# Introducción a las tecnologías y estándares XML

**Autor**: Manuel Collado
**Revisado**: 31 Octubre 2008


---

**Contenido**

---

# 1. Lenguajes de marcado

Son lenguajes para representar información estructurada, fundamentalmente como texto. A continuación se describen las características generales de estos lenguajes, y en particular del lenguaje XML.

## 1.1 Información estructurada

Para almacenar y transmitir información hay que encontrar una forma adecuada de representación en el medio disponible como soporte. Según los casos la representación puede reflejar en mayor o menor medida la estructura de la información.

En la representación destinada a la lectura humana y en casos primitivos de transmisión de datos la información se representa de manera poco estructurada. En estas situaciones la estructura la interpreta el lector.

- Ejemplo gráfico - diagrama como imagen de mapa de bits - se requiere un proceso de reconocimiento de formas para determinar que ciertos conjuntos de puntos son círculos, rectángulos, líneas de conexión, etc.
- Ejemplo de texto - documento con formato para impresión - se requiere un proceso de interpretación para determinar que determinados fragmentos de texto son títulos de sección, párrafos, tablas de datos, etc.

Cuando la información va destinada a ser procesada por un agente informático, lo mejor es representar explícitamente su estructura para facilitar su interpretación.

- Ejemplo gráfico - diagrama codificado (metafile o similar)
- Ejemplo de código - lenguajes formales de programación
- Ejemplo de texto - documento con marcado de contenido

En muchos casos la representación explícita, codificada, de la información la hace poco legible para las personas. La presentación para la lectura humana requiere transformar o complementar la codificación de la estructura con un estilo de presentación visual que facilite su lectura. El diseño de muchas aplicaciones interactivas aplica el principio de separación de contenido y presentación. La información se representa y procesa internamente con una codificación explícita de su estructura, y se visualiza en pantalla o se imprime destacando los elementos de la estructura con estilos de presentación adecuados.

- Ejemplo - editores de código fuente - resaltan la sintaxis del código mediante colores y encolumnado de las estructuras
- Ejemplo - navegador web - traduce el marcado HTML a presentación con estilo

La representación explícita de la estructura de información del tipo texto puede conseguirse mediante un lenguaje de marcado, consistente en una colección de marcas que delimitan los elementos de la estructura y que se codifican como fragmentos de texto fácilmente distinguibles del texto correspondiente al contenido de información. Los lenguajes de marcado se utilizan tanto para representar la estructura abstracta de los datos como para codificar internamente los estilos de presentación.

- Ejemplo - marcado de estilos y estructura en procesadores de texto - troff, LaTex, rtf, etc.

Para generalizar el uso de lenguajes de marcado se han ideado metalenguajes de marcado. Estos metalenguajes consisten en una notación genérica de marcado y un lenguaje de definición de lenguajes particulares de marcado como colecciones de marcas específicas.

- Ejemplo - SGML + DTD - lenguaje particular HTML

- Ejemplo - XML + XSD - lenguaje particular XHTML

## 1.2 Antecedentes históricos

El primer metalenguaje de marcado que ha alcanzado gran difusión ha sido SGML: Standard Generalized Markup Language, establecido como estándar ISO. La notación genérica se basa en marcas de la forma:

```
<x>...</x>
```

para delimitar un elemento de información del tipo 'x'. SGML no exige que se escriba siempre la marca de cierre. En ciertos casos se puede escribir:

```
<x>...<y>...<z>...
```

para representar una secuencia de elementos de información, no anidados. Por otro lado, parte del contenido de información de un elemento se puede expresar en forma de atributos, escritos dentro de la marca inicial:

```
<x atr1=valor1 atr2=valor2 ...>
```

En SGML la definición de un lenguaje de marcado particular (colección de marcas) se hace mediante un DTD: Document Type Definition. Esta definición indica qué marcas forman parte del lenguaje, y cuál es el contenido de información de cada una, bien texto simple o una estructura delimitada por otras marcas, así como los atributos que puede tener cada marca.

Para procesar un documento con marcado SGML hay que saber cuál es el lenguaje de marcado particular. Esto puede ser conocido de antemano, o bien el mismo documento puede contener una referencia al DTD correspondiente mediante una seudomarca de declaración de la forma:

```
<!DOCTYPE nombre ...>
```

El lenguaje de marcado particular más importante basado en SGML es HTML: HyperText Markup Language, el lenguaje de marcado original de las páginas web. Este lenguaje incluye marcas para delimitar secciones (<h1>, <h2>,...), párrafos (<p>), listas de elementos (<ol>, <ul>, <li>,...), enlaces a otras páginas (<a>), etc.

Inicialmente HTML fue concebido como un lenguaje de marcado del contenido de las páginas web. Poco a poco se fue ampliando con marcas y atributos para indicar el estilo de presentación de determinados elementos, tales como tipo y estilo de letra (<font>, <b>, <i>, ...), ajuste del texto ( , <pre>, ...), etc.

Lamentablemente el empleo de marcas para especificar el estilo de presentación ha llevado a la proliferación de páginas web con un marcado deficiente de la estructura de su contenido, por ejemplo, usando marcas de estilo para los títulos de las secciones en lugar de emplear las marcas específicas <h1>, <h2>, ... Esto hace más difícil el reconocimiento y proceso del contenido de información.

Para volver a la idea original de separar lo más posible el marcado del contenido de información y la especificación del estilo de presentación se idearon las llamadas hojas de estilo en cascada CSS: Cascaded Style Sheet. En ellas se indica el estilo particular a utilizar en cada elemento del contenido, distinguiendo incluso estilos diferentes para un mismo elemento en función de la posición que ocupe en la estructura.

Conviene destacar que las notaciones formales empleadas en las DTD y CSS no usan la notación de marcado genérica SGML: <x>, sino que son lenguajes completamente diferentes.

## 1.3 Marcado XML

El metalenguaje de marcado XML: Extensible Markup Language, surge de un proceso de revisión de SGML realizado por el consorcio W3C. XML utiliza prácticamente la misma notación genérica de marcado, pero evita las irregularidades detectadas en SGML. Los elementos deben tener siempre marca de terminación <x>...</x> y estar bien anidados. Los elementos sin contenido se pueden representar de

manera abreviada <z/>. Se introducen las llamadas instrucciones de procesamiento: <?....?>, y se mantiene la referencia al tipo de documento mediante <!DOCTYPE ...>.

Se crea un lenguaje de definición de lenguajes particulares denominado XSD: Xml Schema Definition. Este lenguaje es más completo que DTD, y se describe mediante el mismo marcado genérico XML. Se introducen los llamados espacios de nombres (*namespaces*), que facilitan la combinación en un mismo documento de marcas correspondientes a varios lenguaje de marcado particulares.

XML ha servido para definir un gran número de lenguajes de marcado particulares, tales como:

- XHTML: revisión de HTML para adaptarlo a XML
- SVG: descripción de gráficos vectoriales
- DocBook: esquema general de documentos
- MathML: descripción de fórmulas matemáticas
- ... y otros miles de lenguajes ...

Aunque la notación XML puede servir para cualquier clase de marcado, XML se ha concebido desde el principio en un contexto de separación entre contenido y forma de presentación. Por ello se crea al mismo tiempo un lenguaje para convertir el contenido abstracto de un documento XML en una forma concreta de presentación con el estilo adecuado. El lenguaje de estilo asociado a XML es XSL: eXtensible Stylesheet Language, basado en la misma notación genérica XML. En realidad XSL es una combinación de varios estándares:

- XPath: para referenciar nodos o conjunto de nodos en un documento
- XSLT: XSL Transformations - convierte a otro XML, o bien a HTML, que se visualiza, o bien a texto puro
- FO: Formatting Objects - lenguaje de descripción de páginas

Para facilitar la transición de SGML a XML (y en particular la de HTML a XHTML), XML no impone XSD como único lenguaje de descripción de esquemas. En concreto se mantiene la notación DTD como parte integral del estándar XML. Además otras organizaciones distintas de W3C han definido algún otro lenguaje de definición de esquemas tal como RelaxNG, algo más fácil de usar que XSD.

También se admiten las hojas de estilo en cascada CSS como alternativa a XSL. En concreto en XHTML se suprimen muchas de las marcas y atributos para descripción de estilo de HTML, y se sustituyen por un único atributo general de estilo cuyo contenido usa la notación CSS.

XML es, globalmente, un conjunto de estándares, además del de la notación XML propiamente dicha. Ya se han mencionado algunos de ellos: XPath, XSLT, etc. Otros estándares interesantes son, por ejemplo:

- XLink, XPointer - referencia a fragmentos de documentos
- XQuery - lenguaje de consulta
- RDF: Resource Description Framework - descripción de recursos en la web
- ... etc. ...

