XML para todos

Ignacio Barrancos Martínez

Tabla de contenidos

Resumen

Copyright (C) 2003, Ignacio Barrancos Martínez `<ibarrancos@exagroup.com [mailto:ibarrancos@exagroup.com]>`. Se garantiza el permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.1 (GNU Free Documentation License, Versión 1.1) o cualquier otra versión posterior publicada por la Free Software Foundation; este documento se presenta sin Secciones Invariables (no Invariant Sections), sin Textos de Tapa (no Front-Cover Texts) y sin Textos de Contratapa (no Back-Cover Texts). GNU Free Documentation License: http://www.gnu.org/copyleft/fdl.html

El presente documento pretende acercar XML al programador, al administrador, al usuario avanzado, al alumno de una Facultad, a un escritor, a un futuro ingeniero, a las amas de casa... en fín... a todos. XML se presentará como una herramienta con la que podrá realizar su trabajo diario de una forma más fácil y cómoda, con la garantía de que siempre podrá reutilizar el esfuerzo inicial en el futuro.

Para lograr esto, se descubrirán los principios teóricos sobre los que se fundamenta la tecnología XML, a fín de situar al lector en el conjunto de términos y herramientas que actualmente se usan en cualquier plataforma informática, ya sea de desarrollo o de consumo final, para que sea él, quien decida cuán útil es la tecnología y cómo puede servirse de ella. Por tanto, un estudio más en profundidad quedará fuera del ámbito de este documento.

Introducción

Como se sabe, *XML* es el acrónimo de *eXtensible Markup Language*, o lo qué es lo mismo: Lenguaje Extensible de Marcado... pero ... ¿qué es un lenguaje de marcado?:

- Por *lenguaje* interpretaremos el conjunto de reglas que definen una sintaxis y una gramática, mientras que,
- *marcado* lo podemos entender como un método para escribir o incrustar metadatos.

Esto introduce otro término nuevo:

- *Metadato* : Es la información que podemos aportar sobre otro conjunto de datos.

Si reunimos todas las piezas, podemos hacernos una idea de lo que es XML: *XML es un conjunto de reglas que describen cómo podemos escribir metainformación en un texto*. Para entenderlo un poco mejor, observaremos el siguiente texto:

```
Juan Antonio
Delos Palotes
Cuatro esquinas
30190
```

El texto anterior es poco explícito para nosotros: 4 líneas que pueden significar cualquier cosa: Un nombre de persona, de empresa, un número de lotería, un fragmento de una conversación, etc... y pueden significar cualquier cosa para nostros, dependiendo de la interpretación que queramos darle. Ahora, en cambio observaremos el siguiente texto.

```
<alumno>
   <nombre>Juan Antonio</nombre>
   <apellidos>Delos Palotes</apellidos>
   <direccion>Cuatro esquinas</direccion>
   <codigopostal>30190</codigopostal>
</alumno>
```

El texto anterior ya empieza a tomar sentido para nostros, y podemos deducir que se trata de los datos personales de un alumno. Si piensa que siempre estuvo claro el significado de las cuatro líneas, observe el siguiente ejemplo:

```
<007_mision>
   <tellamaras>Juan Antonio</tellamaras>
   <contraseña>Delos Palotes</contraseña>
   <recuperar_arma>Cuatro esquinas</recuperar_arma>
   <tiempo_estimado>30190</tiempo_estimado>
</007_mision>
```

El mismo texto incial adquiere de nuevo sentido y no deja pensar en que se trate de los datos personales de un alumno : Ahora el mismo texto son palabras claves de una misión para 007. ¿Qué es lo que ha cambiado para que cambiemos nuestra interpretación? ... a poco que nos fijemos, caeremos en la cuenta que lo que han cambiado son las marcas o tags y como acabamos de decir, las marcas son Metadatos, información que habla sobre el significado de otro conjunto de datos: Al cambiar la metainformación del texto, cambió por completo su interpretación. Si volvemos a leer la definición de XML, la entenderemos un poco mejor.

Una vez que tenemos una ligera idea de lo que significa XML, podemos empezar a descubrir los orígenes de XML, porque esta idea de introducir marcas en un texto nos es nueva, y la encontramos en el lenguaje en el que se construyen las páginas que forman la WWW: HTML. XML es un lenguaje relativamente moderno (Febrero de 1998), y al igual que su homólogo HTML, está basado un lenguaje maduro de marcado llamado SGML: Standard Generalized Markup Languaje.

En 1969, tres señores de IBM Research inventaron el primer lenguaje moderno de marcado: GML (Generalized Markup Languaje) . Este, describía cómo acotar mediante marcas la estructura arbitraria de un conjunto de datos, y llegó a convertirse en un metalenguaje: Un lenguaje que puede ser usado para describir otros lenguajes, sus gramáticas y sus vocabularios. GML se convirtió en SGML, y en 1986 fue adoptado por la ISO como un estándard internacional de almacenamiento e intercambio de datos. Cuando a principios de los 90, se desarrolla HTML este se concibe como aplicación directa de SGML. El impacto que supuso la World Wide Web para el comercio y las comunicaciones , la complejidad de SGML y la carencia de metadatos en el marcado de HTML, desembocan en la necesi-

dad de crear un nuevo lenguaje para la WWW : XML

Objetivos de XML

En 1996, el Consorcio Web (W3C) empieza a trabajar en el diseño de un lenguaje extensible de marcado que combinara la flexibilidad de SGML y la aceptación de HTML. En Febrero de 1998, XML version 1.0 vé la luz. Los objetivos del W3C en el diseño de XML los podemos resumir en :

1. *Extensible* : Dejar abierta la posibilidad de definir nuevos tags en nuestros documentos, al igual que sucede en SGML, y no limitar el conjunto de marcas que podemos usar, como sucede en HTML.

2. *Internacionalización*: El texto de un documento XML debía poder escribirse en diferentes alfabetos y sistemas de escritura (no sólo en lenguajes Europeos) tal y cómo se define en el estándard Unicode 3.0 .

3. *Compatible con SGML*: XML tiene su punto de partida en SGML, y de él nace como un subconjunto simplificado del mismo. Ello nos permite usar las herramientas y trabajos desarrollado para SGML, al tiempo que desarrollar nuevas herramientas para XML, sea mucho más sencillo que hacerlo para SGML.

4. *XML debe poder usarse en Internet* : El hecho de que XML se fundamente en texto ASCII plano, permite usar el hardware de comunicación más simple y rudimentario, como una transmisión serie, y por tanto, podamos usarlo a través de cualquier protocolo moderno como HTTP, HTTPS, FTP, etc ... También puede ser usado como formato de almacenamiento universal e intercambio de información, con o sin Internet, como siempre han sido los ficheros de texto planos.

5. *Separar la Información de su presentación*: Esta meta u objetivo es, a mi juicio, el más difíci de entender y merece la pena detenerse sobre él y verlo por separado en la siguiente sección.

La publicación de la primera versión de XML, que cubría completamente todos los objetivos marcados en su diseño, recibió una gran acogida y aceptación en el seno de la comunidad científica internacional y en la industria informática, y rápidamente se pusieron a trabajar en aplicaciones prácticas basadas en XML . Así, empezaron a surgir ...

Vocabularios XML: Descripciones de datos XML que son usuados como medio de intercambio de información, dentro de un dominio de conceptos específico, como pueden ser la química, las matemáticas, legislación, música, etc...

Actualmente, podríamos enumerar los siguientes vocabularios:

- Entre los *vocabularios científicos*, que por otro lado fueron el primer uso real de XML, encontramos ...

  - *CML (Chemical Markup Languaje [http://www.xml-cml.org])* , lenguaje XML que permite la representación de Moléculas sin pérdida de semántica.

  - *MathML (Mathematical Markup Languaje [http://www.w3.org/Math])* , lenguaje XML que permite la representación de expresiones matemáticas, sin tener que representarlas mediante imágenes o complicadas y horribles aproximaciones ASCII.

  - *IML (Instrument Markup Languaje [http://pioneer.gsfc.nasa.gov/public/iml/])* , lenguaje XML usado por la NASA para el control de instrumentos de laboratorio.

