Última revisión del documento:
05/12/1999 16:12:02
AUTOR: Angel Barbero paniagua

TUTORIAL DE

0.-Sobre el tutorial:

Este documento pretende ser una introducción al XML, aunque también entrará en algunos detalles técnicos avanzados. Para entenderlo no hace falta tener conocimiento de ninguna otra tecnología Internet: HTML, SGML, hojas de estilo, DHTML, etc. Pero mi opinión es que para comprender el entorno en el que se coloca XML es bueno haber trabajado con ellas en algún momento, porque sólo entonces se puede obtener una visión completa de las capacidades y potencial de este nuevo lenguaje.

1.-Introducción:

Durante el último año, y sobre todo estos últimos meses se ha estado hablando del Lenguaje Extensible de "Etiquetado", eXtensible Markup Language, o XML, y en todos los foros de Internet que se precien de estar a la última no se puede tratar n ingún tema sin hacer mención a este nuevo estándar. Pero muchas veces se habla de él con poco o nigún conocimiento de causa, y son demasiados los que alaban sus virtudes sin entenderlas ni alcanzarlas por completo.

Antes de nada conviene repasar su historia y precedentes. La versión 1.0 del lenguaje XML es una recomendación del W3C desde Febrero de 1998, pero se ha trabajado en ella desde un par de años antes. Está basado en el anterior estándar SGML (Standard Gene ralized Markup Language, ISO 8879), que data de 1986, pero que empezó a gestarse desde principios de los años 70, y a su vez basado en el GML creado por IBM en 1969. Esto significa que aunque XML pueda parecer moderno, sus conceptos están más que asentado s y aceptados de forma amplia. Está además asociado a la recomendación del W3C DOM (Document Object Model), aprobado también en 1998. Éste no es más que un modelo de objetos (en forma de API) que permite acceder a las diferentes partes que pueden componer un documento XML o HTML.

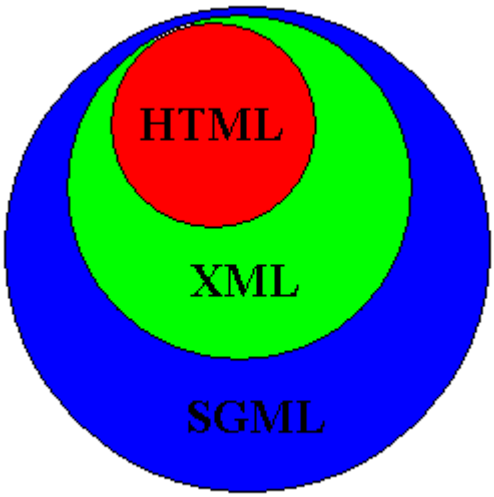
SGML proporciona un modo consistente y preciso de aplicar etiquetas para describir las partes que componen un documento, permitiendo además el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se atribuye a SGML es su e xcesiva dificultad; baste con pensar que la recomendación ocupa unas 400 páginas.

Así que, manteniendo su misma filosofía, de él se derivó XML como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles. Como su padre, y este es un aspecto importante sobre el que se incidirá después, XML es un metalenguaje: es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no

necesariamente sobre su estructura física o presentación, como ocurre en HTML.

Usando SGML, por otro lado, se definió precisamente el HTML, el lenguaje que nos es tan conocido. Entonces, ¿cuál es la diferencia entre ambos?

En una primera aproximación se puede decir que mediante XML también podríamos definir el HTML, con lo que podríamos considerar los siguientes conjuntos:



De hecho, HTML es simplemente un lenguaje, mientras que XML como se ha dicho es un metalenguaje, esto es, un lenguaje para definir lenguajes. Y esa es la diferencia fundamental, de la que derivan todas las demás, que iremos viendo a lo largo del document o.

Desde su creación, XML ha despertado encontradas pasiones, y como para cualquier tema en Internet, hay gente que desde el principio se ha dejado iluminar por sus expectativas, mientras que otras muchas lo han denostado o simplemente ignorado. Debo recono cer que soy de las primeras personas :-), pero espero que esto no afecte al contenido del tutorial, que pretende ser objetivo y sencillo.

