y pueden ir seguidos de letras, números, guiones o de puntos (los guiones y los puntos no son muy recomendables)
- Los nombres de los elementos no pueden comenzar con el texto **xml** tanto en minúsculas como en mayúsculas o combinando ambas.

(1.7.3) reglas para los atributos

- Los nombres de atributos siguen las mismas normas que los nombres de los elementos
- Los atributos sólo se pueden colocar en etiquetas de apertura y nunca en las de cierre

- Los valores de los atributos deben ir entrecomillados (sin importar si se usan comillas simples o dobles). Ejemplos

```
<nombre sexo="Hombre">Antonio</nombre>
<nombre sexo='Mujer'>Sara</nombre>
<nombre sexo='Mujer">Eva</nombre> <!-- error -->
```

La última línea es incorrecta porque no se pueden combinar ambas comillas.

- Todos los atributos deben de tener valor asociado. Es decir esta etiqueta (válida en HTML) no sería válida en XML:

```
<hr noshade>
```

Debería ser, por ejemplo, así:

```
<hr noshade="noshade">
```

(1.7.4) validar documentos XML

Para comprobar si un documento XML está bien formado, necesitamos un software de validación. Posibilidades:

- **Validadores en línea.** Se trata de páginas web a las que las indicamos la dirección de Internet (es decir, la URL) en la que se encuentra el documento XML a comprobar, o bien copiamos el código XML a comprobar. Algunas:
  - **validator.w3.org** Es el validador de la organización que estandariza XML (la W3C). Se basa en la librería libxml2.
  - **www.xmlvalidation.com** Otro validador muy popular.
- **Validadores offline**. Es decir, validadores que son programas a descargar. Sin duda el más popular hoy día es **xmllint**, basado en la librería **libxml2** creada para el lenguaje C. Para utilizarlo:
  - **(1)** En el caso de Windows, hay que descargar la librería **libxml2** de la dirección ftp://ftp.zlatkovic.com/libxml/ y ahí descargar loszip necesarioselegir el archivo ZIP de la librería con la última versión (en el momento de escribir estas líneas es el archivo ***libxml2-2.7.8.zip***)
  - **(2)** Descomprimir el archivo en el sitio deseado y asignar en la que viene dicho validador (lo normal es descargar un archivo ZIP en el caso de Windows y descomprimirlo).

(1.8) espacios de nombres

(1.8.1) el problema de los nombres de elementos

Puede ocurrir que cuando se manejan documentos XML puede ocurrir que diferentes XML que tengamos, utilicen las mismas etiquetas. Aunque el contexto sería distinto, tendríamos un problema si manejamos ambos documentos con el mismo software, ya que el analizador, no sabría cómo manejar ambas etiquetas iguales.

Los espacios de nombres (***namespacing*** en inglés) evitan el problema indicando en cada etiqueta una código que sirve para indicar el contexto de cada etiqueta y así diferenciar las que son iguales. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <title>Documento de prueba</title>
  <content>
    <html>
      <head>
        <title>Titulo HTML</title>
      </head>
      <body>
        Texto del documento
      </body>
    </html>
  </content>
  <author>Jorge</author>
</document>
```

En el ejemplo anterior se usan etiquetas en inglés para el documento (algo muy habitual en el mundo empresarial) y eso hace que la etiqueta ***title*** se repita en contextos distintos, el primero es para poner un título genérico al documento (y es una etiqueta de la empresa en cuestión) y la segunda se corresponde a la etiqueta ***title*** del lenguaje HTML (o mejor XHTML).

(1.8.2) diferenciar elementos

La solución es anteponer al nombre de la etiqueta un nombre que indique el propietario de la misma, por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <jorge.title>Documento de prueba</jorge.title>
  <content>
    <html>
      <head>
        <html.title>Titulo HTML</html.title>
      </head>
      <body>
        Texto del documento
      </body>
    </html>
  </content>
  <author>Jorge</author>
</document>
```

Ese prefijo diferenciador es el espacio de nombres al que pertenece la etiqueta, pero usado así tendríamos el problema de que con un sufijo tan corto, se podría repetir. Por ello una solución es indicar la URL de la entidad responsable de la etiqueta:

```
<?xml version="1.0" encoding="UTF-8"?>
< www.jorgesanchez.net.document>
  <www.jorgesanchez.net.title>
    Documento de prueba
  </www.jorgesanchez.net.title>
  <www.jorgesanchez.net.content>
    <www.w3c.org.html>
      <www.w3c.org.head>
        <www.w3c.org.title>
          Titulo HTML
          </www.w3c.org.title>
      </www.w3c.org.head>
      <www.w3c.org.body>
        Texto del documento
      </www.w3c.org.body>
    </www.w3c.org.html>
  </www.jorgesanchez.net.content>
  <www.jorgesanchez.net.author>
    Jorge
  </www.jorgesanchez.net.author>
</document>
```