- *Vocabularios Jurídicos*, por llamarlos de alguna forma, que tratan sobre la representación digital de formularios en papel, provenientes de la comunidad médica, jurídica o del comercio, y que encierran ciertas consecuencias legales y jurídicas. Actualmente encontramos *XFDL (eXtensible Forms Description Language [http://www.pureedge.com/resources/xfdl.htm])* el trabajo del W3C sobre XML Canónico y Firma electrónica . [http://www.w3.org/Signature]

- *Vocabularios médicos*, que tratan sobre cómo referenciar materiales y documentos médicos usando el marcado XML, de manera que puedan ser entendidos e interpretados por médicos de todo el mundo y empresas farmaceúticas. Desde 1987 existe una inciativa de estandarización conocida como *HL7(Health Level 7. [http://www.hl7.org]).*

- *Vocabularios de telefonía*. La red conmutada de telefonía ha usado durante años un complejo protocolo propietario llamado SS7 (Signaling System 7). Actualmente empresas como Lucent Technologies y Cisco están trabajando en *CPML(Call Policy Markup Language . [http://www.oasis-open.org/cover/cpml.html]) un estandard abierto que pretende desplazar a SS7.*

Separación: semántica - representación

En HTML se combinan un rudimentario marcado descriptivo (los tags aportan poca meta-información) para describir los datos, con un extenso marcado para indicar la representación y presentación de los mismos. Observemos detenidamente el siguiente fragmento de código HTML:

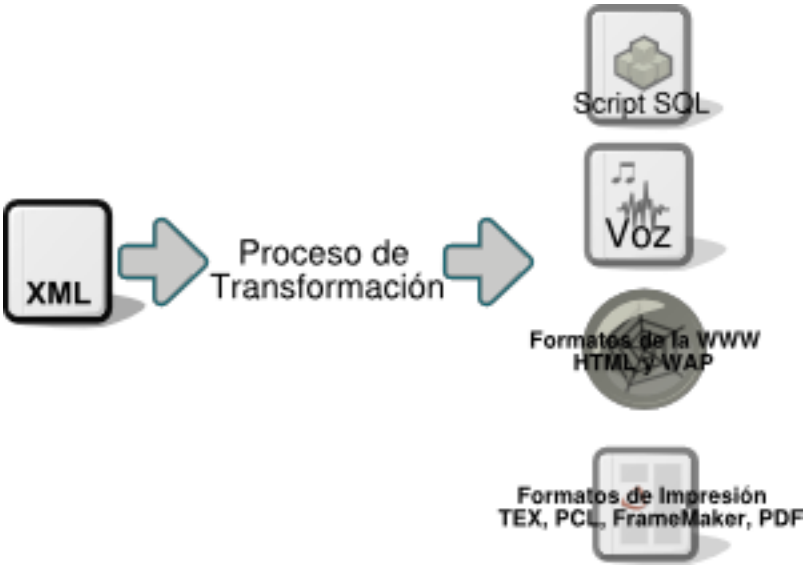
```
<h1 align="right">
   <font family="sans-serif" size="2" color="#F9E">
      <b><i>Separación</i></b>
   </font>
</h1>
```

En vista de este fragmento de código, podemos decir que se trata de un título alineado a la derecha, cuyo tipo de fuente es sans-serif, de tamaño 2 y un color fucsia claro, en negrita e itálica, cuyo contenido es “Separación” ... pero a qué puede referirse esa palabra “Separación” : ¿ a separación de parejas?, ¿a la separación entre dos párrafos?, ¿a la separación mínima entre dos coches en una autovía? . El no encontrar claramente una respuesta es síntoma de haber perdido el contexto semántico del documento, y por contra, hemos ganado muchísimo en cuanto a información de la visualización de ese texto.

Con XML se quería evitar esta pérdida de semántica en los documentos, y ello fué uno de los objetivos principales durante la fase de diseño: En un documento XML , sólo encontraremos la descripción de los datos que forman el documento, y no encontraremos referencias a la representación del mismo.

Esto nos permite independizar el contenido de un documento XML de su representación, mantener separada la información en sí de un documento de su visualización, y tiene como principal ventaja que a partir de un mismo documento XML, podremos obtener diferentes representaciones mediante un proceso intermedio de transformación, tal y como se ilustra en la figura y se puede apreciar en el ejemplo 1.

Figura 1. Representación de ficheros XML



El lenguaje XML 1.0

En la introducción se adelantaron los objetivos perseguidos por los desarrolladores de XML 1.0 en la fase de diseño, y completamente logrados en la especificación final, la cual, no sólo describe el formato de XML-data y su gramática, sino también especifica dos capas cliente en la arquitectura de manipulación de XML-data:

- La primera de estas capas es el *procesador XML* (o *parser XML*). El parser XML debe asegurarse de que el fichero XML cumple completamente con la normativa del lenguaje, proceso que se conoce como *validación*.

  En la introducción, se dijo que uno de los objetivos en el diseño de XML era la Extensebilidad: que cada uno pudiera definir los tags del fichero XML como le viniera en gana, sin limitar para nada el conjunto y la estructura de estos. La consecuencia directa de ello es, que nos podemos definir nuestro propio lenguaje de marcado similar a HTML, pero ... si nos inventamos nuestro propio lenguaje ... ¿cómo podríamos comprobar que un determinado fichero cumple con las reglas que nos acabamos de inventar? ¿debemos usar lex y yacc? ...

  No, es más sencillo que esto: XML contempla este hecho y nos permite especificar la estructura de nuestros lenguajes de marcado en ficheros independientes, llamados DTDs o XML-schemas. De manera opcional, nosotros podemos decirle al parser XML que nos compruebe si un fichero XML, es correcto de acuerdo con las reglas de formación de uno de estos ficheros y él, así lo hará dentro del proceso de validación del XML.

- La segunda de estas capas es el *procesador XSLT*, que a partir de un fichero XML validado es capaz de realizar transformaciones sobre este, de manera obtengamos la presentación del XML-data en el formato que deseemos: A esto se le conoce como *procesado o transformación del XML* .

  El procesado del XML es necesario si queremos separar la información en sí de su presentación, puesto que resultaría muy incómodo leer los documentos en el XML plano, y siempre es preferible leerlos en PDF, HTML, en un móvil WAP, en papel, etc, etc...

  Para realizar estas transformaciones, la especificación propone el uso de hojas de estilo XSLs, que serán para el XML, como CSS para el HTML. En las hojas de estilo se especificarán las transformaciones que deben aplicarse sobre cada tag de nuestro documento XML, para obtener su representación en el formato de salida. Sería algo así como indicarle que el tag ...

  ```
  <nombre>Juan Antonio</nombre>
  ```

  ... deberá transformarse para su representación en HTML en ...

```
<h1 align="right">
    <font color="#A02">
       CONTENIDO_DEL_TAG-XML:<nombre>
    </font>
</h1>
```

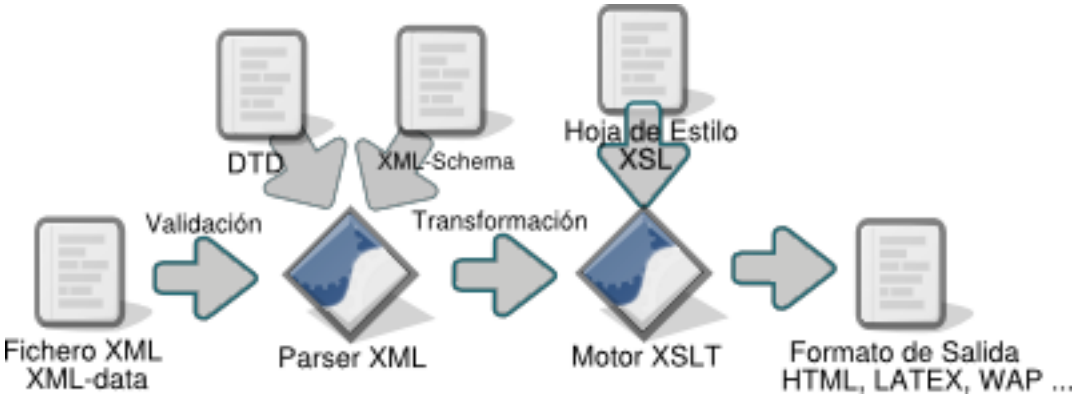
... o en lo siguiente, si quisiéramos que su representación fuera un fichero Latex:

```
\textsf{\textbf{CONTENIDO_DEL_TAG-XML:<nombre>}}
```

Por tanto un fichero XSL contendrá una relación unívoca entre un Tag del XML y un bloque de instrucciones del lenguaje del formato en el que queremos representar el documento. En consecuencia, será necesario tener un fichero XSL por formato de salida.

En la siguiente figura , se resume de manera gráfica esta arquitectura a dos capas para aplicaciones que manipulen XML-data, propuesta en las especificaciones de 1998.

Figura 2. Arquitectura 2 capas para trabajo con XML

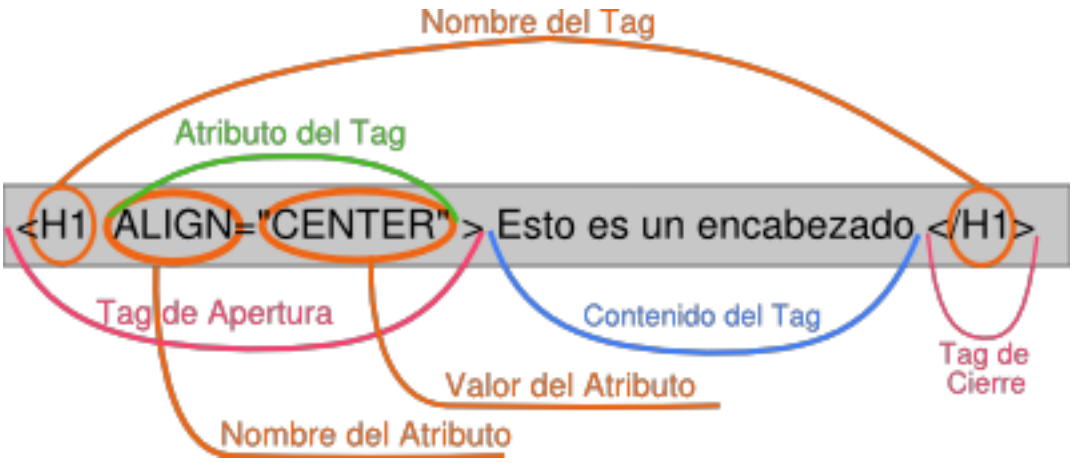


XML-data y sintaxis del marcado

El marcado XML describe la estructura del contenido de un documento XML o paquete de datos. Generalmente se habla de XML-Data para referirse a este contenido. El marcado lo componen los tags que delimitan las diferentes secciones del contenido, las referencias a símbolos especiales y los comentarios que podemos introducir a través de los editores de documentos. Por lo tanto, la sintaxis del marcado nos hablará de cómo deben escribirse los tags, comentarios y símbolos especiales, así cómo de presentar la estructura general de los documentos.

