Unidad 5a. Definición del Tipo de Documento (DTD)

1 Definición de documentos XML

Un documento XML puede contener muchos tipos de información. Es decir, pueden haber muchos lenguajes escritos en XML para cualquier colectivo de usuarios. Por ejemplo,

- Si lo utiliza el colectivo de médicos podría crear un lenguaje en XML específico para almacenar diagnósticos de los pacientes. Este lenguaje se podría llama PacientesML.
- Si los distribuidores de películas utilizan XML podrán crear sus propios lenguajes para guardar la información de las películas. Este lenguaje se podría llamar PeliculasML.
- Si estamos escribiendo aplicaciones para móviles podremos utilizar un lenguaje para aplicaciones inalámbricas (Wireless), que se llama WML.

Como vemos, se pueden crear infinitos lenguajes a partir del XML. Para especificar cada uno de los usos de XML, o lo que es lo mismo, para especificar cada uno de los sublenguajes que podemos crear a partir de XML, se utilizan unos lenguajes propios.

Ya sabemos que los documentos XML deben cumplir una serie de reglas sintácticas definidas en el estándar para estar bien formados. Además de las reglas de buena formación, podemos establecer para nuestros documentos XML descripciones formales, que definen ciertas propiedades y restricciones adicionales (por ejemplo, qué elementos y qué atributos están permitidos en el documento y cuáles son los contenidos que éstos pueden tomar).

Para especificar estas descripciones formales para documentos XML existen dos opciones, dos metalenguajes que podemos utilizar: DTD y XML Schema

- **DTD (Document Type Definition):** es un metalenguaje definido ya para SGML que permite describir la estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información
- **XML Schema :** es un lenguaje escrito en XML, desarrollado con posterioridad a las DTD, y pensado para proporcionar mayor potencia expresiva que DTD. Su principal diferencia con las DTD es que XML soporta la definición de tipos de datos.

En este tema estudiamos las DTD, y en temas posteriores veremos XML Schema.

2 Aspectos generales

XML permite definir para nuestros documentos ciertas propiedades que los convierten en especiales, diferenciándolos de otros. Todas esas restricciones y características, que el autor de un documento puede definir, se engloban bajo un nombre que las representa, el nombre del **tipo del documento**.

La principal ventaja de utilizar una definición tipo es garantizar la consistencia de los documentos. Los documentos que tienen un tipo definido, para ser **válidos**, deben cumplir con las reglas establecidas en la definición del tipo. Recordemos que no es lo mismo un documento válido (el que cumple con la definición de un tipo) que un documento bien formado (el que cumple con las reglas sintácticas de XML).

Una Definición de Tipo de Documento (DTD) permite restringir el contenido de los elementos y los atributos de un documento, especificar el orden en el que deben aparecer, su obligatoriedad, etc. Estas reglas son comprobadas por el procesador XML, y así permi-

ten realizar un filtrado de los documentos que el procesador debe admitir. Esto asegura que cumplen una serie de requisitos o cualidades y, en definitiva, certifica la calidad de los documentos XML. Esto conlleva resultados más fiables y mayor sencillez a la hora de programar las aplicaciones que procesarán los documentos, ya que se trabaja a partir de una base de seguridad y no será necesario contemplar ciertas situaciones de error.

La DTD también es útil para detectar y corregir errores en la fase de elaboración del documento, posiblemente cuando aún se está a tiempo de solucionar la falta con poco esfuerzo. Esta utilidad tiene su sentido sobre todo en entornos manuales en los que es fácil que el operador se confunda y, por ejemplo, olvide rellenar un campo. ¿Qué haríamos con un pedido en el que no se ha especificado el destinatario?

Además, una declaración de tipo del documento nos permite otras cosas como definir nuestras propias entidades o establecer valores por defecto para los atributos.

Para especificar un tipo para un documento XML hay que incluir dos cosas:

- La **declaración del tipo de documento**: es la parte donde decimos a qué tipo pertenece nuestro documento, y se incluye en el prólogo del documento XML
- La **definición del tipo de documento**: es la parte donde se definen todas las reglas que aplican al tipo mediante ciertas estructuras especiales denominadas **declaraciones de marcado**.

Las declaraciones de marcado incluidas en la DTD pueden ser:

- Internas: se encuentran dentro del propio documento XML; forman lo que se denomina el **subconjunto interno**.
- Externas: se encuentran en un fichero separado; forman el **subconjunto externo**
- Una combinación de ambas.

2.1 La Declaración del tipo de documento

La declaración del tipo de documento especifica el nombre del tipo, y se incluye dentro del prólogo del documento XML, mediante la referencia !DOCTYPE, de la siguiente manera:

```
<!DOCTYPE NombreXML ... >
```

donde “NombreXML” es **el nombre del tipo, que debe coincidir con el nombre del elemento raiz** del documento XML:

```
<!DOCTYPE NombreXML ... >
<NombreXML>
        ...
</NombreXML>
```

La declaración de tipo se sitúa entre la declaración XML (en el caso de que esté presente) y la etiqueta de inicio del primer elemento (el elemento raíz). Veamos un ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Casas_Rurales ... >
<Casas_Rurales>
        ...
</Casas_Rurales>
```

El formato concreto de la declaración del tipo varía según se trate del subconjunto interno o subconjunto externo, como veremos más adelante.

2.2 La definición del tipo de documento

Toda declaración de tipo propiamente dicha para un documento se complementa con una **definición del tipo** (abreviada como **DTD**) que asocia al tipo las cualidades que posee. La DTD define los tipos de elementos, atributos, entidades y notaciones que se podrán utilizar en el documento, así como ciertas restricciones estructurales y de contendido, valores por defecto, etc. Para formalizar todo ello XML provee ciertas estructuras especiales, las llamadas **declaraciones de marcado** y que pertenecen a alguno de los siguientes tipos:

- Declaraciones de tipos de elementos.
- Declaraciones de listas de atributos para los tipos de elementos.
- Declaraciones de entidades.
- Declaraciones de notación.

Veremos cada uno de estos tipos de declaración durante el desarrollo de este tema.

2.3 Subconjunto interno

Como ya sabemos, si las declaraciones de marcado están incluidas dentro del documento de partida forman el llamado subconjunto interno. En este caso, dichas declaraciones se incluyen dentro de unos corchetes que siguen a la declaración del tipo del documento, de la siguiente manera:

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

El subconjunto interno contiene declaraciones que pertenecen únicamente a un documento, que son específicas para él y que no es posible compartir.

Para ilustrar esto, vamos a añadir algunas declaraciones de tipos de elementos internas al ejemplo anterior. No es necesario entender todas las declaraciones en este momento, ya las iremos explicando todo poco a poco.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Casas_Rurales [
<!ELEMENT Casas_Rurales (Casa)*>
  <!ELEMENT Casa (Dirección, Descripción, Estado, Tamaño)>
  <!ELEMENT Dirección (#PCDATA) >
  <!ELEMENT Descripción (#PCDATA) >
  <!ELEMENT Estado (#PCDATA) >
  <!ELEMENT Tamaño (#PCDATA) >
]>
<Casas_Rurales>
        ...
</Casas_Rurales>
```

La definición del tipo anterior especifica que el tipo de documento `Casas_Rurales` está formado por elementos de tipo `Casa`. Los elementos de tipo `Casa` contienen a su vez elementos de tipo `Dirección`, `Descripción`, `Estado` y `Tamaño`, en este orden y sin faltar ninguno. El contenido de estos elementos está formado exclusivamente por datos carácter (#PCDATA).