En la actualidad existe un gran número de herramientas para proceso de documentos XML, así como librerías e interfaces estándar para el desarrollo de nuevas herramientas. A continuación se mencionan algunas de ellas.

Interfaces estándar:

- DOM: Document Object Model - API orientada a estructura
- SAX: Simple API for XML - Parsing orientado a eventos

Librerías basadas en las interfaces anteriores o en otras similares, invocables desde diversos lenguajes (C, C++, Java, Perl, Ada, etc.):

- libxml, libxsl
- PerlXML, AdaXML
- Expat: parser XML

Diversos procesadores XML:

- Apache: Xerces (parser), FOP (procesador XSL-FO), ...

- MSXML (Microsoft) - parser XML, procesador XSLT. Incluido en Internet Explorer 5.0 y posteriores
- Saxon, xsltproc: procesadores XSLT

Herramientas: Editores XML

- XMLnotepad (Microsoft) - editor del árbol de contenido
- CookTop - editor de texto XML y acceso a MSXML
- Morphon - editor WYSIWYG de XML, y editor CSS.
- XXE: Xmlmind Xml Editor - editor WYSIWYG del contenido XML
- XmlSpy - entorno completo de desarrollo XML

Como ejemplo de aplicaciones que utilizan XML de manera intensiva, se pueden citar:

- AbiWord: Procesador de texto
- OpenOffice: Paquete de ofimática
- Netscape/Mozilla: Navegador web

# 2. Estándar XML

La información descrita mediante marcado XML se organiza en objetos denominados documentos XML, almacenados como ficheros de texto. A continuación se describen los componentes básicos y se resumen las principales reglas sintácticas establecidas por el marcado XML.

## 2.1 Caracteres

XML se apoya en el estándar Unicode (o más exactamente, en el estándar ISO/IEC 10646) para la codificación de caracteres. Un documento XML es un fichero de texto que puede contener los siguientes caracteres, indicados por su código hexadecimal :

- Caracteres de control: 09 - HT (Horizontal Tab), 0A - LF (Line Feed), 0D - CR (Carriage Return) (la versión 1.1 de XML permite todos los caracteres de control excepto el carácter nulo: 01-1F)
- Caracteres ASCII: 20-7F
- Caracteres no ASCII: 80-D7FF, E000-FFFD, 10000-10FFFF

Por otra parte, el texto del documento XML puede codificarse de cualquier manera que se desee, aunque se recomienda emplear solamente sistemas de codificación reconocidos por la IANA. Sólo dos de estos sistemas de codificación, UTF-8 y UTF-16, deben ser reconocidos siempre por cualquier procesador XML. Además de ellos es frecuente usar ISO-8859-1 o ASCII. De hecho XML provee un mecanismo para poder representar cualquier documento usando sólo caracteres ASCII, aunque el documento contenga caracteres no ASCII.

Algunos caracteres se reservan para delimitar el marcado, y no pueden ser usados directamente en el contenido básico de información (texto) del documento. Estos son, en principio:

```
< > & ' " (menor que, mayor que, ampersand, apóstrofo, comillas)
```

Cuando estos caracteres son parte de los datos se escriben habitualmente como referencias a entidades. Por ejemplo, usando respectivamente las siguientes formas simbólicas:

```
&lt; &gt; &amp; &apos; &quot;
```

## 2.2 Documento XML

La estructura general de un documento XML está formada por tres partes:

- Prólogo, opcional: Conteniendo una secuencia de instrucciones de procesamiento y/o declaración de tipo de documento
- Cuerpo: Un árbol único de elementos marcados, con anidamiento estricto.
- Epílogo, opcional: Conteniendo una secuencia de instrucciones de procesamiento

Además puede haber comentarios en cualquier parte.

Intuitivamente, el contenido de información del documento es el cuerpo. El prólogo y el epílogo sirven para facilitar la interpretación del documento. El documento completo es también una estructura en árbol. Para distinguir entre el cuerpo y el documento completo se usan los términos:

- Document entity (o Document root) - se refiere a todo el documento
- Document element - se refiere al cuerpo

## 2.3 Nombres

En XML se utilizan nombres que deben estar formados de la siguiente manera:

- Inicial : letra _ : (letra, subrayado, dos puntos)
- Resto: letra _ : - . (lo mismo más: guión, punto)
- Se distinguen mayúsculas y minúsculas

Un nombre simple sólo contiene letras, subrayado y guiones. Los caracteres punto y dos puntos se usan en nombres cualificados.

## 2.4 Elementos

Son fragmentos de información delimitados por marcas, de la siguiente manera:

- Marca inicial: <x ....>
- Contenido: texto u otros elementos.
- Marca final: </x>

El contenido del elemento puede incluir, a su vez:

- Referencia a caracteres
- Referencia a entidades
- Secciones CDATA

## 2.5 Comentarios

Un documento XML puede contener anotaciones en forma de comentario. Los comentarios no son parte del contenido de información del documento, y pueden ser ignorados por los procesadores XML. Los comentarios se escriben como

```
<!-- ...texto del comentario... -->
```

El texto de un comentario no puede contener la secuencia `--`.

## 2.6 Instrucciones de procesamiento

Son directivas que pueden ser interpretadas por los procesadores XML. Dependiendo del procesador, se interpretarán determinadas instrucciones de procesamiento, pero otras no.

El formato de una instrucción de procesamiento es:

```
<?nombre ... texto de la instrucción ... ?>
```

El texto no tiene un formato definido. Es analizado por el procesador cuyo nombre se indica.

## 2.7 Marcas

Sirven para delimitar los elementos que componen el documento XML. Un elemento queda delimitado por una marca inicial y otra final. Si el elemento no tiene contenido, se puede escribir en forma abreviada como una sola marca. El formato de las marcas es:

- Marca inicial: <nombre atributos_opcionales>
- Marca final: </nombre>

- Elemento vacío: <nombre atributos_opcionales /> (equivale a <nombre ...></nombre>)

## 2.8 Literales

Sirven para delimitar fragmentos de texto, de acuerdo con las siguientes reglas:

- Delimitados por comillas simples o dobles: 'ejemplo' "ejemplo"
- Se puede usar la otra dentro del literal: "Roger O'Connors dijo 'Sí' al votar"
- Si hay que usar el delimitador dentro del literal se usa la referencia a entidad ' (') o " (")

## 2.9 Atributos

Son fragmentos de información que forman parte de la marca inicial de un elemento. La sintaxis es:

- <nombre_marca nombre_atributo = 'valor' nombre_atributo = "valor" ...>
- No puede haber dos atributos con el mismo nombre en la misma marca
- Los valores de los atributos se dan como literales, entre comillas o apóstrofos.

## 2.10 Contenido: Datos de caracteres

Toda la información básica contenida en el documento se representa como texto. No hay datos numéricos, binarios, lógicos, etc. Estos datos de texto se escriben según las siguientes reglas:

- Pueden contener todos los caracteres Unicode válidos en XML
- Los caracteres < y & no se pueden usar directamente - se introducen como < (<) y & (&)
- Se recomienda usar también > en forma simbólica >
- Se considera espacio en blanco el formado por los caracteres: espacio, HT, salto de línea
- El espacio en blanco es parte del valor del dato
- El salto de línea puede ser: LF, CR, CR-LF. Se convierte internamente a LF

## 2.11 Referencias a caracteres

Los caracteres se pueden escribir directamente si forman parte del conjunto de caracteres correspondiente al sistema de codificación del texto del documento. Además se pueden escribir como referencias, con el siguiente formato:

- &#NNNNN; - decimal (hasta 5 dígitos)
- &#xXXXX; - hexadecimal (hasta 4 dígitos)
- El código numérico (decimal o hexadecimal) corresponde al código Unicode

## 2.12 Referencias a entidades

Una entidad es un fragmento de información, definido como un valor constante, al que se puede hacer referencia mediante un nombre, de la siguiente manera:

- &nombre;
- Sólo hay 5 entidades predefinidas: < (<) > (>) & (&) ' (') " (")
- Otras entidades deben ser definidas para poder usarlas. Por ejemplo, las de HTML.

## 2.13 CDATA

Una sección CDATA es un texto literal que puede contener directamente incluso caracteres de marcado reservados, sin necesidad de escribirlos como referencias. La sintaxis es:

**<!CDATA[ ... *texto con caracteres especiales < > ' & " ...* ]]>**

La combinación `]]>` no puede aparecer dentro de una CDATA, ni tampoco directamente como texto fuera de ella. En este caso debería ser representada como `]]>`.