A poco que hayamos visto HTML, estaremos familiarizados con los tags. En un Tag encontramos los elementos que se ilustran en la siguiente figura:

Figura 3. Elementos de un tag



Como se puede intuir las reglas de sintaxis para el marcado de XML, son muy similares a las de HMTL (ambas provienen de SGML): Están basadas en cadenas de símbolos ASCII delimitadas por los símbolos “< ”y “>”. Las reglas de formación de los identificadores de los tags, las podemos resumir en:

1. Los identificadores de etiquetas, son sensibles a mayúsculas y minúsculas, y siempre empezarán a escribirse a continuación el signo “<” , y sólamente podrán formarse mediante :

   - Letras o números

     El símbolo de subrayado

     El símbolo de los puntos “:”, el “-” (menos) y el “.” (punto)

   Esto significa, que ...

   ```
   <DaTos> <dAtos> <DAtoS> <datos> <DATOS> <daTOS>
   ```

   ... son identificadores distintos, y por tanto denotan tags distintos, al contrario que pasaba en HTML.

2. Los Tags deben tener el mismo nombre en la apertura y en el cierre. Por lo que ...

   ```
   <datos> 123456 </Datos>
   ```

   sería incorrecto. Los identificadores son distintos debido a la sensibilidad entre mayúsculas y minúsculas.

3. Los Tags se abren y se cierran en orden (primero se cierra el último que se abrió), a diferencia de los que sucedía en HMTL. Esto significa que ...

   ```
   <FONT> <B> <I> 123456 </FONT> </B> </I>
   ```

   sería incorrecto. Lo correcto sería:

   ```
   <FONT> <B> <I> 123456 </I> </B> </FONT>
   ```

4. Los Tags vacíos o simples (los no duales), se abrirán y cerrarán en el propio Tag. Por tag simple, entenderemos aquellos tags similares a “<IMG>
 <HR>” en el HTML, que no poseían de un tag para la apertura y otro para el cierre, puesto que carecen de contenido (vacíos). Para cerrarlos en el mismo Tag deberemos, añadirles la barra “/” , antes del símbolo “>” :

   ```
   <img src="imagen.gif" alt="logo" />
   <br />
   ```

```
<hr />
```

Esta regla nos permite abreviar los Tags sin contenido de la siguiente manera:

```
<font color="#456" ></font>   ... equivalente a ... <font color="456" />
<alumno></alumno>             ... equivalente a ... <alumno />
```

Elementos y estructura de XML-data

Básicamente *un elemento* de XML-data es un contenedor de contenido, dónde por contenido entendemos una cadena de caracteres, entidades especiales, otro elemento o una sección CDATA. Los elementos siempre estarán delimitados por un Tag de apertura y otro de cierre, que podrá contener cero o más atributos.

En base a esta definición, deberíamos definir:

- *Entidades especiales*: Si todos los tags van delimitados por los símbolos “<” y “>”, ¿cómo podemos poner en el contenido de un Tag, alguno de estos símbolos sin que el parser entienda que estamos abriendo o cerrando un Tag?. Mediante las siguientes entidades especiales:

  - Para el signo “<” se usará: <
  - Para el signo “>” se usará: >
  - Para el signo “&” se usará: &
  - Para el signo “'” se usará: '
  - Para el signo “"” se usará: "

- Una *sección CDATA (<![CDATA[...]]>)* es un bloque de texto que contiene caracteres, que de otra manera, serían interpretados por el parser de XML. Los bloques CDATA nos recuerdan al Tag <PRE> de HTML, o la etiqueta \verbatim de Latex. El contenido de una sección CDATA siempre será ignorado por el parser.

  ¿Cómo podríamos escribir un documento en xML, sobre XML a base de ejemplos, sin que el parser intentara interpretar continuamente todos los ejemplos?. Para esto sirven los bloques CDATA

```
<![CDATA[
Esto es una sección CDATA y me permite escribir cualquier cosa sin que el parser XML lo interpret
& en vez de &amp; < en vez de &lt; etc...
]]>
```

Todo documento XML tendrá la siguiente estructura:

1. El Prólogo contendrá información determinante sobre el resto de datos del documento, y estará compuesto por

   - La *declaración del XML*, donde se indicará obligatoriamente la versión, y con caracter opcional, el conjunto de caracteres en el que está codificado el resto del documento, seguido de cero o más comentarios o instrucciones de procesado.
   - De manera opcional, también podemos encontrar la *declaración del tipo de documento*, que puede ser una referencia a una DTD externa o parte de una DTD, que será usada por el parser XML para comprobar la validez del documento durante el proceso de validación, seguida de cero o más comentarios o instrucciones de procesado.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Esto es un comentario en XML -->
<!DOCTYPE miLenguajeDeMarcado SYSTEM "http://localhost/miLenguaje.dtd" >
```

Por instrucción de procesado, entenderemos los tags de la forma “<?xml .... ?>” . Es habitual encontrar como instrucciones de procesamiento, el estilo que queremos asociar a nuestro XML. Ejemplos de ello serían:

```
<?xml-stylesheet href="ejemplo13.xsl" type="text/xsl" ?>
<?xml-stylesheet href="ejemplo05.css" type="text/css" ?>
```

2. El *cuerpo* del documento, está formado por un único elemento, que recibe el nombre de *Elemento Raíz*.

3. El *epílogo* de un documento XML, está formado por cero o más comentarios o instrucciones de procesado. .

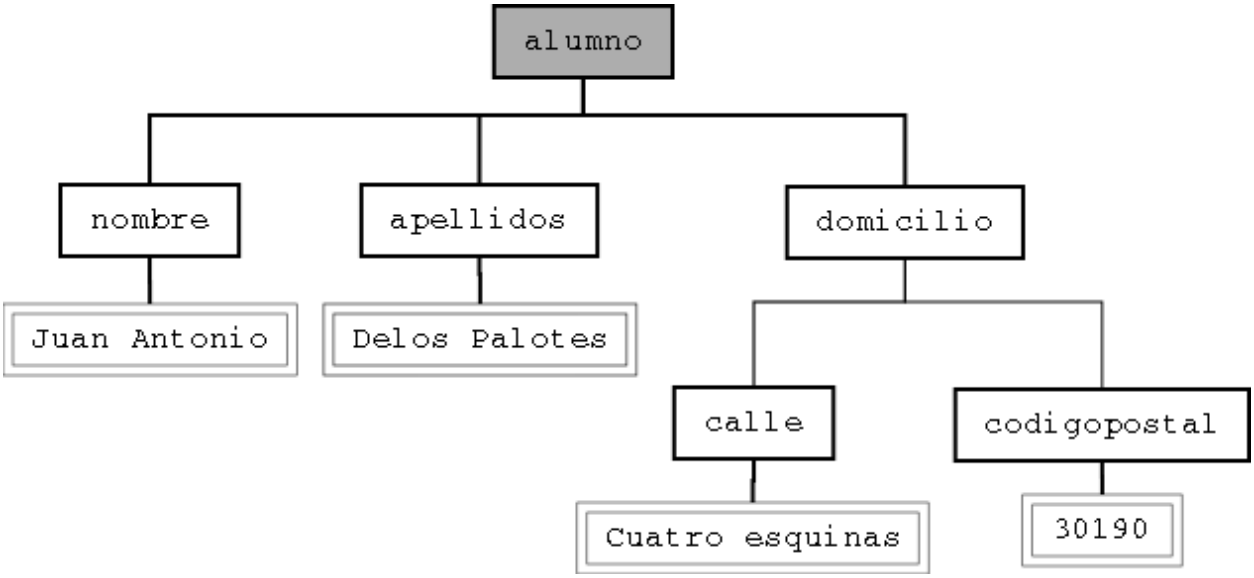
Estructura jerárquica en forma de árbol

Como acabamos de ver en la estructura de un documento XML, cualquier documento deberá contener forzosamente un elemento raíz, y cómo ya se indicó, un elemento a su vez podrá contener uno o más elementos. Esto nos permite poder representar los documentos XML mediante un árbol jerárquico, donde la raíz del mismo será el elemento raíz del cuerpo del documento.