Para acabar de completar el documento XML del ejemplo, incluiremos el contenido del ejemplar del documento. Se puede comprobar cómo el ejemplar cumple con las restricciones de tipo, estructura y contenido especificadas.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Casas_Rurales [
  <!ELEMENT Casas_Rurales (Casa)*>
        <!ELEMENT Casa (Dirección, Descripción, Estado, Tamaño)>
        <!ELEMENT Dirección (#PCDATA) >
        <!ELEMENT Descripción (#PCDATA) >
        <!ELEMENT Estado (#PCDATA) >
        <!ELEMENT Tamaño (#PCDATA) >
]>
<Casas_Rurales>
    <Casa>
         <Dirección>Calle Las Palmas 23. La Sagra. Toledo</Dirección>
         <Descripción>
            Se trata de una casa del siglo XVII, ...
         </Descripción>
         <Estado>El estado de conservación es magnífico, ...</Estado>
         <Tamaño>La casa tiene 259 metros cuadrados, ...</Tamaño>
    </Casa>
    <Casa>
         <Dirección>Calle Or 5, Fregenal de la Sierra. Badajoz
         </Dirección>
         <Descripción>
             De arquitectura espectacular la casa ...
         </Descripción>
         <Estado>Recientemente restaurada su estado actual es ...
         </Estado>
         <Tamaño>La superficie habitable es de ...</Tamaño>
        </Casa>
</Casas_Rurales>
```

2.4 Subconjunto externo

Las declaraciones de marcado también pueden encontrarse fuera del documento XML. Estas declaraciones de marcado externas pueden referenciarse de varias maneras:

- Mediante una declaración explícita de subconjunto externo.
- Mediante entidades parámetro externas.

La segunda forma se estudiará más adelante en el apartado dedicado a las entidades y a las declaraciones de entidades.

Normalmente el subconjunto externo está formado por declaraciones comunes que se comparten entre múltiples documentos XML que pertenecen al mismo tipo. Se escriben y modifican una vez en lugar de tener que repetirla y modificarla muchas veces.

Ahora los corchetes pierden su sentido y a cambio se impone utilizar algún sistema de referencia que permita localizar las declaraciones de marcado externas. Para el subconjunto externo la declaración del tipo del documento puede tomar alguna de las siguientes formas:

```
<!DOCTYPE NombreXML SYSTEM "URI" >
<!DOCTYPE NombreXML PUBLIC "id_publico" "URI" >
```

En la primera, detrás de la palabra SYSTEM se especifica un URI donde se pueden encontrar las declaraciones. En la segunda forma se especifica también un identificador, que puede ser utilizado por el procesador XML para intentar generar un URI alternativo, posiblemente basado en alguna tabla. Hay que notar que también es necesario (obligatorio) especificar un URI.

Podemos transformar el ejemplo de las casas rurales para que todas las declaraciones de los elementos sean externas al documento. El URI en este ejemplo es absoluto pero podría haberse empleado uno relativo, si se desea probar el ejemplo se puede cambiar el URI a uno relativo que incluyera únicamente el nombre del fichero **casasrurales.dtd**, situando ambos ficheros en el mismo directorio.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE Casas_Rurales SYSTEM
"http://www.casasrurales.com/casasrurales.dtd">
<Casas_Rurales>
  ...
</Casas_Rurales>
```

Siendo el contenido del fichero **casasrurales.dtd**:

```
  <!ELEMENT Casas_Rurales (Casa)*>
  <!ELEMENT Casa (Dirección, Descripción, Estado, Tamaño)>
  <!ELEMENT Dirección (#PCDATA) >
  <!ELEMENT Descripción (#PCDATA) >
  <!ELEMENT Estado (#PCDATA) >
  <!ELEMENT Tamaño (#PCDATA) >
```

La primera línea del fichero es una declaración de texto, opcional, que especifica que las siguientes líneas son XML y la versión con la que cumplen. Incluye también una declaración de codificación. Las declaraciones de texto no pueden incluir una declaración de documento autónomo, no tendría sentido.

Destacar que en el contenido no puede aparecer la declaración de un nuevo DTD

La extensión "**.dtd**" del archivo es sólo una convención, no hay ninguna obligación en que así sea.

Recordemos que un documento XML puede incluir en la declaración XML el campo “standalone”, con el que indicamos si el documento es autónomo o no. Observar que si se in-

cluye un subconjunto externo el documento ya no puede ser autónomo, y por tanto el valor de “standalone” debe ser “no”, que es el valor por defecto.

2.5 Orden de evaluación de las DTD's

Normalmente la DTD se compone, como muestra el ejemplo, de una mezcla de definiciones internas y externas:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE Casas_Rurales SYSTEM "dtd/casasrurales2.dtd" [
        <!ELEMENT Casas_Rurales (Casa)*>
        <!ELEMENT Casa (Dirección, Estado, Tipo)>
        <!ELEMENT Dirección (#PCDATA) >
        <!ELEMENT Estado (#PCDATA) >
        <!ELEMENT Tipo (#PCDATA) >
]>
<Casas_Rurales>...</Casas_Rurales>
```

De momento, hasta que veamos las entidades parámetro externas, podemos pensar que el orden de procesamiento es el siguiente: se procesa primeramente el subconjunto interno (junto con el subconjunto externo implícito) y posteriormente el subconjunto externo explícito (SYSTEM).

Este orden permite sobrescribir declaraciones externas, compartidas entre varios documentos y ajustar la DTD a un documento específico. En los siguientes capítulos en los que estudiaremos más detalladamente los distintos tipos de declaraciones se incluyen ejemplos concretos que muestran la forma de proceder cuando existen declaraciones internas y externas.

2.6 Documentos de tipo válido

Los documentos que tienen declarado un tipo y además cumplen con él se denominan **válidos** o de **tipo válido**, los que tienen declarado un tipo pero no cumplen con él se denominan **no válidos** o de **tipo no válido**. Es preferible emplear las segundas denominaciones ya que las primeras (válido o no válido) parecen indicar que el documento "no sirve". Hasta el momento se ha trabajado con documentos que no podían denominarse válidos, ya que carecían de declaración de tipo, pero que nos eran muy útiles. Las expresiones de tipo válido o de tipo no válido dejan más claro este matiz.

Aquellos documentos XML que no incluyen una DTD no pueden clasificarse ni como de tipo válido ni como de tipo no válido, ya que no se puede decir que cumplan o que no cumplan con alguna DTD. Hay que destacar que un documento sea de tipo "no válido" no implica que esté mal formado, de hecho un documento XML para que sea considerado como válido debe estar primeramente bien formado.

No todos los procesadores XML comprueban si un documento es válido o no, sin embargo, todos comprueban que el documento esté bien formado, ya que de otra manera no podrían realizar un procesamiento fiable de los documentos XML. Que un procesador XML no compruebe la validez del documento no significa que la DTD para ese documento quede inservible, existen características de la DTD como la declaración de entidades y valores por defecto de los atributos de las que todavía se puede sacar provecho. A este respecto se añadirá algo más de información en capítulos posteriores, cuando se conozcan las declaraciones de marcado.

3 Declaraciones de tipos de elementos

Las "declaraciones de tipos de elementos" permiten definir qué elementos están permitidos para un documento y cuál es su contenido. Es decir, en lo que respecta a los elementos, para que un documento XML sea de tipo válido se debe cumplir que:

- Todos los elementos estén reconocidos mediante "declaraciones de tipos", o lo que es lo mismo, todos los elementos deben pertenecen a un tipo declarado.
- El contenido de cada elemento se ajusta a lo declarado en su tipo.

Así, las Declaraciones de Tipos de Elementos permitirán la detección de ciertos errores: inclusión de elementos no declarados, contenidos no válidos en un elemento, elementos obligatorios olvidados, elementos repetidos, etc.

Todas las declaraciones de tipos de elementos deben estar incluidas dentro de una DTD, ya sea interna o externa. El formato de una declaración de tipo de elemento viene recogido en las reglas [45] y [46] de la Recomendación de XML:

```
[45]  elementdecl ::=  '<!ELEMENT' S Nombre S contentspec S? '>'
[46]  contentspec ::=  'EMPTY' | 'ANY' | Mixed | children
```

```
Donde S representa cualquiera de los caracteres que se consideran espacios
en blanco en XML: el espacio, retorno de carro, salto de línea y tabulador.
```

Todas las Declaraciones de Tipo de Elementos se componen de una cadena “<!ELEMENT” (sin ningún espacio entre “!” y “ELEMENT”), seguida de uno o más espacios en blanco, un nombre XML, espacios en blanco, la especificación del contenido del elemento, espacios en blanco opcionales y el carácter de cierre “>”.

Los tipos de elementos se declaran de uno en uno, es decir, las Declaraciones de Tipos de Elementos son individuales. No es posible tampoco declarar un tipo de elemento dos veces, esto se considera un error de buena formación. Así en el ejemplo siguiente la única declaración válida, suponiendo que estuviera completa, es la primera. En la segunda se declaran dos tipos de elementos conjuntamente, y en la tercera se repite la declaración de un elemento ya existente:

```
<!DOCTYPE listado_discos [
        ...
        <!ELEMENT cinta ...>
        ...
        <!ELEMENT canción disco ...>
        ...     ...
        <!ELEMENT cinta ...>
        ...
]>
```

En cuanto a la especificación del contenido del elemento, vemos la regla [46] concreta cuatro modelos posibles. Un elemento puede ser vacío (**EMPTY**), tener cualquier contenido (**ANY**), una mezcla de valores (**Mixed**) o un conjunto de elementos hijos (**children**).

3.1 El tipo ANY

Este tipo no impone ninguna restricción, es decir un elemento declarado como tipo ANY puede contener cualquier combinación de datos carácter y elementos (declarados) sin restricciones.

Por ejemplo, en la siguiente DTD el elemento raíz se define como tipo ANY:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <!DOCTYPE listado_discos [
        <!ELEMENT listado_discos ANY>
        ...
  ]>
  <listado_discos>El disco ...</listado_discos>
```

3.2 El tipo EMPTY

Indica que el elemento no puede tener contenido. Siguiendo con el ejemplo anterior, añadimos un nuevo tipo de elemento, `disco`, de tipo vacío. Puede pensarse que un elemento de tipo vacío no tiene mucho sentido. Normalmente un elemento sin contenido tendrá atributos que aporten información y den sentido al elemento. Así, en el ejemplo se han añadido los atributos `titulo`, `autor` y `año` al elemento `disco`. Para que el documento sea de tipo válido, estos atributos habría que declararlos también en la DTD, pero esto lo veremos más adelante.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE listado_discos [
        <!ELEMENT listado_discos ANY>
        <!ELEMENT disco EMPTY>
        ...
]>
<listado_discos>
        <disco título="Maravillas" autor="Amaya Diosdado Sois" año="1974" />
        <disco título="Sin razón" autor="Federico Martín Martín" año="1988" />
        <disco título="Esperándote" autor="Andrea Sonjuan" año="2000" />
        ...
</listado_discos>
```

3.3 El tipo Mixed

Se trata de elementos formados por datos carácter sólo o entremezclados con otros elementos. Los datos carácter se representan con el identificador `#PCDATA`. Así el formato de una definición de elemento de contenido “mixed” puede ser uno de los siguientes:

a) Elemento que contiene sólo datos carácter (las siguientes definiciones son equivalentes):

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

O bien:

```
<!ELEMENT nombre_elemento (#PCDATA)*>
```

Los elementos declarados como de tipo PCDATA no pueden contener los caracteres siguientes:

- El caracter “<”, porque se interpretará como el comienzo de un nuevo elemento y el tipo PCDATA no puede contener etiquetas, ni elementos.
- El caracter “&”, porque se interpretará siempre como el comienzo de una entidad.
- La cadena “]]>”, porque marca el final de una sección CDATA, que estudiaremos más adelante.

Si se desean incluir estos caracteres, para que no se interpreten erróneamente, es necesario escaparlos, como ya se ha explicado. A parte de estas restricciones obligatorias no

existe forma de limitar el valor de un elemento PCDATA a un determinado grupo de caracteres.

Podemos modificar el ejemplo anterior para que el elemento disco contenga una descripción textual de cada disco:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE listado_discos [
        <!ELEMENT listado_discos ANY>
        <!ELEMENT disco (#PCDATA)>
         ...
]>
<listado_discos>
        <disco título="Maravillas" autor="Amaya Diosdado Sois" año="1974">
        El título de este disco es "Maravillas", su autor es Amaya Diosdado y
        se compone de 16 canciones,...
        </disco>
        <disco título="Sin razón" autor="Federico Martín Martín" año="1988">
        Disco titulado “Sin razón”...
        </disco>
        ...
</listado_discos>
```

En el ejemplo anterior se declara el documento como de tipo `listado_discos`, cuya definición podría decir lo siguiente: un documento de éste tipo contiene únicamente elementos de tipo `disco` y datos carácter en cualquier combinación (tipo ANY). A su vez cada elemento de tipo `disco` se compone de datos carácter (#PCDATA), no admitiendo otros elementos en su interior (no se podría, por ejemplo, anidar elementos de tipo disco).

b) Elemento que contiene datos carácter y otros elementos:

```
<!ELEMENT nombre_elemento (#PCDATA | elemento_1 | elemento_2 | ...)*>
```

El asterisco al final es obligatorio.

La aparición de los distintos elementos o de los datos carácter en un elemento concreto no es obligatoria y podrían no estar presentes. No se pueden establecer restricciones adicionales sobre el orden o sobre el número. Este modelo de contenido simplemente define una mezcla entre datos carácter y los elementos que se declararan. Por este motivo es un error especificar dos veces el mismo elemento en la declaración.

Podemos redefinir el elemento disco del ejemplo para que pueda contener otros elementos:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE listado_discos [
        <!ELEMENT listado_discos ANY>
        <!ELEMENT disco (#PCDATA | autor | título | canción)*>
        <!ELEMENT autor (#PCDATA)>
        <!ELEMENT título (#PCDATA)>
        <!ELEMENT canción (#PCDATA)>
]>
<listado_discos>
        Listado de discos de Federico Garcés.
        <disco>Este disco titulado <título>Maravillas</título> y cuyo autor
        es <autor>Amaya Diosdado</autor> se compone de varias canciones,
        todas con una melodía bastante pegadiza.
        La primera canción <canción>Ave María</canción> pretende ser un canto
        a la vida y está llena de notas y sonidos que evocan una tarde de   ve-
rano.
        La tercera canción <canción>Alegre Primavera</canción> recibió muy
        buenas críticas, ...</disco>

        <disco><título>América</título></disco>
        ...
</listado_discos>
```

La definición del tipo anterior especifica que el tipo `listado_discos`, puede contener únicamente elementos de tipo `disco` entremezclados con datos carácter. Los elementos de tipo `disco` a su vez pueden reunir datos carácter mezclados con elementos de tipo `autor`, `título`, `canción`, en cualquier combinación, pudiendo contener éstos últimos únicamente datos carácter.

Podemos observar que el último elemento de tipo `disco` contiene únicamente un elemento de tipo `título`. La declaración para el elemento disco no obliga a que aparezcan elementos o datos carácter e incluso podría estar vacío.

3.4 El tipo children

Se trata de elementos que sólo pueden contener otros elementos, y no datos carácter. Este tipo se utiliza fundamentalmente para agrupar otros elementos y formalizar estructuras. En la declaración del tipo se incluye un patrón que deben seguir los elementos de este tipo y que puede tomar varias formas que se explican a continuación.

Secuencia de elementos

Para especificar que el modelo se compone de una secuencia de elementos, después del nombre del tipo de elemento que se declara se añade una lista de los elementos que puede contener separados por comas:

```
<!ELEMENT nombre_de_elemento (elemento1, ... , elementoX)>
```

Alternativa de elementos

Si lo que se desea es especificar una alternativa de elementos, en lugar de comas se utiliza como separador la barra vertical “|”. En el ejemplo siguiente se especifica que únicamente un elemento de la lista puede formar parte del contenido en cada realización del tipo de elemento:

```
<!ELEMENT nombre_de_elemento (elemento1 | ... | elementoX)>
```

Combinación de modelos

Se pueden utilizar paréntesis para agrupar secuencias y alternativas de modelos:

```
<!ELEMENT nombre_de_elemento (elemento1 | ( elemento2, elemento3))>
```

Especificando una frecuencia

Es posible además especificar una frecuencia de repetición adjuntando a un elemento o a un paréntesis de cierre uno de los siguientes caracteres:

- El caracter [?] indica opción, el elemento o grupo se puede repetir **cero o una** vez
- El caracter [+] indica **una o más** repeticiones. Asegura una ocurrencia como mínimo.
- El caracter [*] indica **cero o más** repeticiones.

No es posible especificar una frecuencia determinada, 3, 5, 6 veces. Si no se especifica ninguno de estos caracteres el significado es de uno y como máximo uno.

Apliquemos esto al ejemplo del listado de discos para imponer una cierta estructura:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE listado_discos [
        <!ELEMENT listado_discos (disco)*>
        <!ELEMENT disco (datos, descripción, comentarios? )>
        <!-- Declaración del tipo datos y todos los tipos que contiene -->
        <!ELEMENT datos ((solista | grupo), título, año?)>
        <!ELEMENT solista (#PCDATA)>
        <!ELEMENT grupo (#PCDATA)>
        <!ELEMENT título (#PCDATA)>
        <!ELEMENT año (#PCDATA)>
        <!-- Declaración del tipo descripción y todos los tipos que contiene -->
        <!ELEMENT descripción (canción)+>
        <!-- Declaración del tipo de elemento canción -->
        <!ELEMENT canción (#PCDATA)>
        <!-- Declaración del tipo de elemento comentarios -->
        <!ELEMENT comentarios (#PCDATA)>
]>
<listado_discos>
  <disco>
    <datos>
      <grupo>IO</grupo>
      <título>Sensación</título>
    </datos>
    <descripción>
      <canción>Luna de Papel</canción>
      <canción>Versos</canción>
      <canción>Naufragio</canción>
    </descripción>
    <comentarios>Este disco es de los que más me gustan.</comentarios>
  </disco>
  <disco>
   <datos>
      <solista>Raúl</solista>
      <título>Tu Barco</título>
      <año>1974</año>
    </datos>
    <descripción>
      <canción>Marina</canción>
    </descripción>
  </disco>
</listado_discos>
```

En este ejemplo los documentos XML de tipo `listado_discos` se componen de elementos de tipo `disco` (cero o más). Cada elemento de tipo `disco` esta formado a su vez por un elemento de tipo `datos`, seguido de un elemento de tipo `descripción`, ambos obligatorios, seguidos de un elemento de tipo `comentarios` opcional (se puede comprobar que el último disco no tiene ningún comentario).

Los elementos de tipo `datos` se componen de tres elementos ordenados, un elemento de tipo `grupo` o de tipo `solista` (uno de los dos, pero no los dos o ninguno de ellos), un elemento de tipo `título` y un elemento de tipo `año` opcional (puede ocurrir una o cero veces).

Los elementos de tipo `descripción` pueden contener elementos de tipo `canción` (uno como mínimo). El resto de los elementos pueden contener únicamente caracteres según se regula en su declaración de tipo.

3.5 Otros ejemplos de declaraciones de elementos

<!ELEMENT aviso (parrafo)>
Indica que <aviso> sólo puede contener un solo <parrafo>.

<!ELEMENT aviso (titulo, parrafo)>
<aviso> debe contener un <titulo> seguido de un <parrafo>.

<!ELEMENT aviso (parrafo | grafico)>
<aviso> puede contener o bien un <parrafo> o bien un <grafico>. El número de opciones no está limitado a dos, y se pueden agrupar usando paréntesis.

<!ELEMENT aviso (titulo, (parrafo | grafico))>
<aviso> debe contener un <titulo> seguido de un <parrafo> o de un <grafico>.

<!ELEMENT aviso (titulo?, (parrafo+, grafico)*)>
En este caso, <aviso> puede tener <titulo>, o no (pero sólo uno), y puede tener cero o más conjuntos <parrafo><grafico>, <parrafo><parrafo><grafico>, etc.

4 Declaraciones de listas de atributos

Las Declaraciones de Listas de Atributos:

- Definen el conjunto de atributos que pertenecen a cada tipo de elemento (cada atributo está ligado siempre a un elemento).
- Establecen restricciones en el contenido de estos atributos, a través de tipos predefinidos a los que se debe ajustarse el atributo.
- Aportan información sobre la naturaleza del atributo (obligatorio, voluntario, de valor fijo) así como la posibilidad de especificar un valor por defecto para el mismo.

En un documento XML de tipo válido todos los atributos deben declararse mediante Declaraciones de Listas de Atributos, incluidas en la DTD del documento. Además, el valor de cada atributo debe estar acorde con lo declarado. Sin las DTDs se pueden asignar valores arbitrariamente a los atributos, no existe ningún control a este respecto, ni valores por defecto, ni restricciones de ningún tipo. Si el documento es pequeño esto puede no ser un problema, pero con documentos grandes, las ventajas que ofrecen las DTDs se vuelven muy valiosas. Sin una DTD tampoco se puede añadir objetos binarios al documento, ya que éstos se añaden a través de atributos.

La forma que debe tener una declaración de lista de atributos queda recogida en las reglas número [52] y [53] de la Recomendación de XML. Estas declaraciones se incluyen siempre dentro de una DTD.

```
[52]  AttlistDecl ::=  '<!ATTLIST' S Nombre AttDef* S? '>'
[53]  AttDef ::=  S Nombre S AttType S DefaultDecl
```

Todas las declaraciones de listas de atributos comienzan con la cadena <!ATTLIST, a continuación, y separados por al menos un espacio, el nombre del elemento al que pertenecen los atributos. Seguidamente el nombre del atributo que se está declarando, espacios,

el tipo del atributo y su declaración por defecto. Es posible declarar varios atributos para un elemento. Las declaraciones de atributos pueden tomar dos formas:

a) declaración de un único atributo en la lista

```
<!ATTLIST nombre_de_elemento nombre_de_atributo tipo declaración_por_defecto>
```

b) declaración de varios atributos conjuntamente

```
<!ATTLIST nombre_de_elemento nombre_de_atributo tipo declaración_por_defecto
                               ...
                             nombre_de_atributo tipo declaración_por_defecto
                               ...
>
```

Las declaraciones simples y múltiples se pueden combinar como se quiera, a gusto del diseñador de la DTD.

