U.D. 10 El lenguaje XML. DTD.

Índice de contenido

1Introducción.....1
1.1 Las DTD (Document Type Definitions).....3
1.2 Un primer ejemplo: "Hola mundo" en XML.....4
1.3 El concepto de elemento en XML.....5
2 DTD (Document Type Definitions).....8
2.1 Referencia a una DTD en un documento XML.....9
2.2 Declaración de Elemento.....11
2.3 La declaración ATTLIST.....16
2.4 Uso de elementos o de atributos.....21
2.5 Declaración de Entidades.....22
2.6 Declaración de Anotación.....27

1 Introducción

Se define el **estándar XML** como: El formato universal para documentos y datos estructurados en Internet, y podemos explicar las características de su funcionamiento a través de 7 puntos importantes, tal y como la propia W3C recomienda:

1. - *XML es un estándar para escribir datos estructurados en un fichero de texto*. Por datos estructurados entendemos tipos de documentos que van desde las hojas de cálculo, o las libretas de direcciones de Internet, hasta parámetros de configuración, transacciones financieras o dibujos técnicos. Los programas que los generan, utilizan normalmente formatos binarios o de texto. XML es un conjunto de reglas, normas y convenciones para diseñar formatos de texto para tales tipos de datos, de forma que produzca ficheros fáciles de generar y de leer, que carezcan de ambigüedades y que eviten problemas comunes, como la falta de extensibilidad, carencias de soporte debido a características de internacionalización, o problemas asociados a plataformas específicas.

2. - *XML parece HTML pero no lo es*. En efecto, en XML se usan marcas y atributos, pero la diferencia estriba en que, mientras en HTML cada marca y atributo está establecido mediante un significado –incluyendo el aspecto que debe tener al verse en un navegador-, en XML sólo se usan las marcas para delimitar fragmentos de datos, dejando la interpretación de éstos a la aplicación que los lee.

3. - *XML está en formato texto, pero no para ser leído*. Esto le da innumerables ventajas de portabilidad, depuración, independencia de plataforma, e incluso de edición, pero su sintaxis es más estricta que la de HTML: una marca olvidada o un valor de atributo sin comillas convierten el documento en inutilizable. No hay permisividad en la construcción

de documentos, ya que esa es la única forma de protegerse contra problemas más graves.

4. - *XML consta de una familia de tecnologías*. Por supuesto, existe una definición (estándar) de XML 1.0 que viene de Febrero 98, pero su desarrollo se ha ido enriqueciendo paulatinamente a medida que se veían sus posibilidades: de esa forma, contamos con una especificación **Xlink**, que describe un modo estándar de añadir hipervínculos a un documento XML. **XPointer y XFragments** son especificaciones para establecer la forma de vincular partes de un documento XML. Incluso el lenguaje de hojas de estilo (CSS) se puede utilizar con XML al igual que se hace con HMTL. **XSL** es precisamente, una extensión del anterior, en la que se dispone de todo un lenguaje de programación exclusivamente para definir criterios de selección de los datos almacenados en un documento XML, y que funciona conjuntamente con las CSS o con HTML para suministrar al programador y al usuario mecanismos de presentación y selección de información, que no requieran de la intervención constante del servidor. Se basa en un lenguaje anterior para transformación (**XSLT**) que permite modificar atributos y marcas de forma dinámica.

El Modelo de Objetos de Documento (**DOM**) es un conjunto estándar de funciones para manipular documentos XML (y HMTL) mediante un lenguaje de programación. **XML Namespaces**, es una especificación que describe cómo puede asociarse una URL a cada etiqueta de un documento XML, otorgándoles un significado adicional. Y finalmente, **XML-Schemas** es un modo estándar de definir los datos incluidos en un documento de forma más similar a la utilizada por los programadores de bases de datos, mediante los metadatos asociados. Y hay otros en desarrollo, pero todos están basados en el principal: XML.

5. - *XML es prolijo, pero eso no supone un problema*. Los ficheros resultantes, son casi siempre mayores que sus equivalentes binarios. Esto es intencionado, y las ventajas ya las hemos comentado más arriba, mientras que las desventajas, siempre pueden ser soslayadas mediante técnicas de programación. Dado el reducido coste actual del espacio en disco y la existencia gratuita de utilidades de compresión, junto al hecho de que los protocolos de comunicación soportan sistemas rápidos de compresión, este aspecto no debe resultar problemático.

6. - *XML es nuevo, pero no tant*o. El estándar empezó a diseñarse en 1996, y se publicó la recomendación en Febrero/98 (los últimos estándares datan de 2009). Como ya hemos comentado, eso no significa que la tecnología no esté suficientemente madura, ya que el estándar SGML en el que se basa, data de una especificación ISO del año 1986.

7. - *XML no requiere licencias, es independiente de la plataforma, y tiene un amplio soporte*. La selección de XML como soporte de aplicaciones, significa entrar en una

comunidad muy amplia de herramientas y desarrolladores, y en cierto modo, se parece a la elección de SQL respecto a las bases de datos. Todavía hay que utilizar herramientas de desarrollo, pero la tranquilidad del uso del estándar y de su formato, hacen que las ventajas a la larga sean notables.

Tenemos pues, dos partes bien definidas dentro de todo documento XML: la definición de contenidos y los propios contenidos (el DTD y los datos). Cada definición, o DTD constituye de por sí una forma de escribir documentos para Internet.

1.1 Las DTD (Document Type Definitions)

Si queremos tener la seguridad de que un fichero XML pueda ser perfectamente interpretado por cualquier herramienta, podemos (aunque no es obligatorio) incluir una definición de su construcción que preceda a los datos propiamente dichos. Este conjunto de meta-datos recibe el nombre de DTD o Definición de Tipo de Documento. Esta definición sí que debe basarse en una normativa rígida, que es la definida por XML.

Así, si una herramienta es capaz de interpretar XML, eso significa que posee un analizador sintáctico (Parser) que es capaz de contrastar la definición dada por el autor del documento contra la especificada por la normativa, indicando si hay errores, y, de no ser así, presentando el documento de la forma adecuada. Esto es por ejemplo lo que hace Internet Explorer, o Netscape, cuando abren un documento XML.

Una DTD es un documento que define la estructura de un documento XML: los elementos, atributos, entidades, notaciones, etc, que pueden aparecer, el orden y el número de veces que pueden aparecer, cuáles pueden ser hijos de cuáles, etc. El procesador XML utiliza la DTD para verificar si un documento es válido, es decir, si el documento cumple las reglas del DTD.

Recordamos:

Un DOCUMENTO XML BIEN FORMADO es el que sigue las reglas sintácticas. Podemos verificar que nuestro documento está bien formado con un cliente web y con otras muchas aplicaciones.

Un DOCUMENTO XML VÁLIDO es el que es correcto gramaticalmente. Pero la corrección gramatical depende del contenido que queramos expresar y de cómo los datos que manejamos se relacionan entre sí. No es algo objetivo o matemático como la corrección sintáctica. Por eso necesitamos que se valide contra las reglas precisas que estructuran esa información. Tendremos que definir esas reglas (con un DTD o con un Esquema) y después analizar nuestro documento con ellas.