```
<?xml version="1.0" ?>
<alumno>
   <nombre>Juan Antonio</nombre>
   <apellidos>Delos Palotes</apellidos>
   <domicilio>
      <calle>Cuatro esquinas</calle>
      <codigopostal>30190</codigopostal>
   </domicilio>
</alumno>
```

Este documento lo podemos representar en forma de árbol, al igual que haríamos con un árbol genealógico:

Figura 4. Representación jerárquica de un XML



Esta simple representación, nos vá a permitir poder referenciar los elementos de nuestro XML mediante el "argot de familia": Es decir;

- *alumno*: Es el padre de nombre, apellidos y domicilio.
- *calle*: es predecesor de domicilio, y como antecesores tiene a domicilio y a alumno.
- *codigopostal*: Es hijo de domicilio, nieto de alumno y hermano (o hijo del mismo padre) de calle.

  Si observamos bien el XML, también podríamos decir que codigopostal es el siguiente del mismo padre a calle, o en nomenclatura XML:

  ```
  codigopostal= following-sibling(calle)
  calle=preceding-sibling(codigopostal)
  ```

Atributos de elementos

Si entendiéramos una elemento XML, como el nombre de una frase, los atributos serían los adjetivos. Básicamente los atributos proporcionan metainformación sobre un elemento: son un medio para añadir información adicional sobre el contenido de un elemento. Si nos fijamos en este fragmento de fichero XML:

```
<?xml version="1.0" ?>
<alumno>
   <nombre>Juan Antonio</nombre>
   <apellidos>Delos Palotes</apellidos>
</alumno>
```

Podemos deducir que este XML describe los datos de un alumno, gracias a la metainformación que aportan los identificadores de los elementos, pero ...

```
<?xml version="1.0" ?>
<alumno curso="1º" universidad="UM" titulacion="ITI sistemas">
   <nombre>Juan Antonio</nombre>
   <apellidos>Delos Palotes</apellidos>
</alumno>
```

El fragmento anterior, nos permite precisar mucho más, sobre qué tipo de alumno es. Por tanto es justo decir , que un atributo nos aporta metainformación sobre un elemento de nuestro XML.

Las reglas de construcción formación de atributos en XML, son muy escuetas, y las podríamos resumir en:

1. Los identificadores de los atributos se formarán de la misma manera que los identificadores de los elementos.
2. El valor de los atributos, siempre irá entrecomillado: Bien con comillas simples o dobles. Esto significa que lo que haciámos en HTML, ya no podremos hacerlo en XML:

   ```
   <h1 align=right color=#456>Hola </h1>
   ```

3. Nunca se repetirá el nombre de un atributo dentro del mismo elemento, y siempre se especificarán en el tag de apertura. En HTML nosotros podíamos hacer ...

   ```
   <h1 style="margin:0px" color="#456"  style="font-family: sans-serif">Hola </h1>
   ```

   ... que en XML, no nos funcionará.

¿ y qué uso: Atributos o Elementos?

No existe una regla infalible y eficaz que nos ayude a decidir cuando usar atributos y cuándo usar elementos: Como se suele responder en Informática a este tipo de preguntas: “No existe una bala de plata”. Se ha escrito mucho sobre ello y aún hoy se sigue debatiendo en foros de XML. Normalmente antes de tomar una decisión deberemos tener en cuenta que:

- Los atributos se suelen usar para añadir información adicional sobre el contenido real del elemento. Ello les confiere la propiedad de ser opcionales. Sirvan de ejemplo los atributos de estilo que especificamos en HTML:

  ```
  <h1 style="font-weight:bold" align="right">Hola</h1>
  ```

- A los atributos se les puede asignar un valor predeterminado, además de que se puede acotar el conjunto de sus posibles valores.

Teniendo en cuenta estas afirmaciones, la decisión será acertada siempre que solucione nuestro problema, independientemente de cómo lo hayamos hecho.

XML Namespaces

XML Namespace , lo podemos traducir por Espacio de nombres XML . Básicamente, un espacio de nombres XML es un grupo de nombres que comparten un mismo contexto o propósito, agrupados bajo un nombre único. Al igual que sucede en Internet, nosotros podemos tener un conjunto de nombres:

- tecnicos.lanasa.com , madrid.tecnicos.lanasa.com, barcelona.tecnicos.lanasa.com, administracion.lanasa.com

del que diríamos que están dentro del mismo espacio de nombres: lanasa.com

XML namespace, nos proporciona un mecanismo para que podamos indicar elementos de otros lenguajes XML o tipos de documentos, en nuestros propios XMLs. Esto significa que en un documento CML , podremos incrustar una fórmula matemática (MathML) o viceversa, sin esfuerzo ninguno: El parser se encargará de comprobar que todo sea correcto de acuerdo, con las reglas de cada lenguaje.

Podemos usar XML namespaces, añadiendo el atributo xmlns, a cualquier elemento de nuestro archivo, y especificando cómo su valor una URI (Uniform Resource Identifier [http://www.ietf.org/rfc/rfc2396.txt]). Existen dos subconjuntos de URIs:

- URL (Universal Resource Locator [http://www.ietf.org/rfc/rfc1738.txt]), que nos permitirá localizadar de manera concreta un objeto en una red, y
- otra un poco más abstracta, llamada URN (Universal Resource Name [http://www.ietf.org/rfc/rfc2141.txt]), RFC 2141.

Un ejemplo de uso de namespaces en XML, podría ser:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" encoding="iso-8859-1"/>
    <xsl:template match="miDocumento">
        <xsl:apply-templates />
    </xsl:template>
</xsl:stylesheet>
```

Validación de XML

Como se ha presentado al principio de este segundo punto, la Recomendación XML propone una arquitectura a dos capas para las aplicaciones que trabajen con XML. La primera de las capas es el Parser de XML, que como ya se dijo, realiza una validación de XML-data. En este proceso de validación encontramos dos etapas :

XML bien formado

Un fichero XML-data se dice que está bien formado cuando cumple la especificación XML, es decir, cuando ...

- Todos los tags cumplen las cuatro reglas de sintaxis que vimos al principio de la sección 2.1
- El documento tiene un Prólogo, Cuerpo y Epílogo, tal y como vimos en la sección 2.1.1, de manera que podamos representarlo en forma de árbol.
- Todos los atributos de los elementos cumplen las tres reglas que se vieron en la sección 2.1.2.

XML válido

Un documento XML se dice que es válido, si está bien formado, y además verifica la gramática que describe el contenido del documento. Para entenderlo un poco mejor, analicemos la siguiente frase:

```
Desesperadamente el ladra azul perro
```

¿Qué le pasa a esta frase? ... Sencillamente le pasa, que no estamos cumpliendo las reglas gramaticales del Castellano: Se supone, que los artículos deben de ir antes del nombre, luego el nombre y detrás adjetivo. Los adverbios irán detrás del verbo de la oración, y el sujeto irá antes que el predicado:

```
El perro azul ladra desesperadamente
```

Lo mismo sucede con XML: Un XML será válido cuando cumpla con las reglas de la gramática especificada en una DTD o XML-schema, y de realizar esta comprobación se encargará el parser de XML. Cuando nosotros especificamos en el prólogo de un XML un tipo de documento mediante el tag DOCTYPE, le indicaremos al parser dónde debe buscar las reglas para comprobar la validez o no de nuestro documento.

El que un documento sea válido, significará que los elementos respetan las relaciones padre-hijo y que los atributos tienen valores válidos, según lo establecido dentro de una DTD o XML-Schema.

Implementaciones de parsers XML

Las actuales implementaciones de los parsers para XML, se pueden agrupar en dos grandes grupos:

- *Orientados al evento*: Realizan un procesado secuencial del XML. Cada vez que se lee un elemento XML-data, el parser disparará un evento capturable por el programador. En consecuencia, no se mantiene en memoria la estructura jerárquica del fichero que estamos procesando, por lo que tendrán pocos requisitos hardware y el procesado será bastante rápido.

  La mayoría de las implementaciones de estos parsers soportan un API estándard llamada *SAX [http://www.megginson.com/SAX/index.html] (Simple Api for XML)*, de la cual existen dos implementaciones: SAX1 y SAX2 .

- *Basados en Árbol*: Construyen en memoria una representación jerárquica del documento entero, y permiten un acceso aleatorio a los nodos del árbol, lo cual resulta que sean más lentos si se comparan con los orientados al evento,dado que el documento debe estar en memoria antes de que se pueda empezar a procesarlo, a la vez que son imprescindibles para los editores de XML .

  Este tipo de implementaciones soportan un API conforme al *W3C Document Object Model (DOM [http://www.w3.org/DOM/DOMTR/])*, que describe como se pueden manipular los documentos estructurados jerárquicamente.

Y ... ¿ por qué debería usar XML ?

Una vez que conocemos un poco mejor XML, surgen las preguntas:

- ¿ Por qué debería usar documentos XML ?

- ¿ Qué ganan mis aplicaciones si uso XML ?
- Edito todos mis documentos en MS Word (TM) o Latex ... ¿Qué beneficio voy a obtener si cambio a XML?
- ¿ Cuál es el futuro a de XML ?