Para ilustrar la declaración de listas de atributos, partimos del siguiente documento ejemplo, para el que se han definido algunas declaraciones de elementos:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tienda_animales [
        <!ELEMENT tienda_animales (perro)*>
        <!ELEMENT perro (#PCDATA | nombre | raza)*>
        <!ELEMENT nombre (#PCDATA)>
        <!ELEMENT raza (#PCDATA)>
]>
<tienda_animales>
        <perro>...</perro>
        <perro>Este es un estupendo ejemplar de perro
        <raza>labrador</raza>, en la tienda le conocemos
        como <nombre>Chispa</nombre> por la viveza de sus ojos.
        Su piel es de color canela...</perro>
        <perro>...</perro>
</tienda_animales>
```

A partir de este ejemplo de partida, podemos añadir atributos como la fecha de nacimiento y la fecha de venta de un perro (no importa si en este punto no entendemos todos los detalles de las declaraciones de atributos).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tienda_animales [
       <!ELEMENT tienda_animales (perro)*>
       <!ELEMENT perro (#PCDATA)>
       <!ATTLIST perro fecha_nacimiento CDATA "">
       <!ATTLIST perro fecha_venta CDATA "">
]>
<tienda_animales>
  <perro>...</perro>
  <perro fecha_nacimiento="14/7/1999">
       Este es un estupendo ejemplar de perro labrador, en la tienda le    co-
nocemos como Chispa por la viveza de sus ojos. Su piel es de color
       canela...
  </perro>
  <perro>...</perro>
</tienda_animales>
```

Hay que tener en cuenta los siguientes aspectos en la declaración de atributos:

- Todas las declaraciones de listas de atributos comienzan con la cadena `<!ATTLIST`, tal y como está aquí escrita (sin ninguna separación entre "`<!`" y "`ATTLIST`").
- El orden de las declaraciones es como se especifica. Cualquier alteración en este sentido será causa de errores. Es decir, el elemento (nombre_de_elemento) al que se aplica la definición siempre se especifica antes que los atributos definidos, el tipo de cada atributo (tipo) se concreta después del atributo (nombre_de_atributo) y a continuación la declaración (declaración_por_defecto).
- Los valores de los atributos por defecto deben delimitarse por comillas dobles o simples.
- Hay que mantener la misma consistencia con las mayúsculas y las minúsculas. Los nombres de los atributos empleados en la declaración deben ser los mismos que luego reciban los valores en los elementos.
- El número de atributos que se declaran para un elemento puede ser tan grande como se desee, XML no impone ninguna restricción.
- El orden en que se especifican los atributos para un elemento en concreto es irrelevante y no viene impuesto por el orden definido en la declaración ni por ningún otro factor.
- Las declaraciones de listas de atributos para un tipo de elemento pueden especificarse en cualquier lugar dentro de la DTD, pero si los atributos se declaran antes que su tipo de elemento, un analizador que valide podría mostrar un aviso al usuario y continuar el proceso, aunque no es un error. También puede ocurrir que el analizador lea toda la DTD y compruebe que no falta ninguna declaración, en cuyo caso no mostraría ningún mensaje.

4.1 Tipos de atributos

El tipo en cada declaración de atributos especifica cómo puede ser el contenido del atributo limitando los valores que puede tomar. Los tipos están predefinidos y se agrupan en tres categorías:

- **Tipos cadena**: CDATA

- **Tipos enumerados**: ENUMERATION y NOTATION
- **Tipos específicos o "token"**: ENTITY, ENTITIES , ID, IDREF, IDREFS, NMTOKEN y NMTOKENS

4.1.1 Tipos cadena

Sólo existe un tipo dentro de esta categoría, el tipo CDATA, que significa "*Character DATA*", es decir, datos carácter. Es el tipo más general y el más permisivo en cuanto a los caracteres que admite. Cualquier cadena de texto (que no incluya los caracteres abajo indicados), es un valor válido para un atributo CDATA.

Los caracteres que **no se pueden incluir** (para utilizarlos habrá que escaparlos) dentro del valor de un atributo CDATA son los siguientes:

- El caracter [<], porque se interpretará como el comienzo de un nuevo elemento, el tipo CDATA no puede contener etiquetas, ni elementos.
- El caracter [&], salvo que se utilice en una referencia a una entidad.
- El caracter de comilla ['] o ["] que se utilice para delimitar el valor del atributo, ya que se confundiría con el final del atributo.

Para declarar un atributo de tipo cadena, lo único que hay que hacer es añadir la palabra CDATA a continuación de su nombre en la declaración de una lista de atributos. Por ejemplo:

```
<!ATTLIST perro fecha_nacimiento CDATA "">
```

4.1.2 Tipos enumerados

Dentro de esta categoría existen dos tipos: tipos ENUMERATION y tipos NOTATION

- **Tipo ENUMERATION**: los atributos ENUMERATION ofrecen una lista de posibles valores que puede tomar el atributo, todos estos valores son nombres de tipo *token*, que veremos más adelante.

  Para declarar un atributo de tipo ENUMERATION hay que añadir a continuación del nombre del atributo una lista con los posibles valores que puede tomar el atributo. Los valores se presentan separados entre sí por un carácter "|" y encerrados entre paréntesis:

  ```
  <!ATTLIST nom_elemento atributo (valor1 | valor2 |...) valor_por_defecto>
  ```

  Ejemplo de tipo ENUMERATION: el sexo del animal no puede tomar más que dos valores; al declarar un tipo enumerado, nos aseguramos que el valor que se le da al atributo está dentro del rango.

  ```
  <!ATTLIST perro sexo (macho | hembra) #REQUIRED>
  ```

- **Tipo NOTATION:** indica que el contenido del elemento que tiene el atributo debe ser procesado mediante alguna aplicación, especificada en una notación declarada en la DTD. Las declaraciones de notaciones las veremos más adelante.

  Para definir un atributo como de tipo NOTATION, hay que añadir la palabra "NOTATION" seguida de una lista de notaciones delimitada por paréntesis (notaciones declaradas previamente). Los nombres de las notaciones que puede tomar el atributo se deben separar por caracteres de barra vertical "|".

```
<!ATTLIST nom_elemento nom_atributo NOTATION (notación1 | notación2 |...)
          valor_por_defecto >
```

Ejemplos de tipos NOTATION: los atributos `formato_foto` y `codificación_foto` se utilizan para incluir fotos de los animales como parte del documento XML. El elemento foto podría contener una foto del animal en formato *gif* o formato jpg, codificada en *base64*. (En el ejemplo falta la declaración de las NOTATION).

```
...
<!ATTLIST foto formato_foto NOTATION (gif | jpg) "gif">
<!ATTLIST foto codificación_foto NOTATION (base64) "base64">
...
<foto formato_foto="gif" codificación_foto="base64">
   MTAAEBAQIQEBAQMTAABQ7GBoaMUF ...
</foto>
...
```

Existen ciertos detalles a tener en cuenta:

- Los valores posibles de un tipo enumerado no se especifican entre comillas
- Los caracteres de barras verticales y los paréntesis de los tipos enumerados (ENUMERATION y NOTATION) admiten a ambos de sus lados los espacios en blanco que se desee, incluyendo ninguno.

4.1.3 Atributos tipo *token*

Los atributos de tipo *token* se parecen a los atributos declarados como CDATA, con la diferencia que estos tipos son más restrictivos, ya que existen ciertos caracteres que no pueden formar parte ellos. Entre estos caracteres no permitidos destacan los siguientes: espacios en blanco, [,], [!], [;], [/] o [\].

Esta categoría incluye los siguientes tipos:

- **Tipos ID:** se utilizan para asignar un identificador único a los elementos. Deben cumplir las siguientes restricciones:
    - Los valores que pueden tomar son nombres XML (por ejemplo, no se pueden asignar valores que comiencen por un dígito).
    - El valor del atributo ID deberá ser único dentro del documento XML, es decir dos atributos ID no pueden tener el mismo valor, aunque los atributos tengan distintos nombres y se refieran a tipos de elementos diferentes.
    - Cualquier elemento puede tener un atributo de tipo ID, pero como máximo uno.
- **Tipos IDREF, IDREFS:** estos tipos de atributos se utilizan para referenciar elementos identificados con un atributo ID, es decir, el valor de éstos atributos coincide con el valor de otro atributo ID. Se deben tener en cuenta las siguientes consideraciones:
    - Al igual que ocurría con los atributos de tipo ID, los valores que pueden tomar son únicamente nombres XML.