## 2.14 Prólogo: Declaración XML

La declaración XML es una instrucción de procesamiento especial. Es opcional. Cuando existe debe ser la primera instrucción del prólogo. Su formato es:

- <?xml version="1.0" encoding='utf-8' standalone="yes"?>
- version: atributo obligatorio, sólo puede valer '1.0' (por ahora)
- encoding: atributo opcional, recomendado, debe ser un valor IANA válido, por defecto 'utf_8' o 'utf-16'
- standalone: atributo opcional, puede valer 'yes' o 'no'. Indica si el documento puede ser procesado sin necesidad de acceder a definiciones externas.

Los nombre 'xml', 'version', ... deben escribirse en minúsculas. Los valores pueden escribirse en minúsculas o mayúsculas ('UTF-8' = 'utf-8')

## 2.15 Prólogo: Declaración de tipo de documento

La declaración de tipo de documento es opcional. Se escriben en el prólogo, y tiene un formato especial, distinto de las marcas y de las instrucciones de procesamiento. Esta declaración puede contener una indicación explícita del lenguaje particular de marcado correspondiente al documento (definido externamente), y también la declaración directa de ciertos elementos del lenguaje de marcado (definidos internamente). El formato es uno de los siguientes:

- <!DOCTYPE nombre-elemento PUBLIC public-ID system-ID ... >
- <!DOCTYPE nombre-elemento SYSTEM system-ID ... >
- nombre-elemento es el nombre del elemento principal (elemento raíz del cuerpo)
- public-ID es un identificador asociado al lenguaje de marcado particular
- system-ID es una referencia a un DTD o XSD externo

## 2.16 Cuerpo

Está constituido por un único árbol de elementos, es decir, con una raíz única. Además de los elementos, puede contener comentarios e incluso instrucciones de procesamiento.

## 2.17 Epílogo

Es opcional, y en general se omite, ya que no está claro para qué sirve. Está pensado para contener instrucciones de procesamiento, pero resulta poco intuitivo poner estas instrucciones al final.

## 2.18 Espacio en blanco

Un documento XML puede contener espacio en blanco (caracteres de espacio, tabulación y saltos de línea), que puede ser significativo o no. El espacio en blanco no significativo puede ser ignorado, modificado o eliminado sin que cambie el significado del documento.

- El espacio en blanco que sea parte del contenido de texto de un elemento es significativo (a menos que se diga expresamente lo contrario).
- El espacio en blanco entre marcas en lugares donde no se permite contenido de texto no es significativo.
- El espacio en blanco entre las partes de una marca de comienzo no es significativo.
- El espacio en blanco en el valor de un atributo puede ser reajustado dependiendo del tipo de atributo. Por ejemplo, para atributos de tipo CDATA se elimina el espacio en blanco al comienzo y al final, y se reemplazan varios caracteres en blanco seguidos por uno solo.

## 2.19 Documento XML bien formado

Se dice que un documento XML está bien formado cuando cumple las reglas sintácticas indicadas. Los procesadores XML pueden rechazar cualquier documento que no esté bien formado.

## 2.20 Documento XML válido

Un documento XML válido es un documento que está bien formado, y además cumple con la definición de un lenguaje de marcado particular. Es decir, el cuerpo del documento tiene una estructura de

elementos compatible con el lenguaje concreto al que corresponde.

# 3. Definición de tipo de documento: DTD

Para mantener compatibilidad con SGML, el estándar XML mantiene el metalenguaje DTD de definición de lenguajes particulares de marcado. Las siglas DTD significan *Document Type Definition*, y se refieren, por tanto, a la definición de un tipo o esquema de documento.

La definición del tipo de documento puede hacerse:

- En un fichero separado, y poner la referencia en el DOCTYPE
- En el propio documento, dentro del DOCTYPE
- Con una combinación de ambos métodos

El lenguaje DTD permite definir elementos, atributos, entidades y notaciones (estas últimas se utilizan poco).

- Los elementos configuran la estructura general de un documento XML, y se anidan unos dentro de otros formando un árbol. Para cada elemento se define su nombre y la estructura de su contenido (texto u otros elementos).
- Los atributos son fragmentos de información asociados a un elemento. Tienen nombre y su contenido es siempre texto. No pueden anidarse.
- Las entidades son similares a las *macros* de ciertos lenguajes de programación. Son fragmentos de texto constantes a los que se puede hacer referencia mediante un nombre. Sirven para simplificar la escritura de documentos y DTDs en los que aparecen repetidamente ciertos fragmentos de texto.
- Las notaciones sirven para delimitar contenido no XML dentro de un documento XML.

El formato general de una definición elemental en una DTD es:

```
<!clase parámetros ...>
```

Donde clase será ELEMENT, ATTLIST, ENTITY o NOTATION, y los parámetros dependerán de la clase de definición.

## 3.1 Definición de elementos

Los parámetros de una definición de elemento son su nombre y el esquema de su contenido. El formato de la definición puede ser uno de los siguientes:

```
<!ELEMENT  nombre  ANY >
<!ELEMENT  nombre  EMPTY >
<!ELEMENT  nombre  (expresión regular) >
<!ELEMENT  nombre  (expresión regular)repetición >
<!ELEMENT  nombre  (#PCDATA) >
<!ELEMENT  nombre  (#PCDATA | nombre | nombre ...)* >
```

La primera forma define un elemento cuyo contenido puede ser cualquiera. La segunda forma define un elemento sin contenido.

Las formas tercera y cuarta definen elementos compuestos que contienen otros elementos y cuya estructura debe ajustarse a la expresión regular que se indica. La expresión regular debe estar formada por nombres de elementos y los metacaracteres de agrupación "(" ")", de secuencia y alterativa "," "|" y de repetición "+" "?" "*". Nótese que siempre es necesario un nivel externo de paréntesis. Además estas formas de expresión no pueden contener el símbolo #PCDATA.

Las dos últimas formas definen elementos con lo que se denomina *mixed content*, formado por texto solo o entremezclado con otros elementos. El nombre especial #PCDATA (que indica contenido de texto) debe aparecer siempre al principio de la expresión, que contendrá sólo ese término o será una repetición de una alternativa simple.

## 3.2 Definición de atributos

Los parámetros de una definición de atributos son el nombre del elemento al que corresponden y los nombres y descripciones de contenido de los atributos. El conjunto de atributos de un elemento puede declararse en una sola definición, o por partes, en varias definiciones separadas. El formato de una definición de atributos es el siguiente:

```
<!ATTLIST elemento
    nombre  tipo  tratamiento_por_defecto
    nombre  tipo  tratamiento_por_defecto
    ...
>
```

- *elemento*: es el nombre del elemento al que corresponden los atributos
- *nombre*: es el nombre del atributo
- *tipo*: CDATA, (*valor* | *valor* | ... ), ID, IDREF, IDREFS, NMTOKEN, NMTOKENS
- *tratamiento_por_defecto*: #REQUIRED, #IMPLIED, #FIXED *valor_por_defecto*, *valor_por_defecto*

Un valor del tipo CDATA corresponde a un valor de texto, en general. Un valor del tipo enumerado (valor | valor | ...) especifica uno entre varios posibles *nmtoken*. Un valor del tipo ID es un nombre que debe ser único en todo el documento, y que sirve para identificar el elemento. Un valor del tipo IDREF es un nombre que debe aparecer como valor de un atributo ID en algún elemento del documento. Un valor del tipo IDREFS es una lista de IDREF separados por espacio en blanco. Un valor del tipo NMTOKEN es similar a un nombre, sin la restricción del carácter inicial. Un valor del tipo NMTOKENS es una lista de NMTOKEN separados por espacio en blanco.

Un atributo #REQUIRED debe aparecer siempre. Un atributo #IMPLIED es opcional, sin un valor por defecto. Un atributo #FIXED debe tener el valor indicado si no se omite, y si se omite se asumirá el valor indicado. Si no se indica #REQUIRED ni #IMPLIED ni #FIXED sino sólo un valor por defecto, el atributo es opcional y si se omite se asume el valor por defecto.

## 3.3 Definición de entidades

Como ya se ha dicho las entidades son valores constantes a los que se puede hacer referencia mediante un nombre. Hay dos clases de entidades:

- *General entities*: para ser usadas en el contenido del documento
- *Parameter entities*: para ser usadas en la DTD

Una *general entity* se define como:

```
<!ENTITY  nombre  valor_de_sustitución >
```

El *valor_de_sustitución* puede ser un texto literal, que a su vez puede contener referencias a otras entidades. A la entidad se hace referencia con `&nombre;` en el contenido del documento.

Una *parameter entity* se define como:

```
<!ENTITY  %  nombre  valor_de_sustitución >
```

El *valor_de_sustitución* puede ser un texto literal, que a su vez puede contener referencias a otras entidades. A la entidad se hace referencia con `%nombre;` en la DTD.