Durante el año 1998 XML ha tenido un crecimiento exponencial, y con ello me refiero sobre todo a sus apariciones en los medios de comunicación de todo tipo, menciones en páginas web, soporte software, tutoriales, etc. Para dar una idea, mientras mi fiche ro de bookmarks sobre el tema en septiembre de 1998 tenia unas 5-7 entradas, ahora se me ha hecho imposible el poder controlar todo lo que aparece sobre él, y tengo del orden de 25-30 entradas que debo cambiar y revisar continuamente.

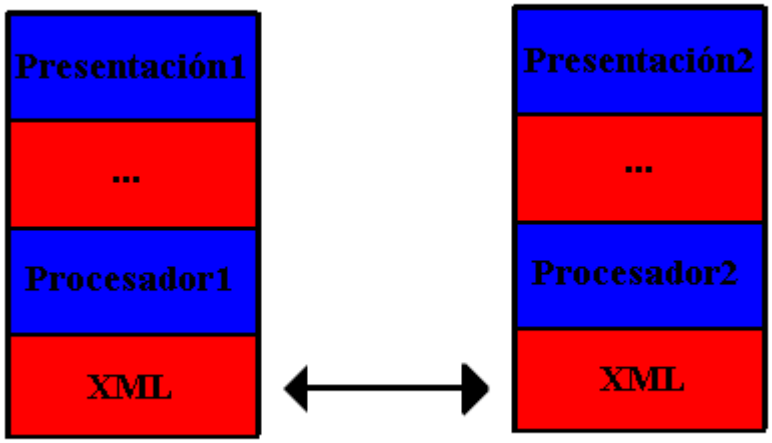
De este modo se ha entrado en 1999 como si se hubiera llegado al final de una pista de despegue; y ahora XML por fin ha despegado. Y esto significa simplemente que desde noviembre del año pasado hasta hoy ha pasado de mera especulación a ser una realidad empresarial palpable y mesurable: los programas que lo soportan han crecido del mismo modo exponencial, y a día de hoy no hay empresa de software que se precie que no anuncie la compatibilidad de sus productos más vendidos con este nuevo estándar: Micros oft (Office 2000), Oracle (Oracle 8i, Web Application Server) o Lotus (Notes) son tres claros ejemplos de ello. Aún más increíble es pensar que hay empresas que se han creado entorno a él, u otras que han movido su actividad hacia su ámbito (de SGML a XML , por ejemplo, como ArborText).

Una pregunta que ha acudido a muchos de nosotros es: ¿será XML el sustituto de HTML, que tan

bien conocemos, dominamos, amamos y odiamos? No. Esa es la respuesta, e intentaré explicar porqué: básicamente XML no ha nacido sólo para su aplicación en Inter net, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, y casi cualqu ier cosa que podamos pensar. Sin ir más lejos, algunos lenguajes de los que hablaremos, definidos en XML, recorren areas como la química y la física, las matemáticas, el dibujo, tratamiento del habla, y otras muchas.

Pues bien, no lo sustituirá, pero, aplicado en Internet, sí va a mejorar algo de lo que HTML empezaba a adolecer desde hace un tiempo: establece un estándar fijo al que atenerse, y separa el contenido de su presentación. Esto significa que a partir de ah ora, o mejor desde el momento que se aplique definitivamente, para ver un documento web no estaremos sujetos a la parte del estándar de hojas de estilo (CSS) que soporte el Navigator de Netscape o el IExplorer de Microsoft, ni al lenguaje de script del se rvidor y al modelo de objetos definido por MS, Nestcape, etc. Además tampoco estaremos atados a la plataforma: podremos ver la misma información desde nuestro PC o desde un Hand-HeldPC, un navegador textual, una lavadora, un microondas o un reloj con acce so a Internet, con presentaciones adecuadas a cada entorno. Ya veremos más adelante el porqué de estas afirmaciones.

Se puede suponer de este modo que XML constituye la capa más baja dentro del nivel de aplicación, sobre el que se puede montar cualquier estructura de tratamiento de documentos, hasta llegar a la presentación. Y así podemos ver la compartición de documen tos entre dos aplicaciones como intercambio de datos a ese nivel:



Pero ahora, y antes de seguir divagando sobre sus posibilidades, expliquemos los conceptos en los que se basa y la teoría de su aplicación, para que podamos escribir documentos XML en poco tiempo.

