XML

En este capítulo se introduce el lenguaje de marcado XML. Se analizará su estructura y sintaxis para posteriormente crear descripciones de documentos mediante DTD y esquemas. Posteriormente se verá como validar documentos XML de acuerdo a su definición.

Tabla de contenido

XML

1 Introducción

XML (Lenguaje de Marcado Extensible *eXtensible Markup Language*), es un lenguaje de marcado creado por W3C para superar las limitaciones de HTML. Este lenguaje se emplea en la confección de documentos electrónicos para la web. En el mismo hay un conjunto de etiquetas que le dicen al navegador cómo tiene que presentar la información, pero sin especificar el significado de esa información. HTML se emplea en exclusiva para la presentación de una página web en un navegador. Fijémonos en el siguiente ejemplo:

```
<p><b>El ingenioso hidalgo Don Quijote de la Mancha</b>
<br>
<i>Miguel de Cervantes y Saavedra</i>
<br>
Editorial Austral
<br>
4ª edición
</p>
```

Es evidente que se trata de los datos de un libro. Cuando este código HTML le llega al navegador interpreta sus etiquetas y realiza la presentación indicada. Sin embargo, el navegador no sabe que son los datos de un libro, el intérprete HTML se limita a presentar estos datos con el formado adecuado.

Con XML, se puede asignar algún significado a las etiquetas en el documento. Más importante aún, también resulta fácil para una máquina el procesar la información. Se puede extraer el autor de un documento simplemente localizando el contenido rodeado por las etiquetas `<autor>` y `</autor>`, técnicamente conocido como el elemento `<autor>`.

```
<libro>
  <titulo>El ingenioso hidalgo Don Quijote de la Mancha</titulo>
```

```
  <autor>Miguel de Cervantes y Saavedra</autor>
  <editorial>Austral</editorial>
  <edicion>4ª</edicion>
</libro>
```

XML es una especificación de W3C como lenguaje de marcado de propósito general. Esto significa que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que debes definir tus propias etiquetas. El propósito principal del lenguaje es compartir datos a través de diferentes sistemas, como Internet.

Por tanto XML no es un lenguaje de marcado como el lenguaje HTML, aunque se parecen. XML es un metalenguaje que nos permite definir lenguajes de marcado adecuado a diferentes necesidades y usos determinados. Al ser XML un metalenguaje es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como ocurre en HTML.

Hay muchos lenguajes basados en XML; Algunos ejemplos son XHTML, MathML, SVG, XUL, XBL, RSS, y RDF. Cada desarrollador puede crear un lenguaje propio para aplicarse en un escenario concreto.

La potencia de esta estrategia radica en que estamos etiquetando e identificando el contenido, olvidándonos en un principio por la forma de presentarlo. Un documento XML también puede tener formato. Mediante CSS o XSL (un lenguaje para transformar un documento XML en otro para darle formato) podremos asignar formato de presentación a un documento XML.

1.1 Características XML

El lenguaje XML presenta las siguientes características:

- ✔ Estructurar datos ➜ XML es un conjunto de reglas o normas para describir y estructurar datos en múltiples aplicaciones, como libretas de direcciones, parámetros de configuración, transacciones financieras, etc. Facilita la generación de documentos para su tratamiento asegurando que su estructura no sea ambigua, soportando internacionalización y localización.
- ✔ Lenguaje de marcado ➜ Al igual que HTML, XML utiliza elementos similares a los utilizados en HTML como etiquetas y atributos. Mientras HTML especifica lo que cada etiqueta y atributo significa, XML usa las etiquetas sólo para organizar los datos y deja la interpretación de los mismos a las aplicaciones que los manejan. En XML no podemos asumir que `<p>` se utiliza para delimitar un párrafo, dependiendo del contexto, podría ser un precio, un parámetro, una persona, una palabra, ...
- ✔ Texto plano ➜ Igual que los de HTML, los archivos de XML son archivos de texto y aunque no están pensados para su lectura por desarrolladores y usuarios, este formato facilita mucho el desarrollo y mantenimiento de las aplicaciones.
- ✔ Sintaxis estricta ➜ Las reglas de XML son estrictas, y en esto se parece menos a

HTML. Una etiqueta olvidada o un atributo sin comillas inutilizan un archivo XML, mientras que en HTML es tolerada y a menudo explícitamente permitida. La especificación oficial de XML prohíbe a las aplicaciones que traten de adivinar las intenciones del creador de un archivo XML dañado; si el archivo está dañado, la aplicación debe detenerse allí mismo y reportar un error.