Aunque enitendo que cada persona deberá decidir en base a sus preferencias, prioridades y necesidades, considero que las siguientes razones tienen el peso suficiente como para obligarnos a una pequeña reflexión:

- *XML es un estándard*. Detrás de XML no hay una empresa que imponga sus intereses económicos o sus estrategias comerciales en el ir y venir de la tecnología . EL W3C lo forman más de 400 organizaciones entre las que figuran Microsoft, Sun, IBM y Oracle. Está llamado al entendimiento y la estandarización, porque lo que es bueno para MicroSoft, a lo mejor no lo es para Sun.
- *XML no es un producto*. En informática estamos acostumbrados a que nos vendan productos que solucionan todos nuestros problemas, y que a la hora de la verdad, solo solucionan parte de nuestros problemas reales, a cambio de un gran esfuerzo económico y de recursos.

  Actualmente, la mayoría de los grandes productos informáticos entienden XML, de alguna manera:

  - .NET, usa XML en la mayoría las capas de su arquitectura. De hecho, ADO.NET construye los recordset en XML.
  - Oracle es capaz de devolver consultas en XML, exportar e importar esquemas relacionales en XML-schema, guardar un repositorio XML de manera natural, así cómo que el propio PL/SQL trabaje con XML.
  - Office XP , (OpenOffice también lo hace en su beta) , exporta e importa XML-data, y se ha anunciado que en la nueva versión que nos viene, el soporte para XML será total.
  - La suite de GNOME 2, trabaja casi de manera íntegra con XML tanto a nivel de configuración como a nivel de formato de ficheros.
  - Tanto Internet Explorer como Mozilla, son capaces de leer, validar y representar XML.

  Podríamos seguir y seguir recapitulando ejemplos de productos que trabajan con XML, pero lo interesante de ellos, es que son las empresas que desarrollan Software las que se encargan de añadir XML a sus productos, nosotros somos meros expectadores. Por tanto, XML no es un producto concreto que me obliga a comprar un software a una empresa concreta para beneficiarme de él.

  Podemos decir de XML, que es una *herramienta que nos permite intercambiar información entre distintas aplicaciones* .
- *XML resulta muy económico*. Tanto los parsers, como las librerías y procesadores de XML, los construyen colectivos de desarrolladores de software libre y propietario, con lo que nosotros sólo tenemos que utilizar, lo que nos supone un considerable ahorro de tiempo y dinero.
- *XML permite la reutilización de contenidos*. ¿Cuántas veces hemos sufrido los caprichos de MicroSoft en su suite Office, por incompatibilidades entre versiones ? ¿ Cuántas hemos tenido que reescribir casi por completo un documento, para poder cambiar el aspecto, tamaño de fuente, colores, etc ? ¿Cuántas veces hemos querido publicar un documento en diferente formato al que lo teníamos escrito? .

  Gracias a que XML desliga el contenido de su presentación, ambas cosas se pueden tratar por separado. Si usted no tiene gusto para la presentación de documentos, puede reutilizar las plantillas escritas por otra persona, para cambiar el formato o la presentación de sus documentos. ¿Cuántas veces no hemos publicado en HTML porque no conocemos el lenguaje, o no tenemos gusto para el diseño?

Latex, es una buena herramienta para la publicación escrita, pero las herramientas de transformación a HTML son poco personalizables, y nuestros documentos no los podemos publicar en la web con el aspecto que queramos. Si usáramos XML para escribir nuestros documentos, con un simple procesado podríamos tener publicaciones en HTML, LATEX, WAP, etc ...

- *XML es el sucesor de HTML en internet*. Los navegadores ya son capaces de renderizar XML, además de que el hecho de que las páginas sean en XML, permitirá a los buscadores realizar búsquedas semánticas con mayor precisión que se realizan actualmente.

Validación de XML

En numerosas circunstancias, el que un documento XML esté bien formado no será suficiente y necesitaremos además que, su estructura jerárquica se ciña a una estructura que nosotros mismos podamos definir, de acuerdo con nuestras necesidades. Cuando un documento bien formado verifique las reglas que nosotros hayamos especificado en otro documento independiente, podremos decir que el documento es válido, con respecto a nuestro tipo de documento, lo que nos permitirá definir nuestro propio lenguaje XML.

Surge la necesidad por tanto, de un mecanismo que nos permita especificar nuestros propios tipos de documentos XML.

DTDs

El primero de los mecanismos que se verán es el uso de DTDs. DTD significa *Document Type Definition [http://www.w3.org/TR/REC-xml]* que, como su nombre indica, permite la defínción de tipos de documentos, mediante un vocabulario SGML .

Las DTDs usan una gramática formal para describir la estructura y la sintaxis de un documento XML, incluyendo los posibles valores que este puede contener. Un documento XML se asociará con su DTD mediante el tag <!DOCTYPE > en el prólogo del mismo. En este tag se podrá especificar una DTD completa (DTD interna), o la URL donde encontrar la DTD (DTD externa) que permite validar el XML.

Un documento XML se puede vincular sólo con una DTD, y esta asociación se puede indicar de dos formas:

- *Usando la sentencia SYSTEM* , lo cual hará que el parser busque la DTD a través de nuestro sistema de archivos o de la red.

```
<?xml version="1.0" ?>
<!DOCTYPE MiDocumento SYSTEM "http://miservidor.mired/mio.dtd" >
<MiDocumento>
  ......

<?xml version="1.0" ?>
<!DOCTYPE MiDocumento SYSTEM "file:///opt/mis-dtds/mio.dtd" >
<MiDocumento>
  ......

<?xml version="1.0" ?>
<!DOCTYPE MiDocumento SYSTEM "mio.dtd" >
<MiDocumento>
  ......
```

- *Usando la sentencia PUBLIC* , seguida de su identificador público. Esto requerirá de un sistema capaz de resolver este identificador, por lo que se suele acompañar de la URL donde encontralo.

```
<?xml version="1.0" ?>
<!DOCTYPE MiDocumento PUBLIC
   "Id_de_Propietario_o_-//Organización_desarrolló_DTD//Qué_es//Idioma"
   "URL_donde_encontrarlo" >
```

```
<MiDocumento>
  ...
```

Ejemplos de identificadores públicos serán:

```
"-//W3C//DTD HTML 4.0//EN"
"-//W3C//DTD HTML//EN/4.1"
```

Se podría decir que una DTD es como una plantilla para un XML, donde especificamos su ubicación a través de tag DOCTYPE.

Contenido de una DTD

Básicamente, el contenido de una DTD estará formado por tags delimitados por los símbolos “<” y “>” de la siguiente manera:

```
<!PALABRA_RESERVADA  parametro1  parametro2 ... parametro_n >
```

...donde, PALABRA_RESERVADA será uno de los siguientes elementos:

- ELEMENT : Declarará el nombre de un tipo de elemento y sus respectivos nodos hijo.
- ATTLIST : Declarará los atributos de un tipo de elemento y sus posibles valores.
- ENTITY : Permite declarar referencias a caracteres especiales, o macros de texto, igual que el #define en C/C++ .
- NOTATION : Permite declarar contenido externo no-XML.

Al igual que hacíamos en XML, podemos introducir comentarios en nuestras DTDs mediante los Tags <!-- y -->.

Espeficación de ELEMENTs

Cada declaración de elementos se identifica por medio de la siguiente sintaxis básica:

```
<!ELEMENT nombre_de_elemento  tipo_de_contenido >
```

...donde *nombre_de_elemento* será el identificador de un tag de nuestro documento, y se nombrará de acuerdo con las reglas de formación de identificadores vistas para XML, y *tipo_de_contenido*, será uno de los siguientes:

- *EMPTY* : que indicará que el elemento no podrá tener contenido.

  ```
  <!ELEMENT  img  EMPTY> <!-- Tag img para el html -->
  <!ELEMENT  hr   EMPTY> <!-- Tag hr (pintar línea horizontal) para el html -->
  ```

- *ANY* : indicará que el elemento podrá contener cualquier contenido. Esto permite definir tags como cajones desastre, tags que pueden contener texto u otros elementos a su vez, o incluso ambas cosas.

  ```
  <!ELEMENT  miTagLoco  ANY>
  ```

- *Colección de elementos*: Describen el conjunto de elementos secundarios (nodos hijo) que puede tener un nodo y el orden en el que deben aparecer estos.

  ```
  <!ELEMENT  alumno ( nombre, apellidos, direccion ) >
  <!-- Esto me obligará a que ....
       <alumno>
          <nombre> ... </nombre>
          <apellidos> ... </apellidos>
  ```

```
        <direccion> ... </direccion>
      </alumno>
    ... y si no, no será válido.
 -->
```

- *Contenido Mixto*:La palabra Mixto, viene porque permite especificar tanto nodos-hijo como texto (#PCDATA) .

```
<!ELEMENT  p  ( #PCDATA | ( b | i ) )* >
<!-- Esto me permite escribir texto en un párrafo en HTML, y en medio insertar tags especiales
     <b> e <i>, para denotar el uso de negrita o cursiva, respectivamente. -->
```

A la hora de especificar el contenido de colecciones , podemos indicar la cardinalidad de las ocurrencias de un elemento o una colección de ellos, mediante :

- A | B : Esto indicará, que o aparece <A> o <B>, pero no ambos.
- A , B : Esto indicarán que aparecen <A> y <B>, y además en ese orden exacto.
- A & B: Aparecen <A> y <B> en cualquier orden.
- A? : Aparece el tag <A> cero o una vez.
- A* : Indicará que aparece el tag <A> cero o más veces.
- A+ : Inidicador de que el tag <A> puede aparecer una o más veces, pero por lo menos una vez.