- Cada referencia de un tipo IDREF debe estar recogida en un atributo de tipo ID, es decir, los valores que tomen estos atributos deben ser referencias a valores de atributos ID que existan en el documento.

- Los atributos IDREFS son análogos a los IDREF con la diferencia de que pueden tomar varios valores, separados por espacios.

- **Tipos ENTITY, ENTITIES:** los objetos (ficheros de imágenes, sonido, etc.) en un documento XML se gestionan a través de atributos entidad. Estas entidades son especiales e incluyen en su declaración el nombre del fichero que contiene los datos y una notación que identifica la aplicación que es capaz de interpretarlos. No se pueden utilizar entidades si no se declaran en una DTD o Esquema. Los nombres de entidad son nombres XML y deben cumplir las restricciones impuestas. Las entidades las veremos en detalle más adelante.

- **Tipos NMTOKEN, NMTOKENS:** son análogos a los de tipo CDATA, pero con la diferencia de que sus valores válidos sólo pueden ser nombres tipo *token*. Los atributos de tipo NMTOKENS son idénticos a los de tipo NMTOKEN con la diferencia de que aceptan varios valores al mismo tiempo. Los distintos valores se separan utilizando espacios. Los atributos de tipo NMTOKEN sólo puede contener letras, dígitos, punto [.], guión [-], subrayado [_] y dos puntos [:]. Los del tipo NMTOKENS pueden contener los mismos caracteres que NMTOKEN más espacios en blanco. Un espacio en blanco consiste en uno o más espacios, retornos de carro o tabuladores.

La declaración de todos estos tipos de atributos token es muy sencilla: lo único que se debe hacer es añadir la palabra, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS después del nombre del atributo.

Ejemplos:

```
...
       <!ATTLIST perro ID ID #REQUIRED>
       <!ATTLIST perro madre IDREF #IMPLIED>
       <!ATTLIST perro padre IDREF #IMPLIED>
       <!ATTLIST perro otrosnombres NMTOKENS "">
       <!ATTLIST perro fotos ENTITIES "">
...
```

Cada perro lleva un atributo `ID` que lo identifica, además se guardará la referencia ID de su `madre` y de su `padre`, siempre y cuando se tenga información de los mismos. Los nombres alternativos para un perro se almacenan en el atributo `otrosnombres`. Finalmente, el atributo `fotos` hará referencia a ficheros con fotos del animal.

4.2 Declaraciones por defecto

La Declaración por defecto es la última parte de la declaración de un atributo. Los posibles valores son:

- **#REQUIRED:** indica que en el ejemplar del documento se debe asignar obligatoriamente un valor al atributo. No hay valor por defecto.

- **#IMPLIED:** indica que es opcional incluir el atributo en el elemento. No hay valor por defecto. El analizador de XML pasará un valor en blanco a la aplicación, que puede poner su propio valor por defecto.

- **"valor por defecto":** si se especifica un valor por defecto, el atributo queda inicializado a ese valor si no se especifica ningún otro valor en el ejemplar del documento. El valor por defecto va entre comillas pueden simples o dobles.

- **#FIXED "valor del atributo":** estos atributos no cambian su valor, todos los elementos de este tipo tienen un atributo con este valor fijo. No es posible modificar el valor de un atributo FIXED en el ejemplar del documento.

Observaciones:

Los atributos ID sólo pueden tener como valor por defecto #IMPLIED y #REQUIRED; no tendría sentido un atributo ID declarado con un valor por defecto o con un valor fijo, ya que sería fácil que dos elementos acabaran con el mismo ID (lo que haría que el documento fuese de tipo no válido).

Veamos como se aplican estas declaraciones por defecto en nuestro ejemplo:

```
...
       <!ATTLIST perro fecha_nacimiento CDATA #REQUIRED>
       <!ATTLIST perro sexo (macho | hembra) #REQUIRED>
       <!ATTLIST perro ID ID #REQUIRED>
       <!ATTLIST perro madre IDREF #IMPLIED>
       <!ATTLIST perro padre IDREF #IMPLIED>
       <!ATTLIST perro otrosnombres NMTOKENS "">
       <!ATTLIST perro fotos ENTITIES "">
       <!ATTLIST perro veterinario CDATA #FIXED "Felix Marquez Sanz">
...
```

- Los atributos `fecha_nacimiento`, `sexo` e `ID` se han declarado como obligatorios, no puede haber ningún perro que no tenga definido, como mínimo, estos tres valores, se entiende que son fundamentales.

- Sin embargo, `madre` y `padre` de tipo IDREF, son de especificación opcional. Conocer cuáles fueron los padres de un perro no es una información imprescindible, puede que esta información se almacene en otro lugar, o que incluso no esté disponible.

- Los atributos `otrosnombres` y `fotos`, se han inicializado a unos valores que tendrán por defecto todas las realizaciones concretas de los mismos (un valor en blanco).

- Existe un único tipo de datos fijo, veterinario, cada animal que está en la tienda tiene asignado uno, todos los perros comparten el mismo. Si en un momento dado el veterinario cambia, debe modificarse este valor #FIXED del atributo en la DTD, y al cambiarlo se actualizarán automáticamente todas las ocurrencias del mismo en el ejemplar del documento.