- ✔ Mejor aprovechamiento de los recursos ➔ Como XML es un formato de texto y usa etiquetas para delimitar los datos. Las ventajas de un formato de texto son evidentes y las desventajas pueden usualmente ser compensadas en un nivel diferente. El coste del espacio en disco se ha reducido considerablemente y aplicaciones de compresión y protocolos de comunicaciones pueden comprimir los datos de manera muy efectiva para el aprovechamiento del ancho de banda.
- ✔ Familia de tecnologías ➔ XML 1.0 es la especificación que define lo que son las "etiquetas" y los "atributos". Más allá de XML 1.0, "la familia XML" es un conjunto creciente de módulos que ofrecen servicios útiles para realizar tareas importantes frecuentemente demandadas: Xlink describe un modo estándar de agregar hipervínculos a un archivo XML. XPointer y Xfragments son sintaxis en desarrollo para apuntar a partes de un documento XML. XSL es el lenguaje avanzado para expresar las hojas de estilo. Se basa en XSLT, un lenguaje de transformación usado para reacomodar, agregar y eliminar etiquetas y atributos. El DOM es un conjunto estándar de llamadas a funciones para manipular archivos XML (y HTML) desde un lenguaje de programación. XML Schemas 1 y 2 ayudan a los desarrolladores a definir con precisión las estructuras de sus propios formatos basados en XML.
- ✔ Modular ➔ XML le permite definir un formato de documento combinando y reutilizando otros formatos. Puesto que dos formatos desarrollados independientemente podrían tener elementos o atributos con el mismo nombre, se debe tener cuidado al combinarlos para eliminar la confusión de nombres al combinar formatos, XML provee un mecanismo de espacio de nombres.
- ✔ XML es gratuito, independiente de la plataforma y bien soportado.

2 Documento XML

XML es un estándar, no una implementación concreta. Es un metalenguaje de marcas, lo que significa que no dispone de un conjunto fijo de etiquetas, como si sucede con HTML. Por el contrario, XML permite definir a los desarrolladores los elementos, con sus respectivas etiquetas y atributos, que necesiten y con la estructura que mejor les convenga.

Lo que XML permite hacer es definir una sintaxis general para maquetar datos con etiquetas sencillas y comprensibles a cualquier persona. Provee, asimismo, un formato estándar para documentos informáticos. Es un formato flexible, de manera que puede ser adaptado al campo de aplicación que se desee. Veamos un ejemplo:

En el campo de la química, tendría sentido la existencia de etiquetas como `<átomo>`, `<molécula>` o `<enlace>`. En el campo de la composición musical, tendría sentido que hubiera etiquetas como `<entera>`, `<negra>` o `<semicorchea>`.

De ahí viene el calificativo de extensible en el acrónimo XML, se pueden crear nuevas etiquetas en función de las necesidades o escenario de aplicación.

Puesto que es un formato de texto plano es adecuado para almacenar información y trasmitirla. Con el más simple editor de textos se puede editar un documento XML. Así mismo, los documentos XML son relativamente ligeros para ser almacenados y enviados, puesto que ocupan lo que los datos que contienen más las etiquetas que los delimitan. Suelen tener extensión `.xml`, aunque no es imprescindible,

Estas etiquetas y sus atributos son meta-información, es decir, información sobre la información, sobre los datos. Nos permiten estructurar el contenido del documento y facilitan su procesamiento, pero no son información en sí mismas.

```
<?xml version="1.0" ?>
- <pedidos>
  - <pedido cod="1">
      <fecha>01-01-2013</fecha>
      <pu>45.5</pu>
      <cantidad>2</cantidad>
      <descripcion>Botella de Vino</descripcion>
      <tipo>C</tipo>
    </pedido>
  - <pedido cod="2">
      <fecha>31-12-2012</fecha>
      <pu>25</pu>
      <cantidad>1</cantidad>
      <descripcion>Menu Ejecutivo</descripcion>
      <tipo>A</tipo>
    </pedido>
</pedidos>
```

Figura 1: Ejemplo de documento XML

2.1 Estructura de un documento XML

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

- ✔ Prólogo (opcional) ➔ Contiene una secuencia de instrucciones de procesamiento y/o declaración del tipo de documento. Se puede dividir en dos partes:
  - ➢ Declaración XML. Establece la versión de XML, el tipo de codificación y si es un documento autónomo.
  - ➢ Declaración de tipo de documento. Establece la estructura del contenido que aparece en el cuerpo.
- ✔ Cuerpo: es el contenido de información del documento, organizado como un árbol jerárquico único de elementos marcados.