EJEMPLO: biblioteca.xml

Parece evidente que en nuestro ejemplo de este tema sería correcto que el elemento raíz sea biblioteca y que dentro de biblioteca podamos encontrar varios libros. Sin embargo alguien podría construir un documento en el que la raíz sea autor y que dentro haya varias bibliotecas. Sintácticamente puede ser correcto (documento bien formado) pero no tiene sentido (no es un documento válido)

1.2 Un primer ejemplo: "Hola mundo" en XML

Siguiendo la costumbre que inició Charles Petzold en su Programación en Windows comencemos por lo más simple: veamos un primer ejemplo, en el Código Fuente1, del mínimo programa que muestra en pantalla el mensaje "Hola Mundo".

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Contenido [ <!ELEMENT Contenido (#PCDATA)> ]>
<!-- este es un comentario -->
<Contenido>¡Hola, mundo!</Contenido>
```

En el ejemplo ya podemos observar 3 líneas clave:

- La primera, es la definición general. Nos indica que lo que viene a continuación es un documento XML (las de inicio y fin son el carácter obligatorio que delimita esa definición). Además, observamos dos atributos: versión -que se establece a 1.0- que nos indica que el intérprete de XML debe de utilizar las normas establecidas en Febrero/98 y enconding, asignado a “UTF-8”, y que el estándar recomienda incluir siempre, aunque algunos navegadores (como Explorer) no lo exijan de forma explícita.

Téngase en cuenta que XML debe soportar características internacionales, por tanto se dice que, tras su interpretación, todo documento XML devuelve Unicode. El valor por defecto es “UTF-8”.

- La segunda línea es una DTD muy simple. Consta de la declaración de tipo de documento mediante !DOCTYPE seguido del nombre genérico que va a recibir el objeto que se defina a continuación (Contenido), e indica que sólo va a contener un elemento (! ELEMENT) que también se denominará Contenido y que está compuesto de texto (#PCDATA).

- Finalmente, la cuarta línea (la tercera es un simple comentario) contiene la información en sí. Dentro de dos etiquetas de apertura y cierre con el nombre definido en la línea 2, se incluye la información propiamente dicha.

Piense que en ésta salida no estamos indicando ningún modo de presentación. Por tanto el navegador asume que lo que queremos es analizar el documento con el parser y averiguar si existe algún error en él: reconoce el tipo de documento, simplifica el DTD limitándose a mostrar su cabecera, y recorre los datos cambiando el color de las marcas y símbolos para que la interpretación sea más sencilla.

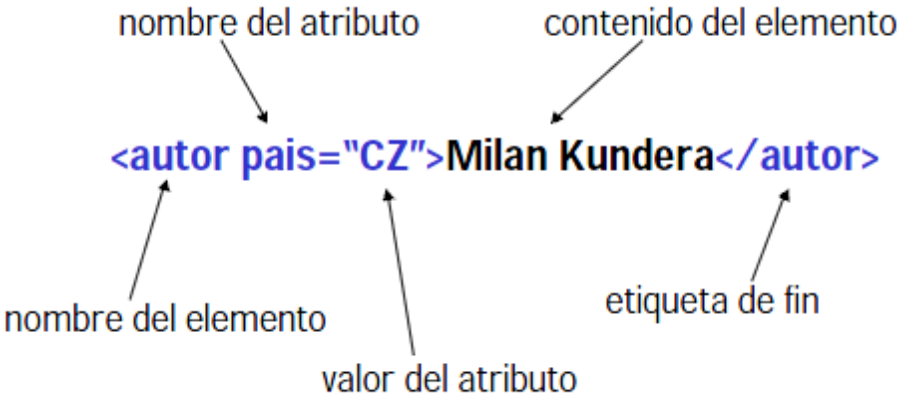
1.3 El concepto de elemento en XML

Los elementos componen la espina dorsal de los documentos XML, ya que crean estructuras que se pueden manipular con programas u hojas de estilos. Los elementos identifican secciones de información con nombres y se crean mediante etiquetas de marcado que identifican el nombre, el inicio y el final del elemento.

Los elementos también pueden incluir valores y nombres de atributos, que proporcionan información adicional sobre el contenido.

Nombres de elementos

Todos los elementos deben tener nombres. Los nombres de los elementos distinguen mayúsculas y minúsculas y deben empezar por una letra o un carácter de subrayado. Un



nombre de elemento puede contener letras, números, guiones, caracteres de subrayado y puntos.

Nota: Los dos puntos se reservan para los espacios de nombres. Si desea obtener más información sobre las letras y los números Unicode aceptados, consulte el Apéndice B de la especificación de XML.

Etiquetas de apertura, etiquetas de cierre y etiquetas vacías

Las etiquetas limitan el contenido, si lo hay, del elemento.

Las etiquetas de apertura indican el comienzo de un elemento y utilizan la siguiente sintaxis general.

```
<elementName att1Name="att1Value" att2Name="att2Value"...>
```

Los atributos son opcionales.

Las etiquetas de cierre indican el final de un elemento y no pueden contener atributos. Las etiquetas de cierre siempre presentan la siguiente forma.

```
</elementName>
```

Un elemento suele incluir las etiquetas de apertura y cierre, y todo el contenido queda delimitado dentro de ellas.

```
<person>
<givenName>Peter</givenName>
<familyName>Kress</familyName>
</person>
```

En este caso, el elemento <person> contiene otros dos elementos, <givenName> y <familyName>, junto con un espacio que los separa. El elemento <givenName> contiene el texto Peter mientras que el elemento <familyName> contiene el texto Kress.

Las etiquetas vacías se utilizan para indicar elementos que no tienen contenido textual, aunque pueden tener atributos. Los elementos HTML img y br son ejemplos de elementos vacíos. Las etiquetas vacías se pueden utilizar como acceso directo cuando no hay contenido entre las etiquetas de apertura y cierre de un documento. Las etiquetas vacías tienen el mismo aspecto que las etiquetas de apertura, pero incluyen una barra diagonal (/) antes del cierre>.

```
<elementName att1Name="att1Value" att2Name="att2Value".../>
```

En XML, puede indicar un elemento vacío con etiquetas de apertura y cierre y sin espacio en blanco o contenido entre ellas; por ejemplo, <giggle></giggle>, o puede utilizar una etiqueta vacía; por ejemplo, <giggle/>. De los dos modos se obtienen resultados idénticos en un analizador XML.

Relaciones de los elementos

Las relaciones entre elementos se describen mediante metáforas relacionadas con familias o árboles. Los documentos XML deben contener un elemento raíz. Aunque puede estar precedido y seguido por otro elemento de marcado, como declaraciones, instrucciones de procesamiento, comentarios y espacios en blanco, la raíz debe incluir todo el contenido que se considera parte del documento. Por ejemplo, el siguiente código puede ser un documento XML con elemento de documento <person> como su elemento raíz.