## 3.4 DTDs modulares

Es posible construir una DTD por partes, en documentos separados. Una DTD maestra puede contener parte de las declaraciones e importar otras DTD. Para ello se utiliza el mecanismo de *parameter entities*, hacendo referencia a entidades externas mediante una notación similar a un <!DOCTYPE ...> para el valor de sustitución.

```
<!ENTITY  %  nombre  PUBLIC  public-id  system-id >
<!ENTITY  %  nombre  SYSTEM  system-id >
```

```
...
%nombre;
```

Las declaraciones externas se insertan en la DTD maestra en el punto en el que se hace referencia a la entidad externa mediante `%nombre;`.

## 3.5 Otros metalenguajes para definición de tipos de documento

Además del lenguaje DTD que forma parte del estándar XML hay otros metalenguajes que también sirven para definir lenguajes de marcado particulares. Los más interesantes son quizá los siguientes:

- Esquemas XML: Denominados habitualmente con las siglas XSD (Xml Schema Definition). Usan una notación XML para describir los tipos de documento (elementos, atributos, ...). Los XSD tienen todas las posibilidades de los DTD y otras adicionales. Por ejemplo, permiten distinguir diferentes tipos de valores simples (texto, número, fecha, etc.). También permiten especificar cardinalidades en la estructura de los elementos compuestos. La notación XSD es mucho más compleja que los DTD. Es un estándar de W3C (al igual que XML)
- RELAX NG: Se pueden utilizar dos notaciones diferentes para describir los esquemas, una de tipo XML y otra más compacta similar a un lenguaje de programación. Hay una equivalencia entre ambas notaciones que permite traducir de una a la otra. RELAX NG combina el lenguaje de esquemas RELAX (REgular LAnguage description for Xml) con el lenguaje de validación TREX (Tree Regular Expressions for Xml). Las especificaciones de tipo de documento mediante RELAX NG resultan algo más manejables que los XSD, y tienen una potencia similar. Cada vez hay más utilidades que soportan RELAX NG. Ha sido desarrollado por la organización OASIS y aceptado como estándar ISO/IEC.
- Schematron: También usa notación XML. No es un lenguaje de esquemas (orientado a gramática) sino un lenguaje de validación (orientado a reglas). Permite especificar condiciones que debe cumplir o no un documento XML válido. Las condiciones pueden ser positvas (*assert*) o negativas (*report*). Las expresiones de condición están basadas en el estándar XPATH. Ha sido aprobado como estándar ISO/IEC.

Los esquemas DTD, XSD y RELAX NG pueden ser utilizados por editores XML para forzar la validez del documento durante la edición. En cambio Schematron sólo se usa para validar documentos ya construidos.

## 3.6 Espacios de nombres

Es relativamente frecuente combinar varios vocabularios de marcado en un mismo documento. Esto puede dar lugar a conflictos de nombres, ya que cada vocabulario o lenguaje particular de marcado puede haber sido preparado de manera independiente y usar nombres de marcas que también forman parte de otro vocabulario.

Para permitir la mezcla de vocabularios se ha ideado el mecanismo de espacios de nombres (*namespaces*). Cada vocabulario se asocia a un espacio de nombres separado. Los espacios de nombres se identifican mediante un designador, que debe tener el mismo formato que un URI, aunque no tiene que existir necesariamente en la red.

El hecho de usar un URI como identificador del espacio de nombres es un artificio para garantizar que no se reutilizará accidentalmente el mismo identificador para dos vocabularios diferentes. Se supone que quien asigna un identificador concreto a un espacio de nombres tiene también la autoridad para crear realmente un recurso en Internet con el URI elegido.

El uso de un espacio de nombres no implica que la defincón del vocabulario exista realmente, ni sirve para localizar la definición en el caso de que exista. Para ello se debe emplear la declaración DOCTYPE como se ha indicado anteriormente.

El siguiente ejemplo (tomado del estándar MathML) mezcla los vocabularios XHTML y MathML en una misma página web:

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

```
    <title>MathML's Hello Square</title>
  </head>
  <body>
    <p> This is a perfect square:</p>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow>
        <msup>
          <mfenced>
            <mrow>
              <mi>a</mi>
              <mo>+</mo>
              <mi>b</mi>
            </mrow>
          </mfenced>
          <mn>2</mn>
        </msup>
      </mrow>
    </math>
  </body>
</html>
```

Como puede verse, los espacios de nombres se declaran en el elemento donde se usan, usando el atributo genérico "`xmlns`". No es necesario declararlo en todos y cada uno de los elementos, ya que la declaración se propaga automáticamente a todos los elementos hijos. Los elementos que aparezcan fuera de todas las declaraciones de espacios de nombres no se consideran asociados a ningún espacio de nombres particular.

Cuando los elementos de distintos vocabularios se entremezclan de forma intensiva puede resultar engorroso tener que declarar el espacio de nombres cada vez que un elemento de un vocabulario aparece dentro de otro elemento de distinto vocabulario. Para simplificar la escritura se pueden usar prefijos cortos que identifiquen el espacio de nombres de cada elemento. En este caso el prefijo hay que ponerlo en cada elemento individual, ya que no hay propagación a los hijos. El ejemplo anterior puede reescribirse de la siguiente manera:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:m="http://www.w3.org/1998/Math/MathML">
  <head>
    <title>MathML's Hello Square</title>
  </head>
  <body>
    <p> This is a perfect square:</p>
    <m:math>
      <m:mrow>
        <m:msup>
          <m:mfenced>
            <m:mrow>
              <m:mi>a</m:mi>
              <m:mo>+</m:mo>
              <m:mi>b</m:mi>
            </m:mrow>
          </m:mfenced>
          <m:mn>2</m:mn>
        </m:msup>
      </m:mrow>
    </m:math>
  </body>
</html>
```

Es este ejemplo se ha mezclado el uso de un espacio de nombres por defecto (XHTML) que no necesita prefijo, y un espacio de nombres (MathML) con prefijo explícito. El prefijo puede elegirse arbitrariamente en cada documento, aunque existe la constumbre de usar siempre el mismo prefijo nemotécnico para cada vocabulario estándar de uso amplio.

En un mismo documento se pueden declarar cuantos espacios de nombres sean necesarios, usando un prefijo diferente para cada uno, además del espacio de nombres por defecto que no necesita prefijo.

Finalmente conviene saber que las declaraciones de espacios de nombres afectan directamente a los nombres de elementos, pero no a los nombres de atributos. Los nombres de los atributos se interpretan según el elemento en el que aparecen, sin necesidad de usar prefijos.

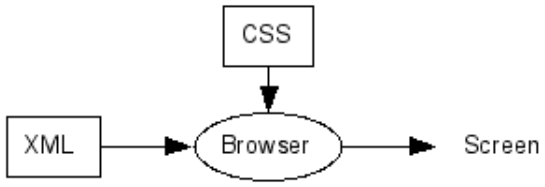
# 4. Hojas de estilo en cascada: CSS

Aunque en algunos casos puede estar preestablecida la manera de presentar el contenido de información de un documento XML (por ejemplo, para páginas web XHTML), lo habitual es independizar el marcado del contenido de la forma de presentación. Dicha forma de presentación se determina mediante lo que se llama una hoja de estilo, escrita en una notación adecuada. Un documento XML puede asociarse con diferentes hojas de estilo, obteniendo así distintas presentaciones del mismo contenido de información.

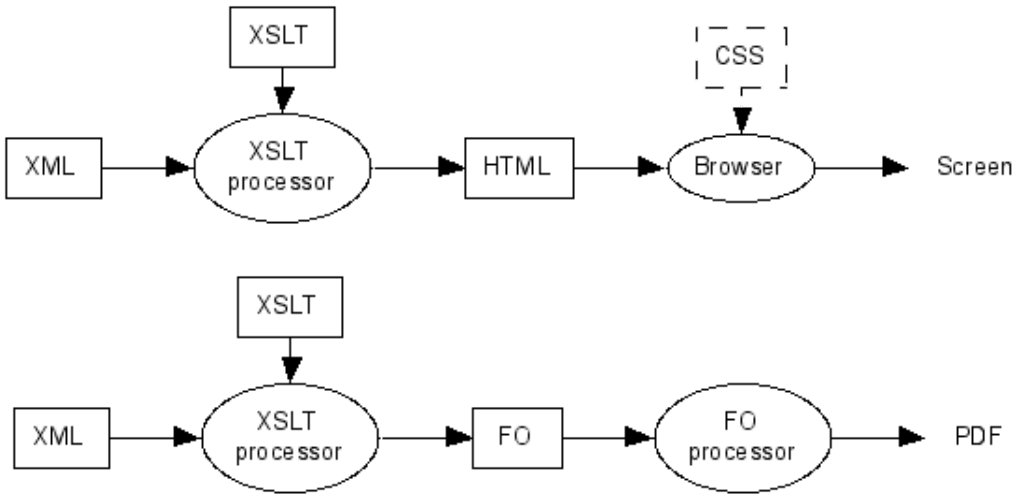
La presentación del contenido del documento mediante una hoja de estilo puede hacerse:

- Generando la presentación directamente a partir del documento XML y la hoja de estilo, o bien
- Transformando el documento XML en otro que lleve asociada una forma de presentación

Un ejemplo de lo primero es el empleo de hojas de estilo en cascada con notación CSS:




Un ejemplo de lo segundo son las hojas de estilo XSL que permiten transformar el documento en una página web HTML, o bien en un documento FO:




## 4.1 Notación CSS

Las hojas de estilo en cascada CSS permiten asignar estilos de presentación particulares a cada elemento del documento XML. La presentación final se obtiene procesando los elementos componentes del documento XML uno a uno en el orden en que aparecen, y aplicando a cada uno el estilo establecido en la hoja CSS. Las hojas CSS no premiten reordenar los elementos, ni repetirlos. Sí se puede omitir parte del contenido y añadir algún texto delante o detrás de determinados elementos.

Una hoja de estilo CSS se compone de una serie de reglas de la forma:

```
selector { propiedades_de_estilo }
```

El selector es un patrón de selección, y las propiedades de estilo son una lista de valores con nombre: `propiedad: valor;`. Por cada elemento a procesar se examinan todas las reglas y se aplican las propiedades indicadas en todos los selectores que se cumplan. Si una misma propiedad aparece en varias reglas aplicables pero con distintos valores, se establece un criterio de prioridad para determinar cuál es el valor a utilizar. La prioridad es tanto mayor cuanto más específico es el patrón de selección.

Si en la presentación se necesita una propiedad no especificada en ninguna de las reglas que se cumplen, entonces se toma del elemento que engloba al que se está procesando (herencia en cascada). Si la propiedad tampoco está definida en los elementos que lo engloban, entonces se aplica un valor por defecto.

## 4.2 Selectores

Permiten seleccionar elementos por su nombre, sus atributos, su situación respecto a otro elemento (descendiente, hijo o siguiente hermano), o por una combinación de los anteriores. Ejemplos:

```
x               elemento de tipo 'x'
*               elemento de cualquier tipo
y x             elemento 'x' descendiente de 'y'
y > x           elemento 'x' hijo directo de 'y'
y + x           elemento 'x' con hermano inmediato anterior 'y'
x:first-child   elemento 'x' sin hermano anterior
x[a]            elemento 'x' con atributo 'a'
x[a=v]          elemento 'x' con atributo 'a' de valor 'v'
x[a~=v]         elemento 'x' con atributo 'a' (lista) que incluye 'v'
x.c             sólo para HTML: equivale a x[class~=c]
```

Hay selectores especiales para insertar texto delante o detrás de un elemento. El texto se especifica en las propiedades de estilo asociadas. Ejemplos:

```
x:before        a insertar delante de un elemento 'x'
x:after         a insertar detrás de un elemento 'x'
```

Finalmente hay selectores especiales que modifican la presentación según se va navegando por el documento con el *browser* XML. Ejemplo:

```
x:hover         elemento 'x' sobre el cual está el cursor
```

Los operadores de selección pueden combinarse entre sí, y además se pueden escribir varios selectores en una misma regla, separados por comas. Ejemplo:

```
y x[a]:hover, z.c  elemento 'x' con atributo 'a', descendiente
                   de 'y', sobre el cual está el cursor, o bien
                   elemento 'z' con atributo 'class' que incluye
                   el valor 'c'
```

## 4.3 Propiedades de estilo

La lista de propiedades que se pueden definir es realmente larga, y además hay restricciones sobre qué propiedades se pueden combinar entre sí, y para qué clase de elementos. Se presentan aquí sólo algunas de las más utilizadas:

```
display: block;       representación en zona rectangular dedicada
display: inline;      se combina con el texto anterior y siguiente
display: table ...;   presentación en modo tabla
border: ...;          recuadro alrededor del elemento
margin: ...;          espacio en blanco fuera del recuadro
```

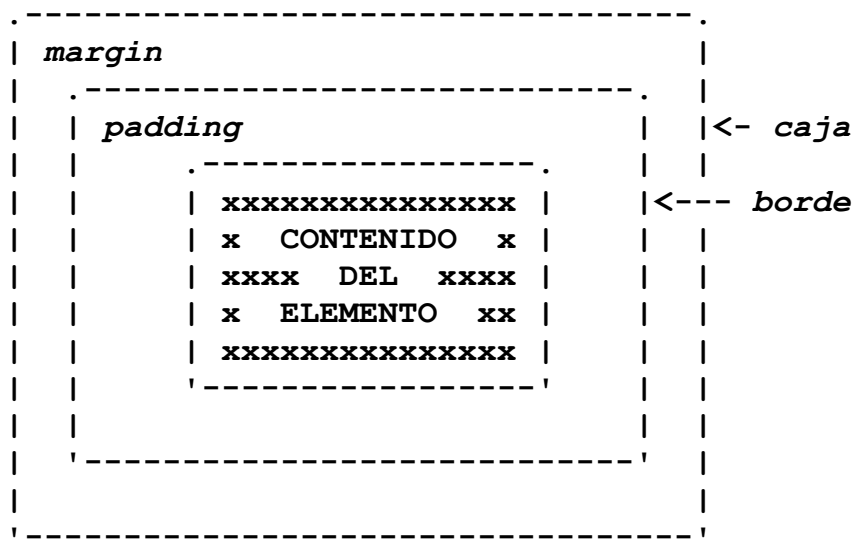
```
padding: ...;         espacio en blanco dentro del recuadro
text-indent: ...;     sangrado en la primera línea
text-align: ...;      izquierda, derecha, centrada, justificada
vertical-align: ...; superior, inferior, centrada, subíndice, ...
font-family: ...;     tipo de letra
font-size: ...;       tamaño de letra
font-weight: ...;     intensidad de letra negra
font-style: ...;      itálica, normal, ....
text-decoration: ..; subrayado, tachado, ...
color: ...;           color del texto
background-color:..; color del fondo
content: ...;         contenido a insertar en :before o :after
```

Los valores de las propiedades pueden ser palabras clave predefinidas (`block`, `center`, `red`), cadenas de caracteres (`"entre comillas"`), o valores numéricos con unidades de medida: (`3.3cm`, `120%`). Entre las unidades posibles están:

```
in          pulgadas
cm          centímetros
mm          milímetros
pt          puntos (1/72 pulgada)
px          píxeles
em          altura de la letra (fuente)
ex          altura de la letra 'x'
%           tanto por ciento respecto al valor que se heredaría
```

## 4.4 Modelo de cajas

Los elementos que se presentan en modo bloque ocuparán un espacio rectangular o caja (*box*), con márgenes (*margin*), borde (*border*) y espacio de relleno (*padding*) opcionales, tal como se indica en la figura:




## 4.5 Otros lenguajes de estilo

Prácticamente no hay alternativas a CSS como lenguaje de estilo directo. Se pueden mencionar un par de ellos, pero apenas se usan.

- DSSSL: Más que una alternativa a CSS es un precusor de XSL en la era SGML. Incluye una notación de estilo y otra de transformación. La sintaxis es de tipo LISP embebida en XML. Parece que aún se sigue usando en el mundo SGML.

- JSSS: JavaScript Style Sheets. Ideada por Netscape como alternativa a CSS. Soportada inicialmente en algunos navegadores, pero luego abandonada en favor de CSS tras el fracaso del intento de aceptación por el consorcio W3C.

# 5. Estándar XSL

XSL es un estándar de hojas de estilo especialmente creado para la notación XML. En realidad es una combinación de varios estándares:

- XPath: Notación para designar nodos o conjuntos de nodos en un documento XML
- XSLT: Lenguaje de transformación de documentos XML
- FO: Lenguaje de descripción de páginas mediante composición de objetos con formato

A continuación se describen los elementos básicos de cada uno de ellos.

# 6. Estándar XPath

El ámbito de aplicación de la notación XPath es la estructura en árbol de un documento XML. Dicha estructura contiene todo lo que hay en el documento, y no sólo la información de sus elementos y atributos. Los diferentes tipos de nodos que se pueden encontrar son:

- Raíz del documento
- Elemento
- Atributo
- Texto
- Espacio de nombres
- Instrucción de proceso
- Comentario

Una expresión XPath permite extraer uno o varios fragmentos de información de un documento XML. Los resultados posibles son:

- Conjunto o lista de nodos
- Valor booleano
- Número
- Texto

XPath incluye una notación para seleccionar nodos o grupos de nodos, y una colección de funciones para calcular valores a partir de ellos. Las expresiones se evalúan a patir de un determinado nodo que establece el contexto de evaluación.

## 6.1 Notación de selección

Un camino de selección puede ser absoluto o relativo. En el primer caso el camino se establece desde la raíz del documento. En el segundo caso se parte del nodo de contexto. El camino se compone de uno o más pasos, que se escriben separados por barras. Un camino absoluto empieza por barra, y uno relativo no. Un paso se describe con hasta tres elementos: eje, test de nodo, y predicado. Cada paso y elemento de selección selecciona un nuevo conjunto de nodos a partir del anterior, empezando por el conjunto formado por el nodo de contexto (camino relativo) o el nodo raíz (camino absoluto). La sintaxis es:

```
camino_absoluto  ::=  /camino_relativo
camino_relativo  ::=  paso/paso/...
paso  ::=  eje::test_de_nodo[predicado]
```

El eje será uno de los siguientes:

- `ancestor` (ascendiente): nodo padre, abuelo, etc.
- `ancestor-or-self`: nodos ascendientes o el mismo nodo
- `parent`: nodo padre
- `self`: el mismo nodo

- `child`: nodos hijos
- `descendant`: nodos hijos, nietos, etc.
- `descendant-or-self`: nodos descendientes o el mismo nodo
- `preceding-sibling`: hermanos anteriores (hijos anteriores del mismo padre)
- `preceding`: nodos anteriores en el orden lexicográfico
- `following-sibling`: hermanos siguientes (hijos siguientes del mismo padre)
- `following`: nodos posteriores en el orden lexicográfico
- `attribute`: atributos
- `namespace`: espacio de nombres

El test de nodo filtra la selección anterior en función del nombre y/o tipo de nodo. En las siguientes definiciones el término *elemento* se refiere a nodos elemento (si el eje lo permite), o bien a nodos del tipo particular establecido por el eje.

- *`nombre`*: elemento con ese nombre
- `*`: elemento con cualquier nombre
- `comment()`: nodo de comentario
- `text()`: nodo de texto
- `processing-instruction(`*`target`*`)`: instrucción de procesamiento con ese nombre
- `namespace:`*`nombre`*: espacio de nombres con ese nombre
- `node()`: cualquier nodo

Finalmente, el predicado es una expresión que debe cumplirse para que un nodo sea seleccionado. La expresión puede usar los operadores y funciones mencionados en el apartado siguiente.

La notación de selección tal como se ha descrito resulta muy farragosa. Por eso se ha previsto una forma abreviada de escribirla, pensada para los casos más frecuentes, como se indica en la siguiente tabla:

| **Notación extendida** | **Notación abreviada** |
|---|---|
| **child::** | ***se puede omitir*** |
| **parent::** | **..** |
| **self::** | **.** |
| **/descendant-or-self::** | **//** |
| **[position() = *n*]** | **[*n*]** |
| **attribute::** | **@** |

## 6.2 Operadores y funciones

Las expresiones utilizables en XPath se componen de operandos, operadores e invocaciones de funciones. Los elementos básicos son:

- Operandos: nodos, conjuntos de nodos, valores numéricos, booleanos y de texto
- Operadores aritméticos: `+ - div * mod`
- Operadores lógicos: `and or |`
- Operadores de relación: `= != < <= > >=`
- Caminos de selección, para extraer conjuntos de nodos
- Paréntesis, para agrupar términos: `()`
- Funciones predefinidas
  - Funciones booleanas: `not() true() false() boolean() ...`
  - Funciones numéricas: `number() sum() ...`
  - Funciones de nodos: `position() last() count() name() ...`
  - Funciones de texto: `string() concat() substring() string-length() normalize-space() ...`

Los argumentos que sean nodos o conjuntos de nodos pueden omitirse, en general. El valor por defecto es el nodo de contexto.

## 6.3 Otros estándares derivados de XPath

La notación XPath ha servido de base para definir otros lenguajes o notaciones que permiten seleccionar o localizar fragmentos de un documento XML. Con ellos se pueden construir, por ejemplo, documentos con enlaces de tipo hipertexto, documentos modulares, etc. Entre esas notaciones están:

- XPointer: es una notación para designar fragmentos de documentos XML.
- XInclude: es un vocabulario de marcado para la creación de documentos XML modulares. Es independiente del mecanismo de entidades externas que forma parte del estándar XML. Se basa en XPointer.
- XQuery: es una extensión de XPath que permite realizar consultas en un documento XML como si se tratara de una base de datos.

**XPointer**

Hay dos formas de notación XPointer: la abreviada y los esquemas. La notación abreviada selecciona un elemento a partir de su identificador (atributo de tipo ID o algo equivalente). Los esquemas adoptan la forma de una función cuyo argumento es una expresión que se evalúa de forma particular para cada esquema. El esquema `element()` lleva como argumento un identificador, o bien una secuencia de selectores por posición de cada hijo, o bien una combinación de ambos. Ejemplos:

```
element(intro)
element(/1/2)
element(intro/3/1)
```

El esquema `xpointer()` lleva como argumento una expresión XPath extendida. Las extensiones permiten designar, además de nodos, puntos entre fragmentos (es decir, lo equivalente a la posición de un cursor) o rangos de elementos o caracteres de un texto. Lamentablemente el esquema `xpointer()` todavía no es oficial.

W3C admite también esquemas propuestos por otras personas o entidades. No los considera recomendaciones oficiales, y se limita a llevar un registro para evitar conflictos de nombres. Por ejemplo, están registrados los esquema `xpath1()` y `xpath2()` que llevan como argumento una expresión XPath (versión 1 y versión2, respectivamente). Ejemplo: primer párrafo de un documento XHTML.

```
xpath1(/html/body/p[1])
```

**XInclude**

Para usar el mecanismo de XInclude hay que declarar el espacio de nombres:

```
xmlns:xi="http://www.w3.org/2001/XInclude"
```

La marca principal de este lenguaje es:

```
<xi:include  href=URI  xpointer=fragmento  parse=xml|text />
```

Con esta marca se incluye el documento referenciado con el *URI*. Puede seleccionarse sólo una parte usando el atributo `xpointer`. El documento referenciado se analiza tal como indica el atributo `parse`, bien como fragmento de marcado XML, o bien como fragmento de texto literal.

**XQuery**

Una expresión sencilla en XQuery es una expresión XPath que puede ir precedida de una referencia a un documento externo que se usará como nodo de contexto. Ejemplo:

```
doc("books.xml")/bookstore/book
```

La notación XQuery permite también expresar consultas de manera parecida a como se hace con SQL. Ejemplo:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

# 7. Estándar XSLT

Es un lenguaje para transformación de documentos XML. Las siglas XSLT significan XSL Transformations. Con él se pueden conseguir transformaciones que incluyen:

- Generar texto fijo
- Suprimir parte del contenido - seleccionar partes
- Cambiar el orden del contenido. Ordenar
- Repetir partes
- Generar información calculada
- etc. ...

## 7.1 Estructura general

XSLT se plantea como un lenguaje declarativo. Una hoja de transformación contiene una serie de plantillas que se aplican en función de un criterio de selección sobre el nodo a procesar en cada momento. En principio, el orden de las plantillas no es significativo. Cada plantilla describe una acción a realizar a partir del nodo que se está procesando.

Como se verá más adelante, en una hoja de transformación se suele mezclar el vocabulario propio de XSLT con otros vocabularios que corresponden a marcas a generar en el resultado de la transformación. Por ello es necesario declarar adecuadamente los espacios de nombres apropiados. La costumbre es declarar el vocabulario de marcas XSLT usando el prefijo "`xsl`":

```
... xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ...
```

El esquema habitual de una transformación XSLT puede ser:

```
<xsl:stylesheet ...>
  <xsl:output method="método" .../>
  ...
  <xsl:template match="/" ...>
    <xsl:apply-templates/>
  </xsl:template>
  ...
  <xsl:template match="xpath" ...>
    ....
  </xsl:template>
  ...
</xsl:stylesheet>
```

La hoja de estilo completa se especifica como:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

La especificación del formato de salida es:

```
<xsl:output method="..." encoding="..." indent="...">

method:  xml, html, text, ...
encoding:  utf-8, utf-16, iso-8859-1, ...
indent:  yes, no
```

Una hoja de transformación XSLT es un documento XML que contiene marcas de cualquier tipo entremezcladas con marcas del lenguaje específico XSLT (espacio de nombres con prefijo `xsl:`). Las marcas XSLT son interpretadas por el procesador XSLT. El resto de las marcas son copiadas literalmente a la salida cuando se procesen.

El proceso general consiste en procesar los nodos del documento a transformar, según se vaya indicando. El proceso de un nodo consiste en:

1. Localizar una plantilla que admita ese nodo en su patrón de selección. Si hay varias plantillas que cumplen se aplica una regla de prioridad. Si no hay ninguna se aplica una plantilla por defecto.
2. Interpretar el contenido de la plantilla, que habitualmente contendrá instrucciones para procesar ese nodo y sus hijos.

El proceso arranca procesando el nodo raíz del documento.

## 7.2 Generación de la salida

XSLT contiene marcas específicas para generar información por la salida. Además se copian a la salida las marcas que no sean del lenguaje XSLT. Los casos más frecuentes son:

- Literales implícitos: cualquier texto o marca no XSLT
- Literales explícitos: `<xsl:text>...texto...</xsl:text>`
- Valores seleccionados o calculados: `<xsl:value-of select="expresión/xpath">`
- Generar elementos XML: `<xsl:element name="nombre">...</xsl:element>`
- Poner atributos al elemento generado: `<xsl:attribute name="nombre">...valor... </xsl:attribute>`
- Copiar el nodo que se procesa: `<xsl:copy>`
- Copiar nodo completo (subárbol): `<xsl:copy-of select="xpath">`
- Numerar los elementos: `<xsl:number level=... count="xpath" from="xpath" />`

## 7.3 Plantillas

Las plantillas son los componentes fundamentales del código XSLT. El esquema habitual de una plantilla es uno de los siguientes:

```
<xsl:template match="xpath" ...>
  ... acciones ...
</xsl:template>

<xsl:template name="nombre" ...>
  ... acciones ...
</xsl:template>
```

El primer caso corresponde a código con estilo declarativo. La plantilla se ejecuta cuando se procesa un nodo que cumple con el patrón indicado en el atributo match.

El segundo caso corresponde a código con estilo imperativo. La plantilla se ejecuta cuando se la invoca expresamente por su nombre.

El contenido de las plantillas es una combinación de:

- Elementos para generar la salida
- Instrucciones de control de ejecución
- Descripción de parámetros

## 7.4 Control de ejecución

El estilo natural del código XSLT es declarativo. Como ya se ha dicho las plantillas a ejecutar se seleccionan automáticamente en función del nodo a procesar en cada momento. El lenguaje XSLT provee

además mecanismos para programar con estilo imperativo, controlando explícitamente el flujo de control de ejecución. Los dos estilos pueden mezclarse en una misma hoja de transformación. Las instrucciones principales para control de ejecución son:

```
<xsl:apply-templates />

<xsl:apply-templates select="xpath" />

<xsl:call-template name="nombre" />

<xsl:for-each select="xpath">
  <xsl:sort data-type="text|number" select="clave">    [opcional]
  ... acciones ...
</xsl:for-each>

<xsl:if test="expresión">
  ... acciones ...
</xsl:if>

<xsl:choose>
  <xsl:when test="expresión">
    ... acciones ...
  </xsl:when>
  ...
  <xsl:otherwise>    [opcional]
    ... acciones ...
  </xsl:otherwise>
</xsl:choose>
```

La primera forma invoca plantillas con estilo declaractivo para procesar los hijos del nodo de contexto. La segunda forma es análoga, pero procesa los elementos seleccionados explícitamente, en lugar de los hijos.

La instrucción `call-template` invoca una plantilla por su nombre, con el mismo nodo de contexto actual.

La instrucción `for-each` realiza las acciones indicadas sobre la colección de elementos seleccionados. Los elementos se procesan implícitamente en el orden de la selección, o bien en el orden indicado explícitamente por la directiva `sort`.

La instrucción `if` corresponde a una acción condicional. La instrucción `choose` elige una acción entre varias alternativas.

## 7.5 Ejemplo: transformación identidad

Cuando se quiere desarrollar código XSLT para realizar algunos cambios en documentos XML es aconsejable empezar por lo que se denomina *transformación identidad* (en inglés: *identity transform*), que reproduce un documento sin cambiarlo, y luego ir añadiendo plantillas para realizar los cambios particulares que interesen. A continuación se presentan algunas variantes de esa transformación identidad, cada vez más detallada.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<!-- This is the simplest identity function -->
  <xsl:template match="/">
    <xsl:copy-of select="*"/>
```

```
  </xsl:template>
</xsl:stylesheet>
```

Esta primera versión copia el elemento raíz, pero no el prólogo ni el epílogo.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<!-- This is a precise identity function -->
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Esta versión corregida copia todo el documento.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<!-- This is a more detailed identity function;
     It is a good starting point to write XSLT transforms. -->
  <xsl:template match="@*|*|processing-instruction()|comment()">
    <xsl:copy>
      <xsl:apply-templates
        select="*|@*|text()|processing-instruction()|comment()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Esta tercera versión es más detallada, ya que selecciona expresamente y por separado el contenido normal del documento, los comentarios y las instrucciones de procesamiento. Resulta más adecuada como punto de partida, ya que en la mayoría de los casos lo que se quiere procesar es el contenido normal (elementos, atributos y contenido de texto), y no el resto de los nodos.

## 7.6 Variables

Las variables del lenguaje XSLT no son realmente variables, sino valores con nombre. Una vez que se ha asignado un valor a la variable, ya no se puede cambiar. La declaración de una variable se hace de las formas:

```
<xsl:variable name="nombre" select="valor" />

<xsl:variable name="nombre">
  ... acciones ...
</xsl:variable>
```

En el primer caso el valor de la variable es el que se indica con el atributo `select`. En el segundo caso el valor será el generado por las instrucciones contenidas en la declaración de la variable. Para hacer referencia al valor de la variable se usa la notación `$nombre` como parte de una expresión.

Las variables pueden declararse dentro de una plantilla, y sólo existen cuando se ejecuta esa plantilla, o bien a nivel global del código XSLT, y son utilizables en todas las plantillas.

## 7.7 Plantillas con parámetros

La acción de una plantilla puede parametrizarse. Los parámetros de una plantilla se declaran dentro de la plantilla, de manera similar a una variable, con un valor inicial por defecto:

```
<xsl:template ...>
  <xsl:param name="nombre" select="valor-por-defecto" />
  ...
  ... acciones ...
</xsl:template>
```

La diferencia entre una variable y un parámetro es que el valor declarado para el parámetro es sólo un valor por defecto, que no se tiene en cuenta si al parámetro se le da valor desde fuera de la plantilla.

Al invocar una plantilla se pueden dar valores a los parámetros mediante las construcciones:

```
<xsl:apply-templates ...>
  <xsl:with-param name="nombre" select="valor">
  ...
</xsl:apply-templates>

<xsl:call-template ...>
  <xsl:with-param name="nombre" select="valor">
  ...
</xsl:call-template>
```

También se pueden declarar parámetros a nivel global. Los procesadores XSLT deben permitir asignar valor a esos parámetros externamente, al invocar la ejecución del procesador.

### 7.8 Modos

Los modos permiten controlar la asociación de nodos a procesar con plantillas de estilo declarativo. Las declaraciones de plantillas y las órdenes de aplicación pueden incluir un atributo de modo:

```
<xsl:template ... mode="modo">
  ...
</xsl:template>

<xsl:apply-templates ... mode="modo">
```

Cuando se especifica un modo en la instrucción `apply-templates`, sólo se tienen en cuenta las plantillas declaradas con ese mismo modo a la hora de elegir la que hay que aplicar, y viceversa. De esa manera se puede procesar varias veces un mismo nodo, con código de estilo declarativo, produciendo cada vez resultados distintos. Por ejemplo, para generar primero el índice de contenido de un documento, luego el cuerpo del documento, y luego un índice alfabético.

### 7.9 Otros lenguajes de transformación

Hay algún lenguaje especializado de transformación que podría aplicarse a documentos XML, pero la aternativa habitual es usar cualquier lenguaje de programación de uso general que tenga librerías de soporte para el manejo de XML. Estas librerías han de ser, fundamentalmente:

- Manejo de cadenas de texto Unicode.
- Parser XML con interfaz SAX.
- Manejo de documentos XML con interfaz DOM.

Prácticamente todos los lenguajes de programación de alguna importancia tienen este tipo de soporte, bien como librerías nativas o mediante interfaz con librerías C/C++. Entre ellos están C, C++, C#, Java, Ada, Perl, Python, Ruby, AWK, etc.