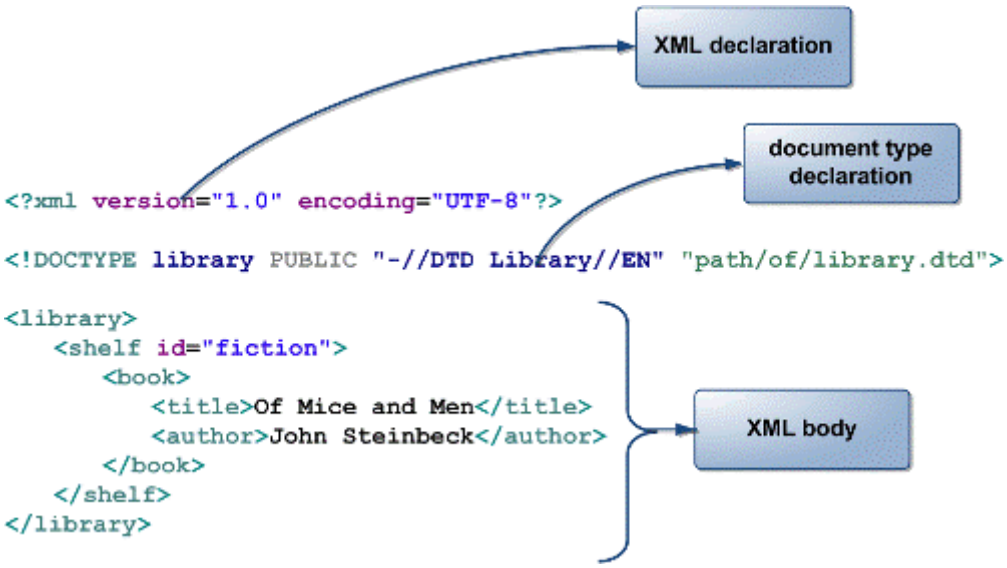

Figura 2: Estructura de documento XML

El estándar también permite la inclusión opcional de un epílogo, al final del documento, que puede contener instrucciones de procesamiento. Esta parte en general se omite dada su poca utilidad, ya que resulta poco intuitivo poner las instrucciones de procesamiento al final.

Las instrucciones de procesamiento se utilizan para enviar información a las aplicaciones que van a procesar el documento XML. Las instrucciones de procesamiento pueden aparecer en varios lugares del documento, por ejemplo entre el prólogo y el cuerpo, dentro del cuerpo o en el epílogo.

2.2 Estructura jerárquica del contenido

En un documento XML la información se organiza de forma jerárquica, de manera que los elementos del documento se relacionan entre sí mediante relaciones de padres, hijos, hermanos, ascendentes, descendentes, etc.

A esta estructura jerárquica se la denomina árbol del documento XML. A las partes del árbol que tienen hijos se las denomina nodos intermedios o ramas, mientras que a las que no tienen se conocen como nodos finales u hojas.

Por ejemplo, para representar la información de una persona con un nombre y unos apellidos en XML se podría representar así:

```
<persona>
    <nombre>María</nombre>
    <apellido>González</apellido>
</persona>
```

Su representación en árbol jerárquico sería la siguiente:

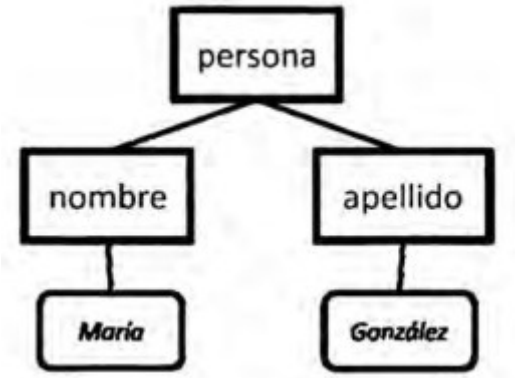

Figura 3: Árbol jerárquico XML

Como se puede apreciar en el ejemplo cada nodo representa una etiqueta o los datos que encierra una etiqueta. Cuando una etiqueta está contenida dentro de otra, el nodo del árbol cuelga del nodo de la etiqueta a la que pertenece. En este ejemplo:

- ✔ El nodo `persona` es la raíz del documento
- ✔ El nodo `persona` tiene dos hijos: `nombre` y `apellido`.
- ✔ Los nodos `María` y `González` son nodos hoja ya que no tienen descendientes. El primero es hijo de `nombre` y el segundo de `apellido`. Generalmente los nodos hoja son el contenido textual de un elemento.