Pero el documento quedaría muy poco legible. Por ello se aplican espacios de nombres, de modo que se asigna una URL a un prefijo de etiqueta; de este modo cada vez que el documento se utiliza el prefijo, se sabe que se refiere a la URL indicada (las URLs son únicas).

Esas URL es en realidad una **URI** (***Universal Resource Identifier***) un identificador único de recurso, de modo que la raíz de la **URI** es el dominio universal (Internet) de la empresa y a él se añade la ruta al recurso. Así si hemos definido documentos XML cuyo elemento raíz es ***document***, la URI relacionada con el espacio de nombres del documento sería por ejemplo www.jorgesanchez.net/document.

(1.8.3) uso del atributo xmlns

Todas las etiquetas en XML pueden hacer uso del atributo **xmlns** (***xml namespacing***) que permite asignar un espacio de nombres a un prefijo en el documento dentro del elemento en el que se usa el espacio de nombres. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document      xmlns:jorge="http.//www.jorgesanchez.net/document"
               xmlns:html="htp://www.w3c.org/html">
  <jorge:title>
    Documento de prueba
  </jorge:title>
  <jorge:content>

  </jorge:content>
  <jorge:author>
    Jorge
  </jorge:author>
</document>
```

En el ejemplo se usa el prefijo *jorge* para indicar etiquetas del espacio de nombres *www.jorgesanchez.net/document* y *html* para el espacio de nombres de HTML.

(1.8.4) espacios de nombre por defecto

En el caso de que las etiquetas, mayoritariamente, en un documento pertenezcan a un mismo espacio de nombres, lo lógico es indicar el espacio de nombres por defecto. Eso se hace sin indicar prefijo en el atributo **xmlns**. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document      xmlns ="http.//www.jorgesanchez.net/document"
               xmlns:html="http://www.w3c.org/html">
  <title>
    Documento de prueba
  </title>
  <content>
    <html:html>
      <html:head>
        <html:title>
          Titulo HTML
          </html:title>
      </html:head>
      <html:body>
        Texto del documento
      </html:body>
    </html:html>
  </content>
  <author>
    Jorge
  </author>
</document>
```

Las etiquetas sin prefijo se entiende que pertenecen al espacio de nombres *www.jorgesanchez.net/document*, para las del otro espacio se usa el prefijo.

(1.8.5) uso de espacios de nombres en etiquetas interiores

El atributo **xmlns** no tiene por qué utilizarse en el elemento raíz, se puede posponer su declaración en el primer elemento que pertenezca al espacio de nombres deseado. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document      xmlns ="http.//www.jorgesanchez.net/document">
  <title>
    Documento de prueba
  </title>
  <content>
    <html:html xmlns:html="http://www.w3c.org/html">
      <html:head>
        <html:title>
          Titulo HTML
          </html:title>
      </html:head>
```

```
        <html:body>
          Texto del documento
        </html:body>
      </html:html>
    </content>
    <author>
      Jorge
    </author>
</document>
```

(1.8.6) uso de espacios por defecto en etiquetas interiores

Un documento puede declarar espacios por defecto en etiquetas interiores lo que permite aún más versatilidad en los documentos. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<document      xmlns ="http.//www.jorgesanchez.net/document">
<!-- comienza el espacio de nombres de jorgesanchez.net -->
  <title>
    Documento de prueba
  </title>
  <content>
    <html xmlns="http://www.w3c.org/html">
   <!-- desde aquí el espacio ahora es el de html -->
      <head>
        <title>
          Titulo HTML
          </title>
      </head>

      <body>
        Texto del documento
      </body>
    </html>
   <!-- fin del espacio html, regresa el espacio jorgesanchez.net -->
  </content>
  <author>
    Jorge
  </author>
</document>
```

(1.8.7) cancelar espacios por defecto

Si se usa el atributo `xmlns=""`, entonces se está indicando (en el interior de la etiqueta en la que se use) que ese elemento y sus hijos no usan ningún espacio de nombres.