2.-La teoría:

Lo primero que debemos saber es que hay dos tipos de documentos XML: **válidos y bien formados**. Éste es uno de los aspectos más importantes de este lenguaje, así que hace falta entender bien la diferencia:

- **Bien formados:** son todos los que cumplen las especificaciones del lenguaje respecto a las

reglas sintácticas que después se van a explicar, sin estar sujetos a unos elementos fijados en un DTD (luego veremos lo que es un DTD). De hecho lo s documentos XML deben tener una estructura jerárquica muy estricta, de la que se hablará más tarde, y los documentos bien formados deben cumplirla.
- **Válidos:** Además de estar bien formados, siguen una estructura y una semántica determinada por un DTD: sus elementos y sobre todo la estructura jerárquica que define el DTD, además de los atributos, deben ajustarse a lo que el DTD dicte.

¿Y qué es un DTD? Pues es una definición de los elementos que puede haber en el documento XML, y su relación entre ellos, sus atributos, posibles valores, etc. De hecho DTD está por Document Type Definition, o Definición de Tipo de Documento. Es una especie de definición de la gramática del documento, en definitiva. Más adelante hablaré de cómo se incluye un DTD en un documento XML, pero lo importante ahora es entender que cuando se procesa cualquier infomación formateada mediante XML, lo primero es comprobar si está bien formada, y luego, si incluye o referencia a un DTD, comprobar que sigue sus reglas gramaticales. Hay pues diferencia entre los parsers que procesan documentos XML sin comprobar que siguen las reglas marcadas por un DTD (sólo comprue ban que está bien formado), que se llaman parsers no validadores, y los que sí lo hacen, que son parsers validadores (comprueba que además de bien formado se atiene a su DTD y es válido).

Y ahora, ¿qué pinta tiene un documento XML? Aquí podemos ver uno muy sencillo, que iremos estudiando para ver las características del lenguaje:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ficha>
<nombre>Angel</nombre>
<apellido>Barbero</apellido>
<direccion>c/Ulises, 36</direccion>
</ficha>
```

Lo primero que tenemos que observar es la primera línea. Con ella deben empezar todos los documentos XML, ya que es la que indica que lo que la sigue es XML. Aunque es opcional, es más que recomendable incluirla siempre. Puede tener varios atributos (los campos que van dentro de la declaración), algunos obligatorios y otros no:

- *version*: indica la versión de XML usada en el documento. La actual es la versión 1.0, con lo que no debe haber mucho problema. Es obligatorio ponerlo, a no ser que sea un documento externo a otro que ya lo incluía (ya veremos qué documentos extern os puede haber).
- *encoding*: la forma en que se ha codificado el documento. Se puede poner cualquiera, y depende del parser el entender o no la codificación. Por defecto es UTF-8, aunque podrían ponerse otras, como UTF-16, US-ASCII, ISO-8859-1, etc. No es obligatori o salvo que sea un documento externo a otro principal.
- *standalone*: indica si el documento va acompañado de un DTD ("no"), o no lo necesita ("yes"); en principio no hay porqué ponerlo, porque luego se indica el DTD si se necesita.

En cuanto a la sintáxis del documento, y antes de entrar en el estudio de las etiquetas, hay que resaltar algunos detalles importantes y a los que nos debemos acostumbrar:

- Los documentos XML son sensibles a mayúsculas, esto es, en ellos se diferencia las mayúsculas de las minúsculas. Por ello <FICHA> sería una etiqueta diferente a <ficha>.
- Además todos los espacios y retornos de carro se tienen en cuenta (dentro de las etiquetas, en los elementos).

- Hay algunos caracteres especiales reservados, que forman parte de la sintáxis de XML: <, >, &, " y '. En su lugar cuando queramos representarlos deberemos usar las entidades <, >, &, " y ' respec tivamente. Más adelante hablaré de las entidades y lo que son, pero baste saber ahora que si escribimos cualquiera de las secuencias anteriores equivaldrá a los correspondientes caracteres citados.
- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y la comilla simple (').

Pasando al contenido en sí, vemos etiquetas que nos recuerdan a HTML, y que contienen los datos. Es importante diferenciar entre elementos y etiquetas: los elementos son las entidades en sí, lo que tiene contenido, mientras que las etiquetas sólo descr iben a los elementos. Un documento XML está compuesto por elementos, y en su sintáxis éstos se nombran mediante etiquetas.

Hay dos tipos de elementos: los vacíos y los no vacíos. Hay varias consideraciones importantes a tener en cuenta al respecto:

- Toda etiqueta no vacía debe tener una etiqueta de cerrado: <etiqueta> debe estar seguida de </etiqueta>. Esto se hace para evitar la aberración (en el buen sentido de la palabra) a la que habían llegado todos los navegadores HTML de p ermitir que las etiquetas no se cerraran, lo que deja los elementos sujetos a posibles errores de interpretación.
- Todos los elementos deben estar perfectamente anidados: no es válido poner:

  ```
  <ficha><nombre>Angel</ficha></nombre>
  ```

  , y sí lo es sin embargo:

  ```
  <ficha><nombre>Angel</nombre> </ficha>.
  ```

- Los elementos vacios son aquellos que no tienen contenido dentro del documento. Un ejemplo en HTML son las imágenes. La sintáxis correcta para estos elementos implica que la etiqueta tenga siempre esta forma: <etiqueta/>.

Hasta aquí la sintáxis de XML resumida. Aunque la especificación entera es más prolija en cuanto a detalles sintácticos, codificaciones, etc., creo que no hace falta extenderse mucho en ello, ya que realmente "el movimiento se demuestra andando" ;, y será en la práctica cuando veamos los posibles problemas que se pueden plantear. Ahora quedan por ver otros aspectos, el más prioritario, los DTD.

3.-DTD: Definición de Tipos de Documento

Como antes se comentó, los documentos XML pueden ser válidos o bien formados (o no serlo, claro, pero entonces no serían documentos XML :-). En cuanto a los válidos, ya sabemos que su gramática está definida en los DTD.

Pues bien, los DTD no son más que definiciones de los elementos que puede incluir un documento XML, de la forma en que deben hacerlo (qué elementos van dentro de otros) y los atributos que se les puede dar. Normalmente la gramática de un lenguaje se defi ne mediante notación EBNF; si alguno la conoce, se habrá dado cuenta de que es bastante engorrosa. Pues el DTD hace lo mismo pero de un modo más intuitivo.

Hay varios modos de referenciar un DTD en un documento XML:

- Incluir dentro del documento una referencia al documento DTD en forma de URI

(Universal Resource Identifier, o identificador universal de recursos) y mediante la siguiente sintáxis:

```
        <!DOCTYPE ficha SYSTEM
"http://www.dat.etsit.upm.es/~abarbero/DTD/ficha.dtd">
```

En este caso la palabra SYSTEM indica que el DTD se obtendrá a partir de un elemento externo al documento e indicado por el URI que lo sigue, por supuesto entrecomillado.

- O bien incluir dentro del propio documento el DTD de este modo:

```
<?xml version="1.0"?>
<!DOCTYPE ficha [
<!ELEMENT ficha (nombre+, apellido+, direccion+, foto?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST nombre sexo (masculino|femenino) #IMPLIED>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT direccion (#PCDATA)>
<!ELEMENT foto EMPTY>
]>
<ficha>
<nombre>Angel</nombre>
<apellido>Barbero</apellido>
<direccion>c/Ulises, 36</direccion>
</ficha>
```

La forma de incluir el DTD directamente como en este ejemplo pasa por añadir a la declaración <!DOCTYPE y después del nombre del nombre del tipo de documento, en vez de la URI del DTD, el propio DTD entre los símbolos '[' y ']'. Todo lo que hay entre ellos será considerado parte del DTD.

En cuanto a la definición de los elementos, es bastante intuitiva: después de la cláusula <! ELEMENT se incluye el nombre del elemento (el que luego se indicara en la etiqueta), y después diferentes cosas en función del elemento:

- entre paréntesis, si el elemento es no vacío, se indica el contenido que puede tener el elemento: la lista de elementos hijos o que descienden de él si los tiene, separados por comas; o el tipo de contenido, normalmente #PCDATA, que indica datos de tipo texto, que son los más habituales.
- si es un elemento vacío, se indica con la palabra EMPTY.

Ejemplos de cada caso se pueden ver en el DTD de muestra.

A la hora de indicar los elementos descendientes (los que están entre paréntesis) vemos que van seguidos de unos caracteres especiales: '+', '*', '?' y '|'. Sirven para indicar qué tipo de uso se permite hacer de esos elementos dentro del documento:

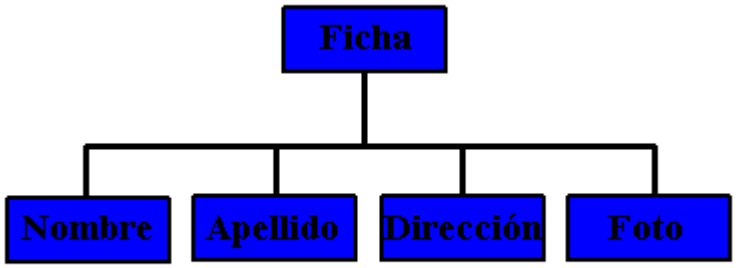
- + : uso obligatorio y múltiple; permite uno o más elementos de ese tipo dentro del elemento padre, pero como mínimo uno.
- * : opcional y múltiple; puede no haber ninguna ocurrencia, una o varias.
- ? : opcional pero singular; puede no haber ninguno o como mucho uno.
- | : equivale a un OR, es decir, da la opción de usar un elemento de entre los que forman la expresión, y solo uno.

De este modo, si por ejemplo encontramos en un DTD la siguiente declaración:

```
<!ELEMENT ficha (nombre+, apellido+, direccion*, foto?, telefono*|fax*)>
```

sabremos del elemento ficha que puede contener los siguientes elementos: un nombre y un apellido como mínimo, pero puede tener más de uno de cada; opcionalmente puede incluirse una o varias direcciones, pero no es obligatorio; opcionalmente también se pu ede incluir una única foto; y por fin, pueden incluirse, aunque no es obligatorio en ninguno de los dos casos, uno o más teléfonos o uno o más números de fax.

Como ya se comentó un documento XML presenta una jerarquía muy determinada, definida en el DTD si es un documento válido, pero siempre inherente al documento en cualquier caso (siempre se puede inferir esa estructura a partir del documento sin necesidad de tener un DTD en el que basarse), con lo que se puede representar como un árbol de elementos. Existe un elemento raíz, que siempre debe ser único (sea nuestro documento válido o sólo bien formado) y que se llamará como el nombre que se ponga en la definición del <!DOCTYPE si está asociado a un DTD o cualquiera que se dese e en caso contrario. Y de él descienden las ramas de sus respectivos elementos descendientes o hijos. De este modo, la representación en forma de árbol de nuestro documento XML de ejemplo sería:



Vemos que es un documento muy sencillo, con una profundidad de 2 niveles nada más: el elemento raíz ficha, y sus hijos nombre, apellido, direccion, foto. Es obvio que cuanta más profundidad, mayor tiempo se tarda en procesar el árbol, pero la dificultad siempre será la misma gracias a que se usan como en todas las estructuras de árbol algoritmos recursivos para tratar los elementos (comentario para aquellos que han programado algo con árboles, aunque habrá un capítulo dedicado a la programación para XML)

El DTD, por ser precisamente la definición de esa jerarquía, describe precisamente la forma de ese árbol. La diferencia (y la clave) está en que el DTD define la forma del árbol de elementos, y un documento XML válido puede basarse en ella para estructur arse, aunque no tienen que tener en él todos los elementos, si el DTD no te obliga a ello. Un documento XML bien formado sólo tendra que tener una estructura jerarquizada, pero sin tener que ajustarse a ningún DTD concreto.

Para la definición de los **atributos**, se usa la declaración <!ATTLIST, seguida de:

- el **nombre de elemento** del que estamos declarando los atributos;
- el **nombre del atributo**;
- los posibles **valores del atributo**, entre paréntesis y separados por el carácter |, que al igual que para los elementos, significa que el atributo puede tener uno y sólo uno de los valores incluidos entre paréntesis. O bien, si no hay valores de finidos, se escribe CDATA para indicar que puede ser cualquier valor (alfanumérico, vamos). También podemos indicar con la declaración ID que el valor alfanumérico que se le de será único en el documento, y se podrá referenciar ese elemento a través de es e atributo y valor;
- de forma opcional y entrecomillado, un valor por defecto del atributo si no se incluye otro

en la declaración;
- por último, si es obligatorio cada vez que se usa el elemento en cuestión declarar este atributo, es necesario declararlo con la clausula #REQUIRED; si no lo es, se debe poner #IMPLIED, o bien #FIXED si el valor de dicho atributo se debe mantener fijo a lo largo de todo el documento para todos los elementos del mismo tipo (notar que no es lo mismo esto a lo que significaba ID).

Es importante destacar un aspecto de cara a la optimización del diseño de nuestros DTDs: muchas veces tendremos que decidir entre especificar atributos de nuestros elementos como elementos descendientes o como atributos en sí mismos. Esto es, podríamo s suponer una pieza de maquinaria con unas características determinadas, y de la que podemos hablar de dos formas diferentes:

```
<pieza>MiPieza
        <color>Rojo</color>
</pieza>
```

o bien considerar la opción:

```
<pieza color="Rojo">Mipieza</pieza>
```

¿Qué diferencia habría entre ambos? Queda a discreción del diseñador el decidir entra ambas, pero hay que tener en cuenta que si se usa la primera forma, el procesador tiene que bajar al siguiente nivel del árbol de elementos para saber el color de *MiPieza*, mientras que en el segundo caso lo puedes obtener referenciando directamente el atributo "color" del elemento en el que estás.

Esto no quiere decir nada de todos modos, porque hay quien prefiere hacerlo del primer modo por estar más acorde con la filosofía de XML de que las etiquetas referencien siempre su contenido, sin necesidad de acudir a los atributos, que era lo que hast a ahora se hacía en HTML.

4.-Las Entidades o *Entities*:

Mediante estos elementos especiales es posible dotar de modularidad a nuestros documentos XML. Se pueden definir, del mismo modo que los propios DTDs de los que ya hemos hablado, dentro del mismo documento XML o en DTDs externos.

Las primeras entidades que hemos conocido son los carácteres especiales &, ", ', < y >, que vimos que debíamos escribir mediante las declaraciones: &amp, ", ', < y >. Es decir, que cuando queramos refe renciar alguna entidad definida dentro de nuestro documento o en otro documento externo, deberemos usar la sintáxis: &*nombre*;.

Existe un conjunto de entidades predefinidas de caracteres ISO, que se pueden ver en una lista resumida publicada en la revista Webmonkey por Jay Greenspan.

Pero por supuesto las entidades no solo sirven para incluir caracteres especiales no ASCII. También las podemos usar para incluir cualquier documento u objeto externo a nuestro propio documento.

Por ejemplo, y como uso más simple, podemos crear en un DTD o en el documento XML una entidad que referencie un nombre largo:

```
<!ENTITY DAT "Delegación de Alumnos de Teleco">
```

Y de este modo, cada vez que queramos que en nuestro documento aparezca el nombre "Delegación de Alumnos de Teleco", bastará con escribir *&DAT;*. Los beneficios son claros, y además es muy sencillo, ¿no?

El texto referenciado mediante la entidad puede ser de cualquier tamaño y contenido. Si queremos incluir una porción de otro documento muy largo que hemos guradado aparte, de tipo XML o cualquier otro, podemos hacerlo por lo tanto de este modo:

```
<!ENTITY midoc SYSTEM "http://www.dat.etsit.upm.es/~abarbero/midoc.xml">
```

Del mismo modo que usábamos con los DTDs externos, indicamos al procesador con SYSTEM que la referencia es externa y que lo que sigue es una URI estándar, y es lo que queda entrecomillado a continuación.

Hay que resaltar que esto se aplica dentro de documentos XML, pero no de DTDs. Para incluir entidades en DTDs debemos usar el carácter %:

```
<!ENTITY % uno "(uno | dos | tres)">
```

Y para expandirlo en un DTD tendremos que escribir:

```
<!ELEMENT numero (%uno;) #IMPLIED>
```

Escapa a mis conocimientos el por qué se ha implementado de forma diferente en los dos casos.

Hasta aquí he resumido los aspectos más importantes de la recomendación de XML y varios temas relacionados con ella. Ahora pasemos a un terreno un poco más abstracto, para conocer otros estándares de facto y tecnologías relacionadas con XML.

5.- Hojas de estilo para XML: XSL

Aunque se ha establecido un modo para que podamos usar hojas de estilo CSS (Cascade Style Sheets) dentro de documentos XML, es lógico pensar que para aprovechar las características del nuevo lenguaje hace falta tener un estándar paralelo y similar asoc iado a él. De este modo en el W3C están trabajando sobre la nueva recomendación XSL: eXtensible Stylesheet Language. Aún no ha llegado a ser recomendación, y de momento es un *public draft*, esto es, un documentosujeto a comentarios y correcciones. < /P>

De todos modos, está bastante avanzado, y de hecho empresas como Microsoft e IBM ya ofrecen soporte a algunas de sus características.

Según el W3C, XSL es "un lenguaje para transformar documentos XML", asi como un vocabulario XML para especificar semántica de formateo de documentos.

En definitiva, además del aspecto que ya incluía CSS referente a la presentación y estilo de los elementos del documento, añade una pequeña sintáxis de lenguaje de script pata poder procesar los documentos XML de forma más cómoda.

Al igual que con HTML hasta ahora, se pueden especificar las hojas de estilo, sean CSS o XSL, dentro del propio documento XML o referenciandolas de forma externa.

6.-EL modelo de objetos de documentos DOM

El modelo de objetos de documentos del W3Consortium, o Document Object Model (DOM) es una representación interna estándar de la estructura de un documento, y proporciona un interface al programador (API) para poder acceder de forma fácil, consistent e y homogénea a sus elementos, atributos y estilo. Es un modelo independiente de la plataforma y del lenguaje de programación.

El W3C establece varios niveles de actuación, coincidiendo con el tiempo en que se presentan como recomendación:

- **Nivel 1:** se refiere a la parte interna, y modelos para HTML y XML. Contiene funcionalidades para la navegación y manipulación de documentos. Tiene 2 partes: el core o parte básica, referida a documentos XML, y la parte HTML, referida precisam ente a los HTML.
- **Nivel 2:** incluye un modelo de objetos e interfaz de acceso a las características de estilo del documento, definiendo funcionalidades para manipular la información sobre el estilo del documento. También incluirá un modelo de eventos para soport ar los XML *namespaces* y consultas enriquecidas.
- Posteriores niveles especificarán interfaces a posibles sistemas de ventanas, manipulación de DTD y modelos de seguridad.

Por el momento ya se ha sacado el 1er nivel como recomendación (Octubre 1998), y el public draft del 2º, es decir, está en etapa de consulta y sujeto a comentarios.

El objetivo es que de una vez por todas cualquier script pueda ejecutarse de forma más o menos homogénea en cualquier navegador que soporte dicho DOM. Siempre por supuesto se podrá elegir el implementar modelos propietarios que es lo que ahora ofrecen Ne tscape y Microsoft, pero tener una plataforma estándar en la que poder crear contenidos sin temor a no estar soportado por alguna marca o versión de navegador, que además sea potente y versátil.

Y por supuesto, como el conjunto de piezas que el W3C está creando para su uso en el intercambio de documentos e información, no estará sujeto al ámbito de los navegadores, sino que su uso será extensible a cualquier tipo de aplicación que acceda a esos documentos.

7.-XML y los navegadores:

Es importante saber qué nos depara el futuro y qué podremos obtener de XML como lenguaje de publicación en Internet. Para ello hace falta conocer de qué forma soportaran los navegadores el nuevo lenguaje. Hay que tener en cuenta que XML es un lenguaje c apaz de describir el contenido de la página, no su aspecto (un metalenguaje, vamos). ¿Y cómo un navegador puede enseñar algo de lo que no sabe su aspecto? Es la pregunta del millón.

La idea es asociar al documento XML una hoja de estilo. Se puede hacer con CSS (Cascading Style Sheets, recomendación del W3Consortium, que va por su versión 2.0), pero se va a aprobar por el W3Consortium una nueva recomendación más completa, la XSL (eXt ensible StyleSheets Language), que permite añadir lógica de procesamiento a la hoja de estilo, en forma de una especie de lenguaje de programación (una especie de lenguaje de script). De este modo, dependiendo de la hoja de estilo que asociemos al documen to o de la lógica que incluyamos, se podrá visualizar en cualquier plataforma: PalmPC, PC, microondas, nevera :-), navegador textual, IE5, Netscape, etc. Y con el aspecto (colores, fuentes, etc) que queramos.