```
<person>
```

```
<givenName>Stephanie</givenName>
<familyName>Bourne</familyName>
</person>
```

El siguiente fragmento no puede ser un documento XML porque tiene varios elementos raíces.

```
<givenName>Stephanie</givenName>
<familyName>Bourne</familyName>
```

Cuando se utilizan metáforas relacionadas con los árboles, las hojas hacen referencia a los elementos que no contienen ningún otro elemento, como las últimas hojas de una rama. Los elementos hoja suelen ser elementos que incluyen sólo texto o no incluyen nada; los nodos hoja suelen ser elementos vacíos o texto.

La presentación textual de un documento XML, como en todo ML, se puede resumir de la forma : Texto XML = datos + marcado. Esto significa que el texto de un documento XML consta de dos conjuntos : marcado y datos.

- El marcado corresponde a las instrucciones que el analizador XML debe procesar (que se incluyen entre los paréntesis angulares)
- mientras que los datos son el texto entre las marcas o etiquetas delimitada, en inicio y final por paréntesis angulares. El procesador, una vez determinado que todos los caracteres de un documento son aceptables, los diferencia entre texto de marcado y caracteres de datos (CDATA).

2 DTD (Document Type Definitions)

Aunque las DTDs forman parte de la recomendación XML, hay que recalcar que nos encontramos con una aportación procedente de SGML incorporada a XML. Por esa razón situamos su presentación antes del estudio sistemático del XML, ya que es una tecnología previa al XML que era usada como mecanismo, para determinar la validez de un documento (como tendremos ocasión de ver, ya dentro del marco estricto de XML : también se encargan de ello los Esquemas XML).

Una DTD es una colección de declaraciones de elementos (ELEMENT), atributos (ATTLIST), entidades (ENTITY) y notaciones (NOTATION) a partir de las cuales se describe la “validez” de un documento.

Se define siguiendo la sintaxis :

```
<! DOCTYPE nombre [
.............
]>
```

Ejemplo de un fichero XML junto con su DTD:

```
<!DOCTYPE Clientes [
<!ELEMENT Clientes (Cliente+)>
<!ELEMENT Cliente (IdCliente, Empresa, Contacto, Ciudad, Provincia)>
<!ELEMENT IdCliente (#PCDATA)>
<!ELEMENT Empresa (#PCDATA)>
<!ELEMENT Contacto (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
<!ELEMENT Provincia (#PCDATA)>
]>
<Clientes>
<Cliente>
    <IdCliente>1</IdCliente>
    <Empresa>Defensa Personal Orient</Empresa>
    <Contacto>Alfonso Papo</Contacto>
    <Ciudad>Terrassa</Ciudad>
    <Provincia>Barcelona</Provincia>
</Cliente>
<Cliente>
    <IdCliente>2</IdCliente>
    <Empresa>Mística y Salud S.A.</Empresa>
    <Contacto>Ana Coreta</Contacto>
    <Ciudad>Santander</Ciudad>
    <Provincia>Santander</Provincia>
</Cliente>
```

```
<Cliente>
    <IdCliente>3</IdCliente>
    <Empresa>Paños Rias Bajas</Empresa>
    <Contacto>Estela Marinera</Contacto>
    <Ciudad>Vigo</Ciudad>
    <Provincia>Pontevedra</Provincia>
</Cliente>
</Clientes>
```

2.1 Referencia a una DTD en un documento XML

La DTD que debe utilizar el procesador XML para validar el documento XML se indica mediante la etiqueta DOCTYPE. La DTD puede estar incluida en el propio documento, ser un documento externo o combinarse ambas.

- **DTD incrustado**

  Para incrustar un DTD, justo tras la declaración del documento <?xml.....>, escribimos:

  <!DOCTYPE elemento_raiz [

  .....reglas del DTD .....

  ]>

  donde, elemento_raiz será el mismo nombre de la primera etiqueta, elemento raíz del documento.

  atributo standalone="yes" significa que este documento no necesita de otros archivos externos para validarse. Si establecemos standalone="yes" y hubiese alguna referencia externa (a DTDs o entidades), éstas quedaría sin efecto. Las referencias a hojas de estilo no están influidas por este atributo.

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

  Por defecto el valor de standalone es "no"

- **DTD externo**

  Para llamar a un DTD externo, justo después de la declaración del documento <? xml.....>, escribimos:

  <!DOCTYPE elemento_raiz PUBLIC "FPI" SYSTEM "URI">

  donde:

  ➢ elemento_raiz cumple la misma función que en el DTD incrustado, será la primera etiqueta de nuestro documento.

  ➢ PUBLIC indica que el siguiente parámetro contiene la definición pública del

  documento en forma de FPI.

  ➢ "FPI" (Formal Public Identifier, Identificador público formal) es un nombre que identifica DTDs públicos y conocidos. Si la aplicación reconoce el FPI (este es el camino más rápido) ya sabe contra que DTD debe validar. Si la aplicación no reconoce el FPI buscará el DTD en el URI señalado.

Un FPI tiene la siguiente estructura:
-//propietario//palabra_clave descripción //idioma
(se trata del idioma del DTD, no del idioma del documento)
EJEMPLO:
-//W3C//DTD HTML 4.01 //EN

➢ SYSTEM indica que el siguiente parámetro contiene la URI que señala al fichero (de nuestro sistema o de la web) que contiene el DTD.

➢ "URI" será siempre un fichero.dtd (camino absoluto o relativo hasta la DTD.)

En la declaración del tipo de documento no todos los elementos son obligatorios:

➢ Podemos omitir PUBLIC "FPI", en ese caso validará contra el DTD indicado en la URI. Este será el caso cuando no se trate de un DTD público, sino uno que creamos nosotros.

➢ Podemos omitir SYSTEM "URI", en ese caso estamos a expensas de que la aplicación reconozca el FPI. Si se trata de DTDs para (X)HTML y de un cliente web no habrá problemas. Pero puede haberlos si se trata de otra aplicación.

➢ Podemos omitir la palabra SYSTEM cuando se ponen ambos (FPI y URI) porque el orden siempre será primero el FPI y segundo el URI.

- **DTD externo e incrustado**

También es posible combinar ambos métodos de modo que tendremos un DTD externo y al mismo tiempo reglas DTD incrustadas en la declaración DOCTYPE. La sintaxis que permite esta mezcla es la del siguiente ejemplo:

```
<!DOCTYPE biblioteca SYSTEM "biblioteca3.dtd" [

        <!ENTITY jd "Juan Diego Gutiérrez Gallardo">

        <!ENTITY js "José Saramago">

]>
```

La parte de la DTD incluida entre corchetes se denomina SUBCONJUNTO INTERNO DE DTD y todas las partes que se encuentran fuera de este documento se denominan SUBCONJUNTO EXTERNO DE DTD.

Ambos subconjuntos forman la DTD completa. Como regla general, los dos subconjuntos deben ser compatibles. Ninguno debe sobrescribir las declaraciones de elemento del otro. Sin embargo, las declaraciones de entidad sí se pueden sobrescribir como se muestra más adelante.

NOTA: Véase el ejemplo xml_ejemploDTD.zip

Hemos visto que un DTD contiene la información sobre la semántica de un documento.

Básicamente tiene que describir los elementos que pueden formar parte del documento y los atributos y contenidos de cada elemento.

Hay tres reglas DTD:

- las reglas ELEMENT para definir los elementos de nuestro documento,
- las reglas ATTLIST para definir los atributos de un elemento,
- y las reglas ENTITY para definir entidades de caracteres.

2.2 Declaración de Elemento

Los elementos de una DTD son los bloques primarios de todo documento y se declaran de la forma:

```
<!ELEMENT nombre_elemento  modelo de contenido  >
```

Se empieza con "<!ELEMENT", seguido de un nombre_elemento identificador válido XML, y los paréntesis especifican el contenido que está permitido, por lo que se conocen como especificación o modelo de contenido, que a su vez puede contener más elementos, creándose así una posible cadena de subelementos anidados entre sí.

La virtualidad de esta declaración reside en la idea de definir las relaciones existentes entre los distintos elementos de un documento. Dado el siguiente ejemplo:

```
<!DOCTYPE receta [
<!ELEMENT receta (plato, ingredientes, preparación)>
<!ELEMENT plato (#PCDATA)>
<!ELEMENT ingredientes (#PCDATA)>
<!ELEMENT preparación (#PCDATA)>
]>
```

El elemento receta contiene a su vez los elementos plato, ingredientes y preparación, que a su vez, están definidos también en la DTD.

Los **modelos de contenido** que puede tener un elemento son :

- (#PCDATA) : conocido como “Parser Character Data”, cualquier combinación de caracteres. Es decir, cualquier letra , dígito menos <, > o & . Estas cadenas son pasadas por el parser (analizador) que deja las secuencias de espacios, tabuladores o saltos de línea en un solo espacio. Puede contener referencias a entidades(< > ó &)Las comillas simples y dobles pueden sustituirse por las entidades " y ' , secciones CDATA y comentarios, pero no puede contener etiquetas ni hijos.

```
<!ELEMENT miElemento (#PCDATA)>
```

- ANY se admite cualquier contenido. Es útil mientras estamos construyendo el DTD pero no debería aparecer en un DTD terminado.

```
<!ELEMENT batiburrillo ANY>
```

- EMPTY el elemento será vacío, sin contenido, aunque podrá tener atributos.

```
<!ELEMENT saltodepagina EMPTY>
```

- (elemento): el contenido de un elemento puede ser otro elemento.

  <!ELEMENT clase (profesor)>

  indica que clase sólo puede contener un solo elemento profesor.

- grupo: (cont1, cont2, ..., contN) el contenido puede ser múltiple, entre paréntesis y separados por comas. Tendrá que respetarse el orden.

  <!ELEMENT clase (profesor,aula)>

  especifica que el elemento clase, debe contener un elemento profesor seguido de un elemento aula.

- alternativas: (cont1 | cont2 | ... | contN) el contenido será uno de entre los incluidos (sólo uno).

```
<!ELEMENT enfasis (#PCDATA)>
<!ELEMENT parrafo (#PCDATA | enfasis)
```

  el primer elemento enfasis puede contener datos de carácter (#PCDATA) y el segundo parrafo puede contener datos de carácter o un elemento de tipo énfasis.

```
<!ELEMENT postre (helado | pastel)>
```

indica que postre puede contener bien un elemento helado bien un elemento pastel. El numero de opciones no está limitado a dos, y se pueden agrupar usando paréntesis, de la forma :

```
<!ELEMENT postre ( sorbete, (helado | pastel))>
```

- Al final de cada contenido podemos poner un modificador de entre los siguientes:

Símbolo	Significado
+	el elemento puede aparecer una o más veces
*	el elemento puede aparecer cero o más veces
?	el elemento puede aparecer cero o una vez
|	el elemento puede aparecer una vez a escoger de una lista enumerada

Ejemplo 1:

```
<!ELEMENT aviso ( titulo?, (parrafo+, grafico)*)>
```

se especifica que aviso puede tener titulo o no (pero sólo uno), y tener cero o más conjuntos de la forma : (parrafo, grafico),(parrafo,parrafo,grafico),etc.

Ejemplo 2:

```
<!DOCTYPE formato [
<!ELEMENT formato (#PCDATA | negrita | italica)*>
<!ELEMENT negrita (#PCDATA)>
<!ELEMENT italica (#PCDATA)>
]>
```

la línea <!ELEMENT formato (#PCDATA | negrita | italica)*> declara el elemento formato de contenido mixto que puede contener bien caracteres de datos (PCDATA), o el elemento negrilla, o el elemento itálica y el contenido puede darse ninguna o varias veces; mientras las dos líneas que siguen especifican que negrita e italica solamente tienen PCDATA como modelo de contenido.

Ejemplo 3:

<!ELEMENT titulo (#PCDATA)>

Define un elemento con nombre titulo, cuyo contenido puede ser cualquier combinación de caracteres. Esta regla permite que nuestro documento XML incluya un elemento como este:

<titulo>La lluvia en Sevilla es pura maravilla</titulo>

Ejemplo 4:

<!ELEMENT biblioteca (libro)*>

Define un elemento con nombre biblioteca, cuyo contenido puede ser 0 ó más elementos libro. Esta regla permite que nuestro documento XML incluya elementos como este:

<biblioteca>

<libro> ... contenido de libro ... </libro>

<libro> ... contenido de libro ... </libro>

</biblioteca>

o como este:

<biblioteca />

Ejemplo 5:

<!ELEMENT libro (titulo, autor+, genero?, editorial)>

Define un elemento con nombre libro, cuyo contenido serán los elementos titulo, autor, genero y editorial. El símbolo (+) indica que el autor puede aparecer 1 ó más veces (o sea, es obligatorio que haya algún autor y es posible que haya más de un autor). El símbolo (?) indica que el genero puede aparecer 0 ó 1 veces (es decir, que incluir el genero de un libro es opcional).

Ejemplo 6:

<!ELEMENT vehiculo (mercancia | pasajeros)>

Define un elemento con nombre vehiculo, cuyo contenido será un elemento mercancia o un elemento pasajeros, pero no ambos. Admite elementos como estos:

```
<vehiculo>
        <mercancia>
                <kilos>2000</kilos>
        </mercancia>
</vehiculo>
 <vehiculo>
        <pasajeros>
                 <capacidad>60</capacidad>
          </pasajeros>
 </vehiculo>
```

Ejemplo 7:

<!ELEMENT libro (titulo, (autor | colaborador | comentarista)*, precio*)>

Cada libro tendrá un título seguido de cero o más autores, colaboradores y/o comentaristas mezclados en cualquier orden y seguidos de 0 o más precios (por ejemplo en distintas monedas)

Ejemplo 8:

Un elemento con contenido mixto se declara así:

<!ELEMENT definition (#PCDATA | term)* >

La declaración anterior permite que definition tenga cero, uno o muchos hijos term. Las declaraciones de contenido mixto no especifican el orden en que aparecen los elementos hijos ni cuántas instancias habrá de cada uno.

El formato de la declaración de contenido mixto es muy rígido:

➢ Siempre debe aparecer en primer lugar #PCDATA

➢ Es obligatorio especificar una lista de opciones

➢ No se pueden aplicar sufijos de repetición a los elementos hijos

➢ Es obligatorio el sufijo asterisco para el grupo de elementos

Por ejemplo, una posible declaración para el elemento párrafo p en XHTML donde estarían las etiquetas para negrita <b> y cursiva <i>:

<!ELEMENT p (#PCDATA | b | i)* >

Aclaraciones y resumen de ELEMENT

- Los elementos se corresponden con los componentes estructurales de un documento, y definen su estructura lógica.
- Un elemento puede contener datos de tipo carácter, otros elementos o ambos a la vez.
- Los elementos pueden contenerse unos a otros, formando una jerarquía o árbol.
- Un documento XML siempre tiene un elemento raíz o ‘elemento documento’, que engloba a todos los demás.

- El elemento raíz se debe llamar igual que su tipo de documento (por ejemplo, si creamos un tipo de documento ‘articulo’, el elemento raíz deberá llamarse ‘articulo’
- El nombre de los elementos puede contener caracteres a-z, A-Z y _. El nombre no debe contener el carácter & o empezar con las letras X,M,L.
- Los nombres de elementos son sensibles a la diferencia entre mayúsculas y minúsculas

2.3 La declaración ATTLIST

Las reglas ATTLIST son las que permiten definir los atributos de los elementos:

```
<!ATTLIST elemento atributo tipo_dato condicionante

                  ......

                  atributoN tipo_datoN condicionanteN>
```

donde:

➢ elemento es el elemento que contiene el atributo.

➢ atributo es el atributo que estamos definiendo.

➢ condicionante puede ser:

- #IMPLIED el atributo será opcional.
- #REQUIRED el atributo será obligatorio.
- #FIXED "valor" el valor del atributo será constante, siempre el especificado. Si aparece deberá tener ese valor.
- "valor" valor predeterminado.

➢ tipo_dato puede ser uno de los siguientes:

- **CDATA** (Character DATA, no confundir con #PCDATA) son cadenas de caracteres. Se respetan los espacios porque no son modificadas por el parser. Los únicos caracteres que no pueden aparecer son los especiales &<">'
- **Tipo de dato enumerado** que se representa entre paréntesis y separados

con barras: (1 | 2 | 3 | 4 | 5). Un tipo de dato enumerado indica que el atributo puede tomar como valor exclusivamente uno de los que aparecen enumerados. En este ejemplo, números enteros entre 1 y 5. Cada uno de los valores debe ser del tipo NMTOKEN.

- ID (identificador único) Los atributos de este tipo no podrán repetir valor en todo el documento, ni siquiera en elementos distintos. Identifican a su elemento con exclusividad, como el atributo id usado en HTML. Tiene que ser un identificador válido XML. En un mismo elemento no puede haber dos atributos de tipo ID.

En ocasiones necesitamos usar datos numéricos como atributos del tipo ID, por ejemplo el DNI. Al tener la restricción de tener que ser un identificador válido XML, necesitamos que comience por un carácter o un guión bajo. Habitualmente se suele utilizar el truco de anteponerle la letra del NIF o un guión bajo:

incorrecto: 32123123A

correcto: A32123123 ó _32123123A

- IDREF (identificador de otro elemento) Contiene un valor que es identificador (ID) de otro elemento. Permite establecer relaciones entre elementos del documento.

- IDREFS (lista de IDREF) Contiene varios identificadores separados por espacios. Es decir el valor del atributo es una serie de valores separados por espacios que coinciden con el valor del atributo ID de otros elementos.

- NMTOKEN (name TOKEN, nombre de token) el atributo sólo contiene letras, dígitos, y los caracteres punto ".", guión "-", subrayado "_" y dos puntos ":" . Difiere de un identificador XML en que cualquiera de los caracteres permitidos puede ser el primer carácter de un NMTOKEN mientras que un identificador XML solamente puede comenzar por una letra. Todo identificador XML es NMTOKEN pero no a la inversa.

- NMTOKENS (lista de NMTOKEN) Contiene varios nmtoken separados por espacios. El atributo sólo contiene letras, dígitos, y los caracteres punto ".", guión "-", subrayado "_", dos puntos ":" (como el tipo NMTOKEN) y también espacios en blanco.

- ENTITY permite especificar un atributo cuyo valor sea una entidad.

Algunos ejemplos son :

EJEMPLO 1:

```
<!ELEMENT texto (#PCDATA)>
<!ATTLIST texto idioma CDATA #REQUIRED>
<!ELEMENT mensaje (de, a, texto)>
<!ATTLIST mensaje prioridad (normal | urgente) “normal”>
```

donde se indica que el atributo idioma pertenece al elemento texto pudiendo contener cualquier carácter, mientras que el atributo prioridad pertenece al elemento mensaje y puede tener el valor “normal” o “urgente”, siendo “normal” el valor por defecto si no se especificara este atributo en el documento.

EJEMPLO 2:

Respecto a la palabra clave #REQUIRED significa que no tiene valor por defecto, por lo que es obligatorio especificar este atributo. Si el atributo es opcional, se usa la clave #IMPLIED ya que a veces interesa que se pueda omitir un atributo, sin que se adopte automáticamente un valor por defecto; por ejemplo:

```
<!ATTLIST IMG URL CDATA #REQUIRED>
<!ATTLIST IMG ALT CDATA #IMPLIED>
```

expresa que el atributo “URL” es obligatorio, mientras en el atributo “ALT” es opcional y si se omite, no se toma ningún valor por defecto.

La tercera palabra clave #FIXED especifica que el valor del atributo es constante, y no puede cambiar a lo largo del documento. Al escribir:

```
<!ATTLIST código postal CDATA #FIXED “46110”>
```

se indica que 46110 es el único código postal que se va a utilizar.

EJEMPLO 3:

Supóngase que al relacionar la plantilla de una empresa a cada empleado se le asigna su número de seguridad social (nss), para ello si se declara en la DTD :

```
<!ELEMENT empleado (#PCDATA)>
<!ATTLIST empleado nss ID #REQUIRED>
```

se trata de indicar que cada empleado tenga su nss y que dos empleados no pueden tener el mismo. Debe empezar con letra y no puede contener espacios.

EJEMPLO 4:

IDREF. Mientras ID identifica a un elemento, IDREF apunta a un elemento con un atributo ID. Por ejemplo, para implementar un sistema de hipervínculos en un documento podemos utilizar :

```
<!ELEMENT enlace EMPTY>
<!ATTLIST enlace destino IDREF #REQUIRED>
```

El tipo de ID y el mecanismo de referencia es muy parecido al utilizado en los sistemas de gestión de bases de datos y de hecho, las estructuras en las DTDs pueden llegar a ser casi tan complicadas como las claves lo son en dichos sistemas.

EJEMPLO 5:

Enumeraciones. Así la expresión :