Fijémonos en el siguiente ejemplo ahora.

```
<libro>
    <capitulo>En un lugar de <negrita>La Mancha</negrita> de
cuyo nombre …
    </capitulo>
</libro>
```

En este ejemplo el elemento `capitulo` tiene tres hijos:

- ✔ El nodo hoja (contenido textual) `En un lugar de`
- ✔ El nodo `negrita`
- ✔ El nodo hoja `de cuyo nombre…`

2.3 Modelo de datos

Como se ha visto en epígrafes anteriores, un documento XML consta de una estructura jerárquica en forma de árbol invertido formada por los siguientes tipos de componentes o nodos.

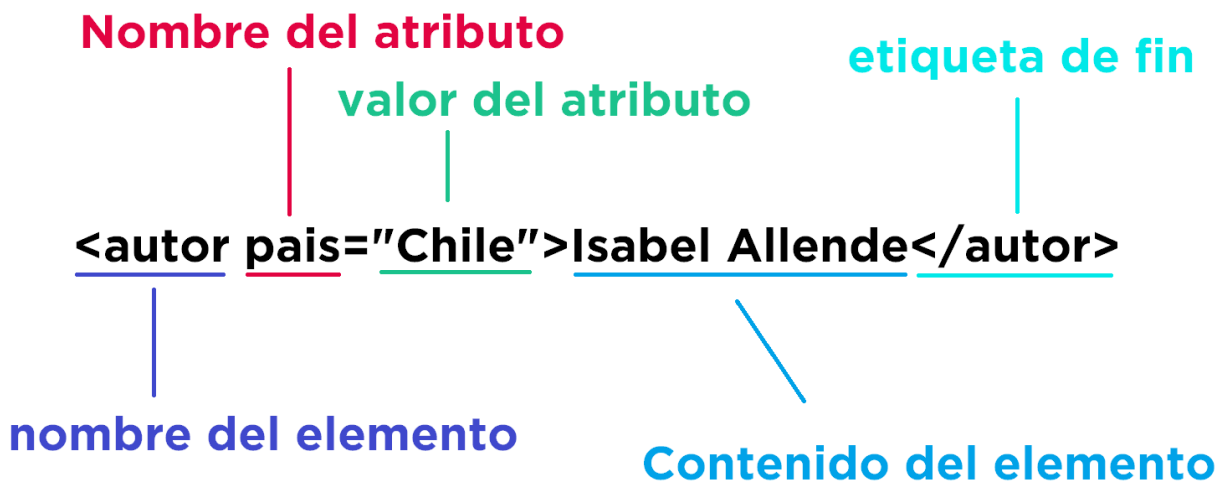

Figura 4: Elemento XML

2.3.1 Raíz

Por encima de cualquier elemento se ubica el nodo raíz, que se designa como `/`. No es un componente que tenga representación dentro del documento XML, pero se utilizará más adelante como punto de partida para recorrer el árbol XML y ubicar el resto de nodos.

2.3.2 Elementos

Es la unidad básica de un documento XML. Actúan como contenedores de la información además de delimitarla. Estos nodos básicamente crean la estructura de la información. Se identifican por una etiqueta de apertura, como `<persona>` y una de cierre, como `</persona>`. Lo que se ubica entre ambas es el contenido de ese elemento, que puede ser textual, otros elementos o vacío. Algunos tipos de elementos especiales son:

- ✔ Elemento raíz ➜ Todo documento XML bien formado debe contener un único elemento raíz que contiene a todos los demás (no tiene ascendentes ni hermanos). También se le llama elemento documento.
- ✔ Elementos sin contenido ➜ Son elementos vacíos, es decir, no encierran otros elementos ni contenido textual. Aunque puede tener atributos, se abre y se cierra con una sola etiqueta. Por ejemplo `<separador/>`
- ✔ Elemento sin contenido pero con atributos ➜ Igual que el anterior, pero el elemento dispone de atributos. Por ejemplo `<separador cantidad="7"/>`

2.3.3 Atributos

Son similares a los atributos de HTML. Son pares nombre-valor que permiten especificar características o propiedades de los elementos en forma de datos adicionales. Se ubican en la etiqueta de apertura del elemento. Para asignar un valor a un atributo se utiliza el signo igual. Todos los atributos, independientemente del tipo de datos que representen, se tratarán como texto y aparecerán entre comillas simples o dobles. Por

ejemplo

```
<distancia unidades="km">70</distancia>
```

También se emplean para recoger información identificativa del elemento que permita distinguirlo de otro elemento. Por ejemplo

```
<persona nif="12345678Z2">...</persona>
```