Espeficación de ATTLISTs

A la hora de definir los atributos que tiene un determinado elemento se usará:

```
<!ATTLIST nombre_de_elemento ( nombre_atributo  tipo_atributo  aparición  valor_predeterminado  )+ >
```

Donde ...

- *nombre_de_elemento* : Es el nombre del elemento XML (el identificador del Tag) sobre el que queremos definir los atributos.
- *nombre_de_atributo* : Es el nombre del atributo del elemento, que estamos definiendo. Se usarán las reglas de construcción de identificadores vistas para XML .
- *tipo_atributo* : Será uno de los siguientes valores:
  - *CDATA* : Indicará que puede contener cualquier caracter alfanumérico, excepto los símbolos especiales (&, <, >, ") que deben aparecer como referencias a entidad: (& < > ")
  - *ID*: Indicará que el valor del atributo será único, y denotará un identificador de elemento.
  - *IDREF*: El valor del atributo será una referencia a un identificador único que deberá existir en nuestro XML.
  - *Enumeraciones*: Estos atributos no están especificados mediante una palabra clave asociada, sino por una lista de valores asociados posibles para el atributo.

    ```
    <!ELEMENT h1 (#PCDATA) >
    <!ATTLIST h1 align ( center | left | right ) >
    ```

- *aparición* : No estamos obligados a especificarlo , pero en caso de hacerlo será uno de los siguientes valores:

  - *#REQUIRED*: Indicará que el atributo es obligatorio y debe tener valor cuando se especifique el elemento en el XML.
  - *#IMPLIED*: Indicará que el atributo es opcional, y no tiene por qué aparecer.
  - *#FIXED*: Cuando se usa este modificador se debe indicar un valor por defecto para el atributo. Cuando debamos editar el XML, no tendremos por qué especificar este atributo, pero en caso de hacerlo, deberá tomar el mismo valor que hayamos especificado como valor por defecto.

    ```
    <!ELEMENT miProyecto (#PCDATA) >
    <!ATTLIST miProyecto tipo CDATA  #FIXED  "pruebas" >
    ```

- *valor_predeterminado* : Sera un texto entrecomillado que se usará como valor por defecto del atributo. Especificarlo no es obligatorio.

  ```
  <!ELEMENT h1 (#PCDATA) >
  <!ATTLIST h1 align ( center | left | right ) "left" >
  ```

Si se quieren indicar más de un atributo para un mismo elemento, se podrán hacer en la misma línea :

```
<!ATTLIST img   src    CDATA                   #REQUIRED
                align  (center | left | right) #IMPLIED >
```

Un ejemplo de DTD: DocBook

DocBook es una DTD desarrollada por O'Reilly y HaL Computer Systems en 1991, y actualmente mantenida por Oasis donada al estandard, que modela documentos técnicos. Por documentos técnicos entenderemos cualquier manual de usuario, memorias de prácticas, libros técnicos, documentación de una API, una seccióndе preguntas frecuentes, etc, etc, etc... En ningún caso modela una carta de amor, un informe de resultados, o un listado.

La idea original de los creadores de DocBook era la de poder generar documentación de Unix de manera que fuera intercambiable, pero actualmente se ha universalizado y su uso es generalizado en cualquier plataforma: Prácticamente todos los editores de XML son capaces de construir documentos en base a esta DTD.

El hecho de generar nuestros documentos usando docbook, nos permitirá benerficarnos de la gran cantidad de programas, y hojas de estilo que existen para este tipo de documentos, con lo que nuestro tiempo se limitará sóla y exclusivamente a escribir, sin preocuparnos de cómo se visualizarán nuestros documentos.

Actualmente se encuentra liberada la versión 4.1.2, disponibleen http://www.oasis-open.org/docbook/sgml/index.shtml [http://www.oasis-open.org/docbook/sgml/index.shtml]

. A poco que busquemos en google docbook2 , encontraremos cantidad de herramientas y scripts que transforman nuestros documentos docbook a páginas man, info, latex, html, etc ...

Este documento que usted leyendo ahora mismo ha sido escrito usando DocBook.

Documentación en La espiral

La espiral [http://www.laespiral.org] es un proyecto abierto que facilita la colaboración de la comunidad hispanohablante dentro del proyecto Debian. Para la entrega de documentación han construido una DTD simplificada y compatible al 100% con DocBook, además de proporcionarnos las herramientas necesarias para transformar documentos escritos a HTML y LATEX, dentro de lo que llaman LE-document, desarrollado Jaime Villate.

Ventajas y desventajas del uso de DTDs

Una vez que estamos introducidos en la construcción de DTDs, podríamos deternos en analizar las ventajas y desventajas que conlleva su uso.

Entre las ventajas podríamos enumerar, que DTD es un estandard robusto y maduro, lo que le aporta mucha experiencia: Ello implica que existen gran cantidad de desarrollos compatibles: programas de edición, parsers, transformadores, ejemplos , etc ... Un ejemplo de ello sería DocBook.

Por contra, podríamos decir que las DTDs son SGML, no XML, por lo que no recoge conceptos recientes como son los NameSpaces. También se le puede reprochar que la definición de tipos es bastante limitada : ¿Cómo indicamos que el valor de un elemento es un float? . Aunque estas desventajas son importantes, no nos impiden que podamos usarlas y aprovecharnos de todo lo que ya hay desarrollado, sin el mínimo temor a que estemos trabajando con algo obsoleto: con XML , todo se puede reciclar.

XML-Schema

A medida que se fué extendiendo el uso de XML, y fueron evolucionando los vocabularios XML, las limitaciones de las DTD se fueron haciendo más presentes, e insalvables. Así en 1999, el w3c empezó a trabajar en un lenguaje de definición de esquemas para XML, e hizo pública la recomendación en Octubre de 200. XML-schema es alternativa a DTD.

Observemos el siguiente código en C:

```
struct Punto {
    int  x;
    int  y;
} pixel = {20,40}
```

Ahora, presentaremos cómo podemos construir esta declaración usando XML-Schema

```
<complexType  name="Punto" >
    <sequence>
       <element  name="x" type="int" />
       <element  name="y" type="int" />
    </sequence>
</complexType>
```

La declaración en el XML, nos quedaría cómo:

```
<pixel type="Punto">
    <x>20</x>
    <y>40</y>
</pixel>
```

Aunque no entraremos en más detalle sobre XML-Schema sí repitiremos que adolece de las desventajas de las DTD y además:

- El contenido de un fichero XML-schema, es XML, y no SGML como el caso de las DTD, además de que permite el uso de NameSpaces.
- Permite la definición de nuevos tipos, que además se pueden extender, lo que nos permitiría heredar. De entrada, ya viene con la definición de todos los tipos usados en los lengujes de alto nivel.
- Permite autodocumentación. Mediante la inclusión de un tag especial, podemos autodocumentar nuestro código de la misma manera que se hace para Java con Javadoc.

Por qué debemos usar DTDs o XML-Schemas

Una vez que estamos introducidos en los mecanismos de validación de documentos XML, nos podríamos preguntar : realmente, ¿es útil el uso de DTDs o XML-Schema?, la respuesta es sí. Veamos por qué:

- *Edición de documentos XML*. Supongamos el caso de que queremos escribir un documento técnico en XML. Podríamos inventarnos una estructura arbitraria y realizarlo sin nigún problema, pero la cantidad de trabajo a realizar no es comparable al uso de una DTD como DocBook. La mayoría de editores que podamos usar, permiten la edición de documentos en base a una DTD que podemos indicarle, con lo que el editor nos irá guiando en todo momento de qué tags podemos insertar y cuales no, a fín de tener un documento válido.

- *Garantiza el intercambio y su representación*. El uso de DTDs permite intercambiar documentos entre usuarios y sistemas: Si cada vez que debo escribir un documento me invento una estructura, apenas podré reutilizar la hojas de estilo que haya definido, ni las herramientas que existen para la importación de XML, dado que no podrán seguir su estructura .

  Oracle, por ejemplo, es capaz de construir un esquema relacional a partir de un XML-Schema. También es capaz de introducir datos en tablas, interpretando un XML válido.

- *Modelado de nuestros tipos de documentos*. En cualquier institución pública o privada, gestionan documentación interna. La mayoría de ellas o son, o estan en proceso de ser ISO-9000, el cual impone que se documenten tanto los procedimientos como los procesos de la misma. Es ideal para este tipo de casos el uso de DTDs y XML, de manera que podemos fijar la estructura de nuestros documentos, y permitir que todo el mundo documente sin el temor de que no respetan las formas, estructuras o normas: Las DTD nos obligarán a ello.