```
<!ATTLIST telefono lugar (oficina | movil | particular) “oficina”>
```

proporciona la localización de un número de teléfono y si éste dato no está presente, la localización del número de teléfono se asume por defecto que es la de la “oficina”.

EJEMPLO 6:

NMTOKEN (Autenticaciones) son parecidos a las cadenas CDATA, aunque imponen restricciones sobre los valores de los atributos, de forma que sólo aceptan caracteres válidos para nombrar cosas (letras, números, puntos, guiones, subrayados y los dos puntos). En el ejemplo :

```
<!ATTLIST pais población NMTOKEN #REQUIRED>
```

pais población =”500000” sería aceptado, pero no lo sería si el valor fuera escrito de la forma “5 000 000”debido a la presencia de espacios en blanco.

EJEMPLO 7:

<!ATTLIST libro isbn CDATA #IMPLIED>

El elemento libro tendrá un atributo isbn opcional cuyo valor será una cadena de caracteres.

EJEMPLO 8:

<!ATTLIST autor sexo (masculino|femenino) #REQUIRED

nacionalidad CDATA #IMPLIED>

El elemento autor tendrá dos atributos: sexo que será obligatorio y que sólo podrá tomar los valores “masculino” o “femenino” y nacionalidad que será opcional y cuyo valor será

una cadena de caracteres.

EJEMPLO 9:

<!ATTLIST autor nacionalidad CDATA "español">

El elemento autor tiene un atributo nacionalidad. Si el documento no incluye ese atributo se asume que por defecto el valor será “español”. Naturalmente el atributo es opcional aunque no se diga explícitamente.

NOTA: el orden en que se definen los elementos es relevante, es decir, el documento XML tendrá que respetar ese orden para ser validado. El orden de los atributos NO es relevante.

EJEMPLO: ATRIBUTOS DEL TIPO ID, IDREF E IDREFS

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE familia [
        <!ELEMENT familia (persona)*>
        <!ELEMENT persona (nombre)>
        <!ELEMENT nombre (#PCDATA)>
        <!ATTLIST persona  id ID  #REQUIRED
                                  madre IDREF  #IMPLIED
                                  padre IDREF   #IMPLIED
                                  hijo IDREFS   #IMPLIED>
]>
<familia>
<persona id="Ana" madre="Maria" padre="Juan">
     <nombre>Ana Sánchez</nombre>
</persona>
<persona id="Juan" hijo="Ana Pepe">
     <nombre>Juan Sánchez</nombre>
</persona>
<persona id="Maria" hijo="Ana Pepe">
     <nombre>María García</nombre>
</persona>
<persona id="Pepe" madre="Maria" padre="Juan">
     <nombre>Pepe García</nombre>
</persona>
</familia>
```

ATRIBUTOS ESPECIALES

Son atributos reservados importantes para XML: xml:space y xml:lang. Estos atributos especiales, como cualquier otro atributo, deberán ser declarados para poder usarlos.

xml:space indica a una aplicación XML si el espacio en blanco dentro del elemento es significativo. Para conservar los espacios en blanco el atributo xml:space toma el valor preserve, para aplicar el tratamiento de espacios en blanco por defecto de la aplicación se usa el valor default.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE poema [
        <!ELEMENT poema (#PCDATA)>
        <!ATTLIST poema autor CDATA #REQUIRED
                        xml:space (default| preserve) 'preserve'
                        xml:lang NMTOKEN 'es'>
]>
<poema autor='Miguel Hernández'>
    La libertad es algo
    que sólo en tus entrañas
    bate como el relámpago.
</poema>
```

En este ejemplo el elemento poema puede tener 3 atributos, pero dos de ellos tienen asignado un valor por defecto que es el que se aplica en el caso de que no aparezcan. xml:lang="es" significa que el poema está escrito en castellano. xml:space="preserve" indica que deben respetarse los saltos de línea puesto que se trata de versos.

NOTA: Cuando en los lenguajes de marcas hablamos de espacios, estamos incluyendo espacios, tabuladores y saltos de línea.

2.4 Uso de elementos o de atributos

Los atributos deben ayudar a clarificar cuál es el contenido de los elementos XML y además, pueden ayudar a definir qué es lo que hace un elemento y como se relaciona con otros elementos.

Para distinguir entre elemento y atributo podemos respondernos a una serie de sencillas cuestiones:

- ¿Estamos definiendo un aspecto particular de un elemento, como puede ser el tamaño, color, peso, etc?
- ¿Es necesaria alguna forma de proporcionar mas información acerca de una instancia de un elemento particular?
- ¿Quieres estar seguro de que, cada vez que se añade un elemento, también se incluye con el mismo algún tipo de información?

Ejemplo:

```
<Libro formato='bolsillo'>Los pilares de la tierra</Libro>
```

2.5 Declaración de Entidades

En general entidad se refiere a un objeto usado para guardar información y por ello necesariamente cada documento tiene al menos la entidad del propio documento. Su razón de ser es doble, permitir guardar un contenido que puede usarse muchas veces y poder descomponer un documento grande en subconjuntos más manejables.

Una entidad consiste en un nombre y su valor (son similares a las constantes en los lenguajes de programación). Con algunas excepciones, el procesador XML sustituye las referencias a entidades por sus valores antes de procesar el documento. Una vez definida la entidad, se puede utilizar en el documento escribiendo una referencia a la entidad, que empieza con el caracter "&", sigue con el nombre de la entidad y termina con ";". (es decir, &nombreEntidad;)

Declaración de entidades

- Entidades internas:

```
<!ENTITY nombreEntidad "valorEntidad">
```

- Entidad externa (archivo de texto):

```
<!ENTITY nombreEntidad SYSTEM "uri">

<!ENTITY nombreEntidad PUBLIC "fpi" "uri">
```

- Entidad externa (archivo no de texto):

```
<!ENTITY nombreEntidad SYSTEM "uri" NDATA tipo>

<!ENTITY nombreEntidad PUBLIC "fpi" "uri" NDATA tipo>
```

- Entidades paramétricas:

```
<!ENTITY % nombreEntidad "valorEntidad">

<!ENTITY % nombreEntidad SYSTEM "uri">

<!ENTITY % nombreEntidad SYSTEM "uri" NDATA tipo>
```

En todos estos casos:

- "nombreEntidad" es el nombre de la entidad.
- "valorEntidad" es el valor por el que se sustituye la entidad.
- "uri" es el camino (absoluto o relativo) hasta un archivo.
- "tipo" es el tipo de archivo (gif, jpg, etc).
- "fpi" es un indentificador público formal (Formal Public Identifier).

Declaración de una entidad

Todas las declaraciones de entidades,independientemente del tipo que sean, utilizan la misma

sintaxis:

```
<!ENTITY nombre "contenido">
```

donde nombre es el identificador de la entidad y el contenido es su contenido o una referencia a éste.

Tipos de entidades

Debido a que las entidades XML pueden hacer tantas cosas, existen muchas variedades de entidades.

Existen tres propiedades que definen el tipo:

Dependiendo de si la **entidad** debe ser analizada (Contiene XML) o no (no es XML sino datos binarios, texto.. contiene NDATA en la declaración de la entidad) por el parser, una entidad se puede clasificar como **analizada** o **no analizada**.
Dependiendo de si el contenido de la **entidad** es interno o externo es decir de si su contenido aparece en la declaración o no, la entidad se puede clasificar como **interna** o **externa**.
Dependiendo de si se utiliza en la DTD o en el documento se clasifica como **paramétrica** o **general.**

EJEMPLO 1: ***Entidad general interna analizada.***

Son las más sencillas de todas y son básicamente abreviaturas totalmente definidas en la sección de declaración de tipo de documento.

Ejemplo1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejemplo1 [
<!ENTITY nomclub "Club XMLiense">
<!ENTITY copy "&#169;">
<!ELEMENT ejemplo1 (#PCDATA)>
]>
<ejemplo1>
      En el &nomclub; realizamos cursos sobre XML. &copy; Lenguaje de Marcas.
</ejemplo1>
```

Cuando el parser lea el documento cambiará la llamada a la entidad &nomclub; por su contenido.

Esta entidad es:

- General al ser utilizada en un documento XML.
- Interna al aparecer su contenido en la declaración de la entidad.
- Analizada porque contiene XML.

Este tipo de entidades es muy utilizado también cuando es necesario emplear algún carácter que el juego de caracteres que hemos definido en el atributo `encoding` de la declaración XML no soporta. Por ejemplo, si quisiéramos colocar el símbolo del copyright © .

EJEMPLO 2: *Entidad general externa analizada*

En este caso la declaración no lleva implícito el contenido de la entidad. Lleva un identificador externo que indicará a la aplicación XML dónde se encuentra el contenido de la entidad.

Este identificador externo es la palabra SYSTEM seguida de la dirección (URI) donde se encuentra el recurso.

Ejemplo2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libro [
<!ENTITY capitulo1 SYSTEM "capitulo1.xml">
<!ELEMENT libro (capitulo)>
<!ELEMENT capitulo (para)>
<!ELEMENT para (#PCDATA)>
]>
<libro>
      &capitulo1;
</libro>
```

capitulo1.xml

```
<capitulo>
      <para>Este es el primer capitulo</para>
</capitulo>
```

Las posibilidades de esta forma de trabajar son enormes, ya que:

- Esta misma entidad podría ser utilizada desde múltiples documentos XML.
- Permite que diferentes autores desarrollen diferentes partes de un documento XML de forma independiente,no teniéndose que preocupar del conjunto del documento XML.

!Cuidado al ser externa no se visualizan los datos en el navegador! Pero si la analiza el parser.

EJEMPLO 3: *Entidad parámetro interna*

Como ya hemos dicho,las entidades XML se clasifican según puedan usarse en una DTD o en el documento. Las entidades que sólo pueden utilizarse en la DTD se denominan entidades parámetro.

Están diseñadas para contener listas de atributos y modelos de contenido. Nos ayudarán a organizar y agilizar la creación de las DTD, y hacerla más eficiente y consistente.

Lo que nos permiten las entidades paramétricas es definir este modelo de contenido como una entidad y luego hacer referencia a ella tantas veces como sea necesario. De esta manera si necesitamos modificar estos elementos, tendremos que hacerlo sólo una vez, en un único lugar, y no tener que ir elemento por elemento realizando la modificación.

Las entidades paramétricas tienen un identificador especial en su declaración que las diferencia de las generales. Este indicador es el símbolo de tanto por ciento (%), y se usa precediendo a un espacio en blanco y después el nombre.

```
<!ENTITY % nombre "contenido">
```

También cambia la forma de referenciarla. En lugar de colocar el nombre entre el & y ; se coloca entre % y el ;.

Ejemploentidadinterna.dtd

```
<!ENTITY  % meta "(titulo | autor)">
<!-- Elemento raiz: articulo -->
<!ELEMENT articulo (metainfo, cuerpo)>
<!ELEMENT metainfo ( %meta; , clave, resumen)>
<!ELEMENT cuerpo (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT clave (#PCDATA)>
<!ELEMENT resumen (#PCDATA)>
```

Ejemplo3int_param.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE articulo SYSTEM "ejemploentidadinterna.dtd">
<articulo>
     <metainfo>
            <titulo>ll</titulo>
            <clave>33</clave>
            <resumen>JDJJDJD</resumen>
      </metainfo>
    <cuerpo>Antonio</cuerpo>
</articulo>
```

Hay que tener en cuenta que las dtd que incluyan entidades no pueden estar incrustadas en el documento XML sino vinculadas.De lo contrario aparecen errores al validar.

EJEMPLO 4: *Entidad parámetro externa*

Recordar: Hay que tener en cuenta que las dtd que incluyan entidades no pueden estar incrustadas en el documento XML

ejemplo4_ext_param.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE articulo SYSTEM "ejemplo4a.dtd">
<articulo>
     <metainfo>
            <titulo>ll</titulo>
            <clave>33</clave>
            <resumen>JDJJDJD</resumen>
      </metainfo>
    <cuerpo>Antonio</cuerpo>
</articulo>
```

ejemplo4a.dtd

```
<!ENTITY % clav SYSTEM "ejemplo4b.dtd">
%clav;
<!ENTITY  % meta "(titulo | autor)">
<!-- Elemento raiz: articulo -->
<!ELEMENT articulo (metainfo, cuerpo)>
<!ELEMENT metainfo ( %meta; , clave, resumen)>
<!ELEMENT cuerpo (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT resumen (#PCDATA)>
```

ejemplo4b.dtd

```
<!ELEMENT clave (#PCDATA)>
```

UN EJEMPLO REAL

A continuación se muestra una pequeña parte del DTD para HTML 4.01 Strict, ligeramente alterada para destacar lo más adecuado a nuestros conocimientos. Puedes ver el DTD completo en la especificación de la W3C para HTML 4.01 Strict.