Con todo esto, el ejemplo de la tienda de animales completo podría quedar así:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tienda_animales [
        <!-- Esta parte del documento no se ha estudiado todavía -->
        <!NOTATION jpg SYSTEM "programadegraficos.exe">
        <!NOTATION gif SYSTEM "programadegraficos.exe">
        <!ENTITY P250-01 SYSTEM "fotos/250-01.JPG" NDATA jpg>
        <!ENTITY P250-02 SYSTEM "fotos/P250-02.GIF" NDATA gif>
        <!ENTITY P250-03 SYSTEM "fotos/P250-03.JPG" NDATA jpg>
        <!-- Esta parte ya es familiar -->
        <!ELEMENT tienda_animales (perro)*>
        <!ELEMENT perro (#PCDATA | nombre | raza)*>
        <!ELEMENT nombre (#PCDATA)>
        <!ELEMENT raza (#PCDATA)>
        <!ATTLIST perro fecha_nacimiento CDATA #REQUIRED>
        <!ATTLIST perro sexo (macho | hembra) #REQUIRED>
        <!ATTLIST perro ID ID #REQUIRED>
        <!ATTLIST perro madre IDREF #IMPLIED>
        <!ATTLIST perro padre IDREF #IMPLIED>
        <!ATTLIST perro otrosnombres NMTOKENS #IMPLIED>
        <!ATTLIST perro fotos ENTITIES #IMPLIED>
        <!ATTLIST perro veterinario CDATA #FIXED "Felix Marquez Sanz">
]>
<tienda_animales>
        <perro fecha_nacimiento="1/4/1994" sexo="hembra" ID="P040">
          Extraordinaria hembra <raza>labrador</raza>, se trajo a la tienda
          con una herida en una pata trasera que ...
        </perro>
        ...
        <perro fecha_nacimiento="30/5/1996" sexo="macho" ID="P050">
          Macho <raza>labrador</raza> con un pelaje fantástico
          y una gran fortaleza,...
        </perro>
        ...
        <perro fecha_nacimiento="19/7/2000" sexo="macho" ID="P250"
               madre="P040" padre="P050" otrosnombres="Barullo Canela"
               fotos="P250-01 P250-02 P250-03">
          Este es un estupendo ejemplar de perro <raza>labrador</raza>, en
          la tienda le conocemos como <nombre>Chispa</nombre> por la viveza
          de sus ojos. Su piel es de color canela...
        </perro>
        ...
</tienda_animales>
```

Se puede comprobar cómo los valores que finalmente toma cada ocurrencia de los atributos concuerdan con lo declarado. Así, `sexo` toma el valor de "hembra" y "macho". `ID` toma el valor de "P040", "P050" y "P250" (no podría tomar como valor una cadena que comenzara por un dígito como "250" por las limitaciones del tipo ID). Así mismo, los valores que tomen `padre` y `madre` tienen que estar definidos en un atributo ID en el documento, como ocurre para el caso de *Chispa*, perro con la ID "P250". También podemos fijarnos en que, como mínimo, siempre están definidos los valores para los tres atributos requeridos: `fecha_nacimiento`, `sexo` e `ID`

Vemos que los atributos `otrosnombres` y `fotos` admiten varios valores, por ser de tipo NMTOKENS y ENTITIES. Todos estos valores son nombres *token* separados entre sí por espacios.

En teoría habría muchos más perros que formarían parte del documento, esto se indica mediante los puntos suspensivos colocados entre elementos de tipo perro. Si se desea validar el ejemplo anterior hay que eliminar estos puntos suspensivos, ya que de lo con-

trario un validador indicaría error, ya que estos puntos se consideran texto y los elementos `tienda_animales` sólo pueden contener elementos de tipo perro.

La parte que todavía no se ha estudiado declara las entidades (las fotos de los animales) que utiliza el documento, su localización y sus notaciones. Las notaciones deben estar declaradas e identifican la aplicación que puede gestionar estas entidades.

Por último cuando visualizamos el documento podemos comprobar cómo el procesador XML ha incluido atributos con valores por defecto dentro de cada elemento, como veterinario.

4.3 Normalización de los valores de un atributo

Antes de utilizarse, los valores de los atributos se normalizan y se comprueba si su valor concuerda con el tipo con el que se declaró. Para normalizar el valor de un atributo se siguen los siguientes pasos:

- Las referencias a carácter se sustituyen por el carácter correspondiente.
- Las entidades se sustituyen recursivamente si es necesario hasta que todas las referencias a entidades se han reemplazado.
- Los caracteres de espacio en blanco [**espacio** (Unicode/ASCII 32), **tabulador** (Unicode/ASCII 9), **retorno de carro** (Unicode/ASCII 13), y **salto de línea** (Unicode/ASCII 10)], se cambian, cada uno, por un espacio. La secuencia de caracteres retorno de carro y fin de línea se sustituyen por un único espacio.
- Si el atributo es de tipo CDATA, la normalización termina en el paso anterior; en otro caso se efectúan los siguientes pasos adicionales: se eliminan todos los espacios por delante y por detrás y cada secuencia de espacios se reemplaza por un único espacio.

Los analizadores validadores normalizan los valores de los atributos sin ningún problema, conocen todos los tipos de los atributos. Sin embargo los analizadores que no validan pueden no conocer todos los tipos, en ese caso normalizarán el valor del atributo como si fuera de tipo CDATA.

4.4 Atributos predefinidos

Existen una serie de atributos que están reservados para uso con XML. **Estos atributos solamente tienen sentido cuando se declaran en la DTD**, de otra manera su inclusión genera un error de buena formación del documento.

4.4.1 Atributo predefinido *xml:lang*

El atributo **xml:lang** se utiliza para identificar el lenguaje en el que se ha escrito el contenido de un elemento y los valores de sus atributos (si los tiene). Es un atributo de tipo NMTOKEN, que tiene que estar declarado en la DTD.

Normalmente cada valor de este atributo se divide en dos partes, separadas por un guión: una primera que identifica el lenguaje y una segunda (opcional) que identifica una variante de un país.

- Ejemplos de códigos para los lenguajes [ISO 639]: ca (catalán), de (alemán), el (griego), en (inglés), es (español), eu (vasco), fr (francés), gl (gallego), it (italiano), y pt (portugués).
- Ejemplos de códigos para los países [ISO 3166]: AR (Argentina), CA (Canadá), CU (Cuba), ES (España), FR (Francia), GB (Gran Bretaña), GR (Grecia) y MX (Méjico).

Aunque, a diferencia de otros atributos, en este caso no se distingue entre mayúsculas y minúsculas, lo habitual es escribir los códigos de lenguajes en minúsculas y los códigos para los países en mayúsculas. Así, posibles valores para el atributo xml:lang son los siguientes: **es-AR**, **es-CU**, **es-ES** y **es-MX**.

Veamos un ejemplo que podría aplicarse en una librería, en la que los títulos de los libros se guardan en atributos; su contenido puede estar en varios en varios idiomas que se especificarán mediante un atributo xml:lang:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE libros [
       <!ELEMENT libros (libro)*>
       <!ELEMENT libro EMPTY>
       <!ATTLIST libro título CDATA "">
       <!ATTLIST libro xml:lang NMTOKEN "es-CU">
]>

<libros>
 <libro xml:lang="fr" título="Tous les hommes sont mortels"/>
 <libro xml:lang="es-ES" título="Todos los hombres son mortales"/>
</libros>
```

4.4.2 Atributo predefinido *xml:space*

Los espacios en blanco pueden ser significativos o no dentro del marcado, pueden utilizarse para estructurar el texto o ser realmente una parte importante del texto. Para el procesador XML, dentro de los elementos que pueden tomar en sus valores datos carácter, los espacios en blanco son significativos y se tratan como cualquier otro carácter que forme parte de los datos carácter, no obstante, la aplicación que maneja los datos puede decidir eliminar estos espacios.

El atributo predefinido xml:space indica a la aplicación que gestiona los datos qué debe hacer con los espacios en blanco. Puede tomar los valores "default" o "preserve". El primero le indica a la aplicación que se comporte de manera predeterminada y el segundo le indica que conserve los espacios en blanco.

El atributo xml:space tiene un efecto heredado, es decir todos los hijos del elemento con este atributo preservarán los espacios en blanco, salvo que exista otro xml:space de nivel inferior en la jerarquía que determine otro modo de tratar los espacios

La forma general de declarar el atributo xml:space es la siguiente:

```
<!ATTLIST nombre_elemento xml:space (default | preserve) valor_por_defecto>
```

Veamos un ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE Poema [
        <!ELEMENT Poema (#PCDATA)>
        <!ATTLIST Poema xml:lang NMTOKEN "es-ES">
        <!ATTLIST Poema xml:space (default | preserve) "preserve">
  ]>

  <Poema xml:space="preserve">
        El Gallo (Popular)

        Gallo montero,
        gentil caballero,
        vestido de plumas
        como un coracero.