HTML válido = XHTML

A poco que conozcamos HTML, todo lo que llevamos visto nos habrá resultado familiar y ayudado a entender muchos otros conceptos relacionados con HTML. Entonces, nos podríamos preguntar: ¿Html es XML?: La respuesta es no por los motivos siguientes:

- HTML no nos obliga a cerrar los tags, o permite hacerlo de manera desordenada:

  ```
  <p><font color="#4564aa"> Hola esto un <b></i>párrafo</b></i> </p></font>
  ```

- HTML permite que se obvien partes importantes del documento, como por ejemplo <head>

  ```
  <html>
     <body>
        <p>Este párrafo se podría visualizar</p>
     </body>
  </html>
  ```

- HTML no distingue entre mayúsculas y minúsculas en los identificadores de los Tags.

  ```
  <FONT color="#4564aa"> Esto está permitido </font>
  ```

Esto que podríamos llamar vicios, tienen gran parte de la culpa los navegadores, que han interpretado siempre cualquier cosa con tal, de no ceder en la guerra de los navegadores.

Por lo tanto podríamos decir que HTML no es XML, aunque está muy cercano. Si construyéramos HTMLs , atendiendo a las siguientes reglas:

1. Respetar las reglas de construcción de identificadores para los Tags y Atributos de XML.

2. Todos los identificadores de elementos y atributos se escribirán en minúsculas.

3. Los valores de los atributos irán siempre en minúsculas y no se obviarán, como sucede con los select de html:

   ```
   <select name="lista>
      <option name="op1" selected >Esto es ilegal</option>
   </select>
   ```

Los correcto abría sido ...

```
<select name="lista">
   <option name="op1" selected="true">Esto es legal</option>
</select>
```

4. El elemento raíz será <html> y siempre llevará <head> y <body>.

5. Para identificar de manera única los elementos se usará el atributo id, y no el atributo name.

```
<table id="tabla1">
```

6. Los tags <style> y <script> se usaran con secciones CDATA:

```
<script language="JavaScript">
<![CDATA[
  function load( a ) {
     alert ("Esto es un mensaje: "+ a);
  }
]]></script>
```

Respetando estas normas a la hora de construir nuestros HTMLs, en realidad *estaremos escribiendo XHTML*: HTML bien formado y válido. El escribir nuestras páginas web usando XHTML tiene principalmente dos grandes ventajas:

- Se visualizarán correctamente en cualquier navegador, ya sea Linux o Windows.
- Podremos reutilizar los contenidos con una simple transformación. El hecho de que sean XML, nos lo garantiza.

Existen herramientas y webs que nos permiten comprobar todo esto, en concreto: http://www.w3.org/People/Raggett/tidy/ nos permitirá transformar nuestros antiguos HTMLs a XHTML .

Herramientas para trabajar con XML

Una vez que hemos despertado la necesidad de trabajar con XML en el lector, haremos un breve repaso sobre las herramientas que he tenido oportunidad de evaluar. Estas herramientas están orientadas sólamente a la edición de documentos XML, y la mayoría de ellas incluyen su propio parser, para ir comprobando la validez del documento conforme vamos editando.

Las veremos por separado, dependiendo del sistema operativo.

Herramientas para GNU/Linux

getox

GeTox, es una herramienta de edición de XML desarrollada por la empresa francesa IDEALX, y está disponible para gnome con libxml. Sólo permite la edición de documentos y el parseo de los mismos a través de la DTD que se especifique en el tag DOCTYPE del documento, la cual es capaz de recuperarla por http y por el sistema de archivos local.

Lo mejor de GeTox son los frames que nos indican de manera contextual, los elementos que podemos insertar a partir del elemento actual: Tanto hermanos como hijos, y siempre dependiendo del nodo que tengamos seleccionado.

Lo peor de GeTox es, que a pesar de indicarle el charset en el prólogo del XML, sea incapaz de insertar caracteres con tildes, incluso cuando visualiza los que ya hubieran. No permite corrección ortográfica ni diccionarios.

Al parecer IDEALX desapareció como empresa hace un año, y no darán más soporte al programa, que a todos luces, está inacabado aunque es medianamente usable. Es habitual encontrar a alguno de sus desarrolladores (Philippe Bourcier) en el canal xml de irc.gnome.org .

Emacs

Todo un clásico de GNU/Linux. Emacs permite la edición de ficheros XML, con inserción contextual de elementos, dependiendo de la DTD que se le especifique. Para habilitar el soporte de edición XML en emacs, necesitaremos tener instalados:

- *psgml*: Paquete para la edición de ficheros sgml/html en emacs.
- *sp*: Incluye el programa nsgmls que permite examinar la sintaxis de un fichero XML.

Lo malo de emacs ...

- Trabajamos con el fichero en texto plano, de manera que podemos borrar un símbolo < o >, involuntariamente y volvernos locos buscando el error.
- No permite la edición de documentos donde la DTD está accesible mediante http. Probablemente con las últimas versiones de los paquetes lo permita, aunque estos son para la edición SGML, no XML.
- Las teclas y sus combinaciones. Las personas que estamos acostumbradas a trabajar con VI, se nos tuercen los dedos cuando intentamos pasarnos a emacs :-) .

XML mind

XML mind es un programa propietario en Java, cuya distribución estándard es gratuita registrándose on-Line. Se podría decir que es el mejor de los actualmente disponibles, aunque requiere buen hardware debido a la máquina virtual . Viene de casa con DocBook y permite la construcción de XML-Schemas.Básicamente lo tiene todo, aunque se hecha no tener que usar el ratón tan a menudo, y la posibilidad de usar un diccionario en Castellano.

Este documento se ha escrito de manera íntegra usando XML mind. Es el mejor para GNU/Linux.

Herramientas para Microsoft(TM) Windows(TM)

XML Spy

Este programa es de los más famosos y extendidos para edición de XML en Windows, probablemente porque en Internet existe el crack. Aunque no estoy muy familiarizado con el IDE, permite la edición de documentos XML, XML-Schema, DTDs (conversión entre ambos), edición de XSL , y XSL:FO, etc, etc, etc. Es muy completo. Le hecho en falta un vista de edición intermedia, como tiene XMetaL y mayor comodidad en la edición, resulta muy cargante y poco intutitivo.

XMetaL

Este programa de SoftQuad, para mi gusto es el mejor: Permite 4 vistas de nuestros documentos (texto plano, tags, diseño, presentación en el navegador), mediante una interfaz intuitiva y completamente personalizable. Todo se puede personalizar en XMetaL, e incrustar macros en JavaScript y VBScript.

Lo peor de XMetal: Que no tiene diccionario en castellano y que es para Windows, por lo demás es extraordinario.

Presentación de XML

Hasta el momento, tan sólo hemos visto cómo crear y editar nuestros propios tipos de documentos en XML, pero no nos engañemos: El hecho de editar un documento en el formato que sea, la mayoría de las veces es para presentarlo, exponerlo o imprimirlo.

De acuerdo con la arquitectura propuesta por el W3C, la segunda capa se encargaría de transformar nuestro documento XML en un formato presentable como PDF, HTML, WAP, etc. Actualmente existen 2 procedimientos para poder presentar XML: CSS y Transformaciones XSLT. Esto es lo que vamos a ver durante esta sección.

CSS: Cascading Style Sheets

Las hojas de estilo en cascada, son la forma más popular de dar formato a archivos de lenguaje de marcado como HTML. Aprovechando precisamente esta utilidad, podemos usarlas para representar nuestros documentos XML en un navegador web. Esto lo haremos añadiendo una Instrucción de procesado a nuestro XML en el prólogo del mismo:

```
<?xml-stylesheet href="MI_HOJA_DE_ESTILO.CSS" type="text/css" ?>
```

Ello hará que su navegador al visualizar el contenido del XML, acceda a la URL “MI_HOJA_DE_ESTILO.CSS” y la use para renderizar el documento.

Construyendo CSS

Básicamente un fichero CSS, es un fichero de texto plano en el que especificamos reglas CSS con la siguiente sintaxis:

```
selector {
   propiedad_1:valor_1 ;
   ....
   propiedad_n:valor_n;
}
```

Donde ...

- *selector* : es una expresión que identifica a uno o más elementos de nuestro documento XML. Existen cuatro tipos de selectores:

  - *Selector de elementos*: Consiste en poner el identificador de un elemento.

    ```
    alumno { font-size: 12px; }
    ```

  - *Selector de clases*: Consiste en hacer referencia al valor de un atributo para un elemento del XML.

    ```
    alumno [curso="primero"]  { font-color: #4AF; }
    ```

  - *Selector por ID*: Hace referencia al valor del atributo ID, para un elemento del XML, de una manera simplificada:

    ```
    //-Esto es un comentario en CSS
     #figura1 { font-color: #4AF; }
    // equivale a ...
     *[ id="figura1"]  { font-color: #4AF; }
    ```

  - *Selector por contexto*: Hace referencia a un elemento en función de su ubicación dentro del arbol del documento :

    ```
    alumno > nombre { margin: 10px; }
    ```

  Se pueden especificar un mismo bloque de atributos a varios selectores, separándolos por comas.

```
alumno > nombre, apellidos  { border: 3px; }
```

- *Propiedad* : Es una de las propiedades de CSS predefinidas según el estándard CSS2, que se está configurando. Las propiedades las podemos agrupar en varias categorías:

  - *Formatos de FONT*: Nos permite declarar y definir las características del tipo de letra que presentará el elemento en pantalla.
  - *Formatos de FONT*: Nos permite declarar y definir las características del tipo de letra que presentará el elemento en pantalla: Tamaño, familia, espesor, estilo, etc...
  - *Formatos de TEXT*: Nos permite declarar y definir las propiedades del texto que presentamos en pantalla: Espaciado de letras, alineación, identación, etc...
  - *Formatos de COLOR y BACKGROUND*: Estos formatos se utilizan básicamente para definir las características relativas al color y la forma del fondo usado para representas nuestros elementos.
  - *Formatos de BLOQUE*: Cada elemento de nuestro documento XML es un bloque. Mediante esta categoría podemos definir propiedades como: Márgenes, padding, bordes, etc...

- Valor: Será uno de los tipos de valores definidos para esa propiedad concreta.

Se ha adjuntado una guía de referencia rápida para CSS2, de manera que recoja todas las propiedades y sus posibles valores de una manera más detallada.

Desventajas del uso de CSS

Aunque el uso de CSS nos permite dotar a nuestros documentos XML de una “visualización visible”, presenta una serie de inconvenientes que obligaron al w3c a buscar otra alternativa:

- *Un fichero CSS no tiene sintaxis XML*. Si estamos utilizando tecnologías XML carece de sentido que usemos algo no XML. Es el mismo caso que pasaba con DTD y XML-Schema.
- *Es un simple mecanismo de Visualización*. Sólo podemos usarlo en un navegador WEB que soporte CSS2. ¿Qué pasa si queremos imprimirlo, verlo por un móvil o un terminal de Texto?.
- *No existe iteracción con el documento XML*. ¿Y si queremos que los elementos se ordenen? ¿y si queremos que sólo se muestren los documentos que tengan un determinado atributo definido?

  CSS no puede procesar los elementos del documento XML.

XSL Básico

XSL es el acrónimo de eXtensible Stylesheet Language, y lo componen dos estándares abiertos del W3C: XSL:FO (Formatting Objects) y XSLT (XSL Transformations). Mientras el segundo presenta una forma de transformar documentos XML en otros documentos XML o no XML, el primero se compone de un conjunto integral de marcado que funciona de forma muy similar a CSS, y permite entregar documentos XML para su presentación visual y/o impresa.

Los objetivos del W3C cuando diseñó el estandard XSL los podríamos resumir en:

- Lenguaje basado en XML , no como sucedía con CSS.

- Lenguaje Declarativo, claro, legible y extensible, pero que tuviera como mínimo la misma funcionalidad que CSS.
- Debiera poder interactuar con cualquier tipo de documento estructurado, y permitiera la visualización e impresión de documentos XML.

Bajo estas premisas nace el estándard XSL, que nos vá a permitir realizar transformaciones sobre nuestros documentos XML para poder aplicarles una presentación.

XSLT y XPath

Un fichero, donde nosotros especificamos las transformaciones que debemos realizar sobre un documento XML, para obtener otro fichero resultado de aplicarle ciertas transformaciones, normalmente tendrá extensión XSL y lo conoceremos como XSL, hoja de estilo, o plantilla XSL. Este fichero, junto a nuestro documento XML será procesado por un motor XSLT para producir un fichero resultado, en un formato arbitrario, que estará especificado en la propia XSL.

XPath es el lenguaje de expresiones que se usa para referirse a los elementos de un documento XML dentro un fichero XSL: Las expresiones Xpath guardan cierta analogía a la especificación de rutas de directorios en un sistema de archivos.

Un fichero XSL , es a su vez un XML, que empezará de la siguiente forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 PLANTILLAS

</xsl:stylesheet>
```

El elemento raíz será <xsl:stylesheet>, y los hijos de este serán a su vez plantillas, que tendrán el siguiente aspecto...

```
<xsl:template match="ELEMENTO_XML">
  <!--
     Qué debes hacer cuando encuentres el tag <ELEMENTO_XML>
   en el documento XML que estás procesando
  -->
</xsl:template>
```

... donde para cada plantilla mostraremos un bloque de texto (según el formato en el que queramos mostrar nuesto documento de salida), con lo que debe sacar a la salida el motor XSLT, una vez encuentre el Tag <ELEMENTO_XML> en el documento que está procesando.

Nuestro motor XSLT empezará a leer nuestro documento XML desde el nodo raíz, y por cada identificador que elemento que encuentre, intentará encontrar una plantilla en el fichero XSL, que le indique lo que debe mostrar. Si no la encontrara, mostraría el contenido del elemento tal cual.

Hoy día los navegadores web más conocidos traen incorporados su propio motor XSLT, de manera que si nosotros especificamos la XSL que debe usarse para renderizar un documento XML, en el prólogo del mismo, mediante:

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

estos serán capaces de procesar el XML con esta XSL, y mostrar los resultados en el espacio de la página.

XSLT está formado por una gran variedad de instrucciones que nos permiten repetir un bloque de código, ordenar los elementos de un XML, comprobar una cierta condición, bloques switch-case, definir nuestras propias funciones a través de plantillas, etc, etc... Todas ellas las podemos encontrar resumidas en la documentación adjunta en la guía de referencia rápida XSLT y XPATH de Mulberry.

Motores XSLT y Aplicaciones prácticas.

Actualmente existen multitud de motores XSLT, que son capaces de realizar transformaciones sobre nuestros documentos XML para obtener un fichero en otro formato resultante. Destacaría:

- *MSXML* : Motor de Microsoft que usa Internet Explorer. Está catalogado como uno de los más rápidos del mercado. El principal problema que tiene es que es de Microsoft (nunca estará para Linux), no implementa completamente el estandar XSLT hasta la fecha, y no se puede usar desde la línea de consola.
- *SABLOTRON* : Motor Open-Source que viene con la mayoría de las distribuciones de linux. Hasta la versión 0.90 pude comprobar que no implementaba completamente la importación e inclusión de XSL a través de http, además de que no tenía soporte para las instrucciones XSLT de formateo numérico . Con documentos XML de 3MB, y ficheros resultantes de unos 600 folios en latex, era extremadamente lento.
- *XALAN* : Motor Open-Source implementado dentro del proyecto xml.apache.org. Existen dos implementaciones: Una en java y otra en C++. Implementa toda la funcionalidad del estándard XSLT , junto a :
  - *XERCES*: Librería tambien de xml.apache.org que implementa un parser XML: También disponible en Java y C++.
  - *ICU*: Librería open-source de IBM , para el uso de diferentes charsets y formateo de números .

  Existe tanto para Windows como para GNU/Linux y viene a ser el doble de rápido que sablotron, además de no presentar los inconvenientes de este.

Otro motor también muy conocido es Saxon, implementado por Michael Kay, coeditor de XSLT, sólo que no he tenido la oportunidad de probarlo. Según la documentación existente sobre XML, lo sitúan entre los más rápidos.

Aplicaciones prácticas

Con todo lo visto hasta ahora, y con los ejemplos de XSLs, DTDs y XMLs que se adjuntan podremos ir haciéndonos un poco XSLT, que es lo realmente complicado, y buscarle nuestras propias aplicaciones prácticas, aunque yo lo he encontrado las siguientes:

- *Desarrollo Web* : Realmente, gracias XML y XSLT , pude desarrollar la web de Gulmu (grupo de usuarios Linux de Murcia) en unas 16 horas, contando el tiempo que tardé en escribir los contenidos. Lo mejor no es que el desarrollo fuera rápido, sino que permite Separar por completo la presentación visual del portal, de los contenidos, pudiéndolo realizar personas diferentes.

  Esto es importante en un grupo donde los conocimientos sobre informática, HTML o XML pueden ser bastante heterogéneos.
- *Gestión documental* : En instituciones o proyectos, donde se usa una gran cantidad de documentación escrita, con un formato uniforme y bien definido, XML es una gran herramienta. Los documentos pueden ser editados en XML , y luego en base a unas hojas de estilo (XSL), podemos formatearlos dándoles el formato que deseemos tanto para su visualización como para su impresión, siempre de acuerdo con unas normas de estilo definidas dentro de la propia institución.

  Hacerlo con herramientas como MSWord resulta bastante engorroso, sobre todo a la hora de aplicar los formatos.
- *Impresión de Informes (Reports)*: Cualquier aplicación que trabaje con una base de datos, necesita poder imprimir informes. Actualmente, dependiendo del framework de desarrollo de aplicaciones que se use, podrá usar unas herramientas u otras. Implantando una arquitectura de Reports basada en XML podemos obtener principal-

mente dos grandes beneficios:

- Por un lado, se consigue independencia del sistema: Los Reports no será algo que viene con la plataforma y que está limitado por el producto que se usa o la plataforma desde donde se usa.
- Por otro lado, se consigue independencia en cuanto al formato de salida: Podemos imprimir nuestros informes en RTF, XML, PDF, HTML, etc ...