## 8. Estándar FO (XSL)

Este es realmente el estándar XSL. De hecho no existe un estándar FO, sino que el estándar XSL lo que realmente define son los *Formatting Objects*, y se limita a hacer referencia a XSLT en cuanto a cómo generar FOs a partir de un documento XML.

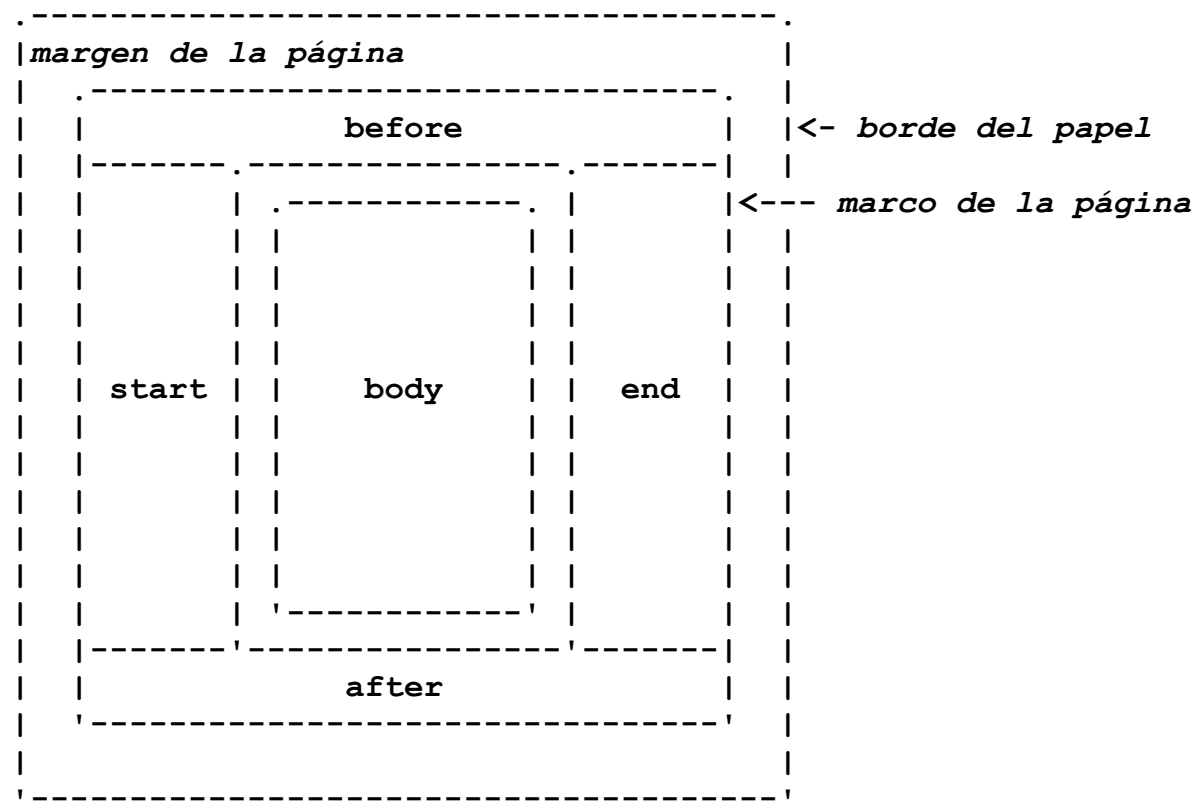
FO es un lenguaje de descripción de páginas, que usa la notación XML. Un documento FO es un documento XML con la siguiente estructura (simplificada):

```
<fo:root ...>
  <fo:layout-master-set>
    ... formatos de página ...
  </fo:layout-master-set>
  <fo:page-sequence ...>
    ... contenido ...
  </fo:page-sequence ...>
  <fo:page-sequence ...>
    ... contenido ...
  </fo:page-sequence ...>
  ...
</fo:root>
```

En el `layout-master-set` se describen los distintos formatos de página que se van a usar (puede ser uno solo). El resto del documento contendrá una o más secuencias de páginas. Cada secuencia de páginas tendrá un contenido que se irá componiendo con el formato de página que corresponda.

## 8.1 Formato de las página

Un formato de página puede ser de página simple o secuencia de páginas. El formato básico es el de página simple, y permite definir hasta cinco regiones diferentes que se irán rellenando con el contenido especificado en las secuencias de páginas. Los contenidos se denominan genéricamente flujos (*flows*), de manera que un formato de página simple puede ir recibiendo contenido de hasta cinco flujos diferentes. La distribución general de la página, mostrando las posibles regiones (o flujos) es:




La zona útil de la página (indicada como marco de la página) se define dejando ciertos márgenes desde el borde del papel. En esa zona útil se distinguen cuatro regiones exteriores: `before`, `after`, `start`, `end`. Estas regiones se definen simplemente por su dimensión (`extent`) desde el marco de la

página. La región central (`body`) se define mediante márgenes desde el marco de la página. Por lo tanto los bordes de la zona central no tienen que coincidir con los de las regiones exteriores. Es posible dejar un espacio en blanco intermedio, o hacerlos coincidir, o incluso solaparlos.

## 8.2 Contenido de las páginas

Una secuencia de páginas contendrá una colección de elementos (texto o imágenes) que irán rellenando las regiones definidas. Hay elementos con nombres similares a los valores de la propiedad `display` en las hojas de estilo CSS, así como otros nuevos. Algunos de esos elementos son:

```
<fo:block ...>
<fo:inline ...>
<fo:list-block ...>
<fo:list-item ...>
<fo:table ...>
<fo:table-row ...>
<fo:table-column ...>
<fo:table-cell ...>
...
```

Estos elementos pueden contener texto u otros elementos de contenido. Además tienen atributos similares a las propiedades de estilo de las hojas CSS. Por ejemplo:

```
border="color, estilo, grosor, ..."
margin="espacio en blanco fuera del borde"
padding="espacio en blanco dentro del borde"
text-indent="sangrado en la primera línea"
text-align="left|right|center|justify|..."
vertical-align="top|middle|bottom|sub|super|..."
font-family="tipo de letra"
font-size="tamaño de letra"
font-weight="intensidad de letra negra"
font-style="itálica, normal, ...."
text-decoration="subrayado, tachado, ..."
color="color del texto"
background-color="color del fondo"
...
```

También hay elementos de contenido especial, tal como imágenes, números de página, etc:

```
<fo:external-graphic src="URI" .../>
<fo:page-number .../>
...
```

El consorcio W3C está realizando un esfuerzo de convergencia entre los atributos de estilo de CSS y XSL-FO, de manera que se puedan usar los mismos motores de composición (*rendering*) para ambos.

## 8.3 Composición de las páginas

Las páginas se van construyendo a base de rellenar las regiones con los elementos de contenido. Para indicar a qué región corresponde cada elemento de contenido, éstos se agrupan en flujos, usando construcciones como las siguientes:

```
<fo:page-sequence ...>
  <fo:static-content flow-name="nombre de la región">
    ... contenido ...
  </fo:static-content>
  ...
```

```
    <fo:flow flow-name="nombre de la región">
      ... contenido ...
    </fo:flow>
    ...
  </fo:page-sequence ...>
```

El contenido estático se repite en cada página. Se usa normalmente en las regiones exteriores de la página, típicamente para la cabecera y el pie de página. El contenido de flujo va rellenado la región hasta agotar el espacio, y entonces se inicia una nueva página. Se usa normamente para la región central (`body`).

## 8.4 Alternativas a XSL-FO

Existen otros lenguajes para descripción de la presentación con estilo, que pueden usarse como alternativa a XSL-FO. La idea es convertir el documento XML a ese otro lenguaje, y luego procesarlo con la herramienta específica correspondiente. La opción más interesante es quizá la basada en TeX/LaTeX, para lo cual hay algunas utilidades ya desarrolladas:

- TeXML es una transcripción a XML del marcado LaTeX. Hay una utilidad XSLT para convertir TeXML -> LaTeX, y así procesarla luego como cualquier otro documento LaTeX.
- ConTeXt es una alternativa a LaTeX que también está soportada por TeXML.
- Consodoc es una utilidad que facilita la repetición de correcciones manuales en la notación TeXML intermedia.

En todo caso parece que la transformación XML -> TeXML requiere un programa XSLT particular para cada lenguaje de marcado XML (igual que como se haría para transformar a XSL-FO). La ventaja de TeXML o ConTeXt es que dicha transformación inicial puede resultar más sencilla que generar LaTeX directamente. Por otra parte los entusiatas de LaTeX dicen que se obtienen documentos impresos con mejor calidad tipográfica que los obtenidos con los procesadores XSL-FO -> PDF.