```
<!ELEMENT TABLE (CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, TBODY+)>
<!ELEMENT CAPTION (%inline;)*>
<!ELEMENT THEAD (TR)+>
<!ELEMENT TFOOT (TR)+>
<!ELEMENT TBODY (TR)+>
<!ELEMENT COLGROUP (COL)*>
<!ELEMENT COL EMPTY>
```

```
<!ELEMENT TR (TH|TD)+>
<!ELEMENT (TH|TD) (%flow;)*>

<!ATTLIST TABLE   %attrs;
                  summary %Text; #IMPLIED
                  width %Length; #IMPLIED
                  border %Pixels; #IMPLIED
                  frame %TFrame; #IMPLIED
                  rules %TRules; #IMPLIED
                  cellspacing %Length; #IMPLIED
                  cellpadding %Length; #IMPLIED
                  %reserved;
                  datapagesize CDATA #IMPLIED>
```

2.6 Declaración de Anotación

La palabra inglesa notation tiene entre sus significados el de “notas” o “anotación”, dando la idea de proporcionar información adicional, cosa necesaria en un DTD para detallar el significado de los datos en un atributo o en una entidad.

Para poder utilizar esta información, antes hay que proceder a su declaración, cuya sintaxis general es de la forma:

<!NOTATION nombre SYSTEM “información de notación”>

El nombre asociado a NOTATION corresponde a la declaración de entidad o de atributo al que se refiere y el parámetro “información de notación” se pasa a la aplicación correspondiente. Esta información puede ser una simple palabra clave (como “gif”), un URL, o cualquier otro tipo de descripción. Cuando se define una información adicional, esta anotación puede ser de distintas clases: pública, un identificador del sistema que especifica documentación de la propia anotación, una especificación formal, o un asistente de la aplicación que contenga objetos representados en la anotación.

Son ejemplos de ello:

```
<!NOTATION HTML SYSTEM “http://www.w3.org/Markup”>
<!NOTATION HTML PUBLIC “//W3C//DTD HTML 4.0 Transitional//EN”>
<!NOTATION gif SYSTEM “gif”>
```

Distingamos su uso en atributos y entidades. Para el caso de hacerlo dentro de atributos, las anotaciones se declaran añadiendo NOTATION a ATTLIST, lo que permite al autor del documento especificar en el momento de declarar el atributo que su valor se ajusta a una anotación dada.

Son ejemplos :

```
<!ATTLIST fecha NOTATION (ISODATE | EUROPEANDATE) #REQUIRED>
```

```
<!ATTLIST imagen fuente CDATA #REQUIRED tipo NOTATION (gif) #REQUIRED>
```

Donde en el segundo ejemplo se observa que se está en condiciones de usar “gif” como un atributo.

Para su uso dentro de una entidad, el esquema que se sigue es análogo al anterior y se lleva a cabo incorporando en la entidad externa la palabra clave NDATA. Veamos el uso de ello en el siguiente ejemplo: supóngase que se quiere incluir en la DTD un logo de una compañía del que se tienen dos versiones, una en GIF y otra en JPEG. Se empieza con las correspondientes declaraciones de la anotación :

```
<!NOTATION gif SYSTEM “gif”>
<!NOTATION jpeg SYSTEM “jpeg”>
```

Y a continuación, ésta se incluye en las entidades correspondientes, con las notaciones de “GIF” o “JPEG” de la forma :

```
<!ENTITY logogif SYSTEM “imagenes/compañialogo.gif” NDATA gif>
```

ejemplo5.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE facturas[
<!-- Notation que indica el programa para procesar jpg -->
<!NOTATION jpg SYSTEM "aplicaciones/visordeimagenes.ini">

<!-- Entidades con las imágenes de las carátulas de las películas-->
<!ENTITY foto_p0360 SYSTEM "caratulas/p0360.jpg" NDATA jpg>
<!ENTITY foto_p0437 SYSTEM "caratulas/p0437.jpg" NDATA jpg>
<!ENTITY foto_p1201 SYSTEM "caratulas/p1201.jpg" NDATA jpg>

<!-- Elemento raiz: facturas -->
<!ELEMENT facturas (factura*)>

<!-- Elemento factura -->
<!ELEMENT factura (datos_cliente, datos_factura)>

<!-- Elemento datos_cliente -->
<!ELEMENT datos_cliente (nombre, apellido, apellido, dni, tfno)>
<!ATTLIST datos_cliente
  ident ID #REQUIRED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT dni (#PCDATA)>
<!ELEMENT tfno (#PCDATA)>

<!-- Elemento datos_factura -->
<!ELEMENT datos_factura (resguardo, (alquileres | compras |
(alquileres,compras)))>

<!-- Elemento resguardo -->
<!ELEMENT resguardo (forma_pago, importe_total)>
```

```
<!ELEMENT forma_pago (#PCDATA)>
<!ELEMENT importe_total (#PCDATA)>

<!-- Elemento alquileres -->
<!ELEMENT alquileres (fecha, peliculas)>
<!-- Elementos fecha -->
<!ELEMENT fecha (#PCDATA)>
<!-- Elemento peliculas -->
<!ELEMENT peliculas (pelicula+)>

<!-- Elemento pelicula -->
<!ELEMENT pelicula (titulo, genero, duracion, actores)>
<!ATTLIST pelicula
  id_pelicula ID #REQUIRED
  valoracion CDATA ""
  caratula ENTITY #IMPLIED>

<!ELEMENT titulo (#PCDATA)>
<!ELEMENT genero (#PCDATA)>
<!ELEMENT duracion (#PCDATA)>
<!-- Elemento actores -->
<!ELEMENT actores (actor, actor, actor)>
<!ELEMENT actor (nombre, apellido, apellido)>

<!-- Elemento compras -->
<!ELEMENT compras (dvds | cintas | (dvds,cintas))>
<!-- Elemento dvds -->
<!ELEMENT dvds (dvd+)>
<!-- Elemento dvd -->
<!ELEMENT dvd (extras?, titulo, fecha_salida_mercado)>
<!ELEMENT extras EMPTY>
<!ELEMENT fecha_salida_mercado (#PCDATA)>
<!-- Elemento cintas -->
<!ELEMENT cintas (cinta+)>
<!-- Elemento cinta -->
<!ELEMENT cinta (titulo, formato, rebobinado?)>
<!ELEMENT formato (#PCDATA)>
<!ELEMENT rebobinado EMPTY>
]>

<!-- XML **********************************/-->
<facturas>
   <factura>
      <datos_cliente ident="c01">
    <nombre>Antonio</nombre>
<apellido>Moreno</apellido>
<apellido>Flores</apellido>
<dni>123456789X</dni>
<tfno>916663322</tfno>
 </datos_cliente>

 <datos_factura>
    <resguardo>
   <forma_pago>efectivo</forma_pago>
<importe_total>35</importe_total>
</resguardo>
<alquileres>
```

```
   <fecha>12/01/2007</fecha>
<peliculas>
  <pelicula id_pelicula="p320" caratula="foto_p0360">
     <titulo>AQUELLOS DIAS</titulo>
 <genero>Comedia</genero>
 <duracion>97min</duracion>
 <actores>
    <actor>
   <nombre>Luke</nombre>
               <apellido>Wilson</apellido>
               <apellido></apellido>
</actor>
    <actor>
<nombre>Will</nombre>
               <apellido>Farrel</apellido>
               <apellido></apellido>
</actor>
    <actor>
<nombre>Vince</nombre>
               <apellido>Vaughn</apellido>
               <apellido></apellido>
</actor>
 </actores>
   </pelicula>
</peliculas>
</alquileres>
<compras>
   <dvds>
  <dvd>
       <titulo>El oro de Moscú</titulo>
       <fecha_salida_mercado>2006</fecha_salida_mercado>
  </dvd>
</dvds>
<cintas>
  <cinta>
     <titulo>Gladiator</titulo>
 <formato>VHS</formato>
  </cinta>
</cintas>
</compras>
 </datos_factura>
   </factura>
</facturas>
```