2.3.4 Texto

Representa los datos del documento XML. Mientras que los anteriores sirven para definir la estructura de los datos, estos siempre aparecen como contenido de los elementos o como valor de atributo. No puede aparecer en ningún otro lugar

Un tratamiento especial tienen los espacios en blanco. Hay cuatro tipos de caracteres de espaciado en XML

- ✔ Tabulador \t 	
- ✔ Nueva línea: \n

- ✔ Retorno de carro \r
- ✔ Espacio \s

Dentro del contenido textual de un elemento se mantendrán como están y así serán tratados por el procesador. Como valor de un atributo, los espacios en blanco adyacentes se condensarán en uno solo. Los espacios en blanco entre elementos serán ignorados. Veamos los siguientes ejemplos:

- ✔ `<nombre> José </nombre>` no es igual que `<nombre>José</nombre>` , es decir, los espacios incluidos dentro del contenido textual de un elemento se tienen en cuenta.
- ✔ `<producto ref="FA 001">Caja galletas</producto>` es igual que `<producto ref="FA001">Caja galletas</producto>` lo que significa que los espacios adicionales en el valor de un atributo no se tienen en cuenta.
- ✔ `<pedido numero="240"> <producto ref="FA001">Caja galletas</producto>` es igual que `<pedido numero="240"><producto ref="FA001">Caja galletas</producto>` ya que los espacios entre elementos se ignoran.

2.3.5 Comentarios

Son iguales que los de HTML. Empiezan por los caracteres `<!--` y se cierran con los caracteres `-->`. Dentro de ellos se puede escribir cualquier carácter sin necesidad de caracteres de escape, excepto el doble guión (`--`) que confundiría al analizador con un posible cierre del comentario.

Pueden ubicarse en cualquier lugar de un documento, menos dentro de una etiqueta de apertura, ni dentro de una etiqueta de cierre.

2.3.6 Instrucciones de procesamiento:

Las instrucciones de procesamiento empiezan por `<?` y terminan por `?>`. Son instrucciones para el procesador XML. Por tanto el contenido depende de él. No forman parte del contenido del documento XML. Se utilizan para dar información a las aplicaciones que procesan el documento XML. Por ejemplo la primera línea de cada documento XML es una instrucción de procesamiento para declarar el tipo de documento: `<?xml versión="1.0" encoding="utf-8"?>`

2.3.7 Entidades predefinidas

La especificación XML utiliza cinco entidades predefinidas representando caracteres especiales, y requiere que todos los procesadores de XML los utilicen. Son similares a los caracteres referenciados en HTML. Son las siguientes

Nombre	Carácter	Punto de código Unicode	Descripción
quot	"	U+0022	comillas dobles
amp	&	U+0026	símbolo & o ampersand
apos	'	U+0027	apóstrofe
lt	<	U+003C	signo menor que
gt	>	U+003E	signo mayor que

Por ejemplo, `<titulo>Ricky &apm; Morty</titulo>`. El contenido será interpretado como `Ricky & Morty`.

2.4 Restricciones sintácticas

Un documento XML se dice que está bien formado si cumple todas las reglas sintácticas establecidas por el W3C en la especificación XML. Existen muchas reglas, pero las más significativas son las siguientes:

- ✔ El documento puede (el W3C lo recomienda) empezar por una instrucción de procesamiento `xml`, que indica la versión del XML y, opcionalmente, la codificación de caracteres (`encoding`), que por defecto es UTF-8, y si está listo para procesarse independientemente o requiere de otros archivos externos para dicha tarea (`standalone`), que por defecto es `no`. La instrucción de procesamiento correcta más simplificada es:

```
<?xml version = "1.0"?>
```

Otra más extensa e igualmente correcta:

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

- ✔ Debe existir un único elemento raíz, que cuelga del nodo raíz (/). Este elemento tendrá como descendientes a todos los demás elementos. El documento XML bien

formado más simple contendría sólo un elemento raíz, sin ningún descendiente.

- ✔ Los elementos que no sean vacíos deben tener una etiqueta de apertura y otra de cierre.
- ✔ Los elementos vacíos deber cerrarse con `/>`. Por ejemplo: `<br />`
- ✔ Los elementos deben aparecer correctamente anidados en cuanto a su apertura y su cierre, y no solaparse. Es decir, los elementos se cierran en