  </Poema>
```

5 Entidades

Aunque la clasificación de las entidades es un asunto que se puede abordar de varias maneras, según la *recomendación* XML se pueden distinguir dos tipos principales de entidades: las llamadas "entidades generales" y las "entidades parámetro".

Las **entidades generales** básicamente se utilizan para nombrar cadenas de texto que más tarde se utilizarán dentro del ejemplar del documento: valores de atributos, el contenido de algún elemento, elementos completos... También se utilizan para incluir datos que, en principio no son XML, como imágenes o sonidos, como parte del documento XML actual.

Las **entidades parámetro**, también se utilizan para poner un nombre a una cadena de texto, pero se utilizan exclusivamente dentro de la DTD del documento XML y contienen texto con sentido en este entorno.

Podría decirse que existe un tercer tipo de entidad, las "**entidades predefinidas**", que como ya hemos visto anteriormente, se utilizan para escapar caracteres y que, a diferencia de los dos tipos anteriores, no hay que declarar en la DTD.

Las entidades generales y las entidades parámetro tienen algunas características comunes:

- Los nombres de las entidades son nombres XML, deben cumplir sus restricciones.
- No se permiten referencias recursivas. Dentro del valor de una entidad no puede haber una referencia a sí misma.
- Ambos tipos de entidades se deben declarar en la DTD.
- Si una entidad se declara varias veces, el procesador XML toma como válida la primera e ignora el resto.
- No admiten entre sus valores los siguientes caracteres (salvo que se escapen apropiadamente):
    - El carácter [&] a menos que se utilice como comienzo de una referencia a una entidad general.
    - El carácter [%] a menos que se utilice como comienzo de referencia a una entidad parámetro.
    - El carácter de comilla [" o '] que cierra el valor literal de la entidad.

Nosotros solo vamos a hablar de las **entidades generales**. Dentro de estas, existen varios tipos:

- Entidad general interna, si se define en el mismo documento en el que se utiliza
- Entidad general externa, si se define en otro documento.
- Entidad general analizada (parsed): contiene marcado o datos carácter, información que el procesador XML entiende y que es capaz de tratar. Estas entidades se declaran y se utilizan de manera que puedan ser empleadas dentro del documento XML.
- Entidad general no analizada (unparsed): contiene información, que para el procesador XML no es XML, aunque podría. Generalmente, son ficheros en un formato no nativo de XML, imágenes, sonidos, etc. Estas entidades se deben incluir a través de atributos especiales y no mediante referencias, como ocurre con el resto de las entidades.

Combinando todo esto, se puede ver que sólo existen 3 tipos de entidades generales:

- **Entidades generales internas analizables**: se utilizan a modo de comodín. Representan una cadena de texto que se encuentra definida completamente dentro del documento XML de partida. Se utilizan a través de referencias, que el procesador sustituirá por un texto. Su contenido es texto analizable por el procesador XML y que tiene sentido en el lugar donde se enlaza la referencia.
- **Entidades generales externas analizables:** son análogas a las entidades generales internas, con la diferencia de que están definidas (total o parcialmente) fuera de la entidad documento. Se utilizan, de igual manera a través de referencias.
- E**ntidades generales externas no analizables:** este tipo es algo especial, puede contener ficheros de sonido, imágenes, documentos generados con un procesador de textos, hojas de cálculo, presentaciones, etc. Ya sabemos que en un documento XML no es posible la inclusión de ciertos caracteres, con lo que estos contenidos tienen que ser obligatoriamente externos. La utilización de este tipo implica algunas declaraciones de marcado adicionales aparte de la de la propia entidad. Su contenido se referencia a través de tipos de atributos especiales en los elementos.

Observar que las entidades generales internas no analizables no pueden darse, ya que como sabemos en un documento XML no se puede incluir directamente contenido no analizable, es decir que no sea marcado o datos carácter.

En este curso vamos a estudiar las entidades generales externas no analizables, ya que proporcionan el mecanismo para poder insertar objetos no XML dentro de los documentos.

5.1 Entidades generales externas no analizables

Las entidades generales externas no analizables se utilizan para relacionar con el documento XML contenidos que a efectos del procesador XML no serán XML. Estas entidades se relacionan con ficheros de imágenes, sonido y, en general, cualquier fichero que no sea XML.

5.1.1 Cómo se declaran

Las declaraciones de las entidades generales externas no analizables toman la forma siguiente:

```
<!ENTITY nombre_de_la_entidad SYSTEM "uri" NDATA nombre_de_notación >
```

Donde:

- nombre_de_la_entidad es el nombre de la entidad que se está declarando.
- A continuación, detrás de la cadena SYSTEM se especifica un URI al recurso que se quiere asignar a la entidad.
- La cadena NDATA indica al procesador XML que detrás viene una notación (nombre_de_notación). A través del nombre de la notación el procesador XML puede relacionar la entidad con una aplicación que será capaz de tratarla. En un documento XML de tipo válido todas las notaciones se deben declarar (lo veremos más adelante).

5.1.2 Cómo se referencian

Estas entidades únicamente se pueden referenciar a través de atributos de tipo ENTITY o ENTITIES ya estudiados, es decir, se admiten únicamente en atributos declarados especialmente a tal efecto y toman la siguiente forma:

```
<Elemento Atributo="nombre_de_entidad"> ...  </Elemento>
```

5.1.3 Componentes necesarios

Asociados a estas entidades existen pues:

- **La declaración de la entidad**: especifica dónde se localiza esta entidad y le asigna un nombre de referencia.
- **Un atributo declarado como de tipo ENTITY o ENTITIES** que **contendrá** el nombre de la entidad.
- **Una notación declarada**: a través de la notación **se** identificará a la aplicación que es capaz de procesar la entidad general externa que se quiere incluir.
- **Un elemento que recoge el atributo**: para que un atributo recoja el nombre de la entidad es necesario el elemento asociado al atributo.

Veamos todo esto en un ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE Noticia [
        <!ELEMENT Noticia (#PCDATA)>
        <!NOTATION gif SYSTEM "aplicacióngraficos.exe">
        <!NOTATION jpg SYSTEM "aplicacióngraficos.exe">
        <!ENTITY Foto_Noticia_1 SYSTEM "47autobuses.gif" NDATA gif >
        <!ENTITY Foto_Noticia_2 SYSTEM "47autobuses.jpg" NDATA jpg >
        <!ATTLIST Noticia Fotos ENTITIES "">
]>
<Noticia Fotos="Foto_Noticia_1 Foto_Noticia_2">
...En Madrid, 47 autobuses y 250 agricultores ... el hecho ocurrió el
día ...
</Noticia>
```

6 Notaciones

Las notaciones, básicamente, identifican un nombre XML con una aplicación auxiliar capaz de procesar ciertos datos y que podrá ser empleada por el procesador XML, o por la aplicación a la que sirve, para realizar alguna tarea. Las notaciones también se utilizan para realizar negociación de contenidos.

Las notaciones se declaran de una de estas dos maneras:

```
<!NOTATION nombre_de_la_notación SYSTEM "uri_aplicación">
```

o bien

```
<!NOTATION nombre_de_la_notación PUBLIC "id_pública" "uri_aplicación">
```

Donde:

- nombre_de_la_notación se refiere al nombre de la notación y debe ser un nombre XML
- "uri_aplicación" identifica un URI que localiza a la aplicación que es capaz de procesar los datos.
- En el caso de la segunda forma, "id_publica" es un identificador público para el recurso que podrá ser utilizado para localizar alternativamente a la aplicación.

Estas declaraciones se sitúan dentro de la DTD, en el subconjunto interno o externo y en un documento XML de tipo válido todas las notaciones deben declararse.

Las notaciones se utilizan básicamente para:

- Indicar a través de un atributo notación cómo interpretar el contenido de un elemento (ver el ejemplo de la página 16).
- En la declaración de entidades generales externas no analizables, relacionadas con los elementos a través de atributos ENTITY y ENTITIES (ver el ejemplo de la página 19).
- En las instrucciones de proceso, para identificar la aplicación destino de la instrucción (esto no lo veremos en este curso).

7 Referencias

[1] Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008

http://www.w3.org/TR/2008/REC-xml-20081126/

[2] Extensible Markup Language (XML) 1.1 (Second Edition).W3C Recommendation, 16 August 2006.

http://www.w3.org/TR/2006/REC-xml11-20060816/

[3] XML a través de ejemplos. Abraham Gutiérrez y Raúl Martínez. Ed. Ra-Ma.

[4] Manual imprescindible de XML. Juan Diego Gutiérrez Gallardo. Ed. Anaya Multimedia.

[5] Material del curso de formación de profesores de la Comunidad de Madrid “Desarrollo de aplicaciones WEB con PHP y XML”, impartido por EUI Universidad Politécnica de Madrid.