El problema surge del hecho de que el navegador debe incluir de algún modo un parser para XML y un motor que acepte las hojas de estilo (CSS o XSL). Y aqui empiezan a verse diferencias:

- **Microsoft** anuncia que el Explorer 5.0 incorporará internamente todo lo anterior, permitiendo incluir documentos XML dentro de los clásicos HTML (mediante tags <XML> </XML>), o directamente leyendo documentos *.xml; y por otro lado permite el uso con ellas de hojas de estilo CSS y XSL. Y lo peor de todo es que es verdad: lo he probado en una versión Beta y se tragaba XML como nada, y todo quedaba muy bonito. Y ahora en la definitiva lo sigue haciendo, claro está. Ya en el IE4.x ofrecía un parser en forma de control ActiveX, que sin embargo no permitía validar el documento XML; algo que ahora sí hace.
- **Netscape**, sin embargo, parecía que iba a perder el tren del XML (algo bastante grave, según mi opinion), porque su único movimiento en este sentido fue el incluir en el Communicator 5.x soporte para metadatos, pero a través de un estándar sobr e el que también esta trabajando el W3 Consortium, el RDF (Resource Description Framework), que no es sino una implementación de XML (un subconjunto). Sin embargo en marzo por fin ha aparecido en mozilla.org (la organización descendiente de Netscape que g estiona los fuentes liberados de su software) una propuesta para incluir un parser XML para que el navegador soporte de forma nativa tanto RDF como XML. De momento no está implementado, pero desde luego si saben lo que les conviene lo tendrán hecho pronto .

Mi opinión es que parte del futuro del intercambio de datos está en el XML, con lo que el no ofrecer soporte para él es arriesgado. De hecho y para que nos hagamos una idea, algunos productos que darán soporte back-end de XML (lo usarán como formato i ntermedio y de intercambio entre aplicaciones) son el Oracle 8i (la nueva base de datos de Oracle), sus productos orientados al web (Appliaction Server y otros) Windows y Office 2000 y Lotus Notes en un cercano futuro. Y si Microsoft mete dinero ahí, desd e luego es porque algo se huele, ¿no?

8.-Enlaces Relacionados:

- SGML:
  - Extract from the Information Interchange Technology Handbook: What is SGML?: http://www.techapps.co.uk/iihb_sgml.html

- Sitios genéricos:
  - Extensible Markup Language (XML): http://www.w3.org/XML/
  - IBM alphaWorks: http://www.alphaworks.ibm.com/
  - Cafe con Leche XML News, and Resources: http://metalab.unc.edu/xml/
  - SUN: http://java.sun.com/xml/
  - XML.com: http://www.xml.com
  - Webreference:http://www.webreference.com/authoring/languages/xml/
  - DataChannel XML Resources, XML = Revolution: http://www.datachannel.com/xml_resources/
  - POET XML Resource Library: http://www.poet.com/xml/xml_lib.html

- Ejemplos:
  - Examples from XML: Extensible Markup Language: http://metalab.unc.edu/xml/books/xml/examples/
  - Heart of Darkness: http://home.sprynet.com/~dmeggins/texts/darkness/index.html

- Tutoriales:
  - IBM XML Web Site, Education - XML Tutorials for Programmers:

http://www.software.ibm.com/xml/education/tutorial-prog/overview.html
    - DEVELOPERLIFE.COM brought to you by The Bean Factory: http://developerlife.com/xmljavatutorial1/default.htm
    - Introduction to XML (Webmonkey): http://metalab.unc.edu/xml/books/xml/examples/

- Software:
    - IBM: http://www.software.ibm.com/xml/

- Ayudas/Foros:
    - IBM | alphaWorks: http://www.alphaWorks.ibm.com/discussion
    - DejaNews Search Results: http://www.dejanews.com/=netcenter/dnquery.xp?query=jvm&VW=&maxhits=25&format=terse&showsort=score&ST=QS&LNG=ALL

- DOM:
    - XML.COM - DOM: http://www.xml.com/xml/pub/DOM
    - Document Object Model (DOM): http://www.w3.org/DOM/

- XML/EDI:
    - EDItEUR: http://www.editeur.org
    - Guidelines for using XML for Electronic Data Interchange: http://www.geocities.com/WallStreet/Floor/5815/guide.htm

- Varios:
    - Soporte XML en IE5.0: http://www.xml.com/xml/pub/1999/03/ie5/first-x.html
    - XML in Mozilla: http://www.mozilla.org/rdf/doc/xml.html
    - XML: It's Not Your Father's HTML:http://www.webreference.com/authoring/languages/xml/intro/browser.html
    - XML: http://www.harvardcomputing.com/Resources/Reports/XML/xml.html
    - XML.COM - What's the Big Deal With XSL?: http://www.xml.com/xml/pub/1999/04/holman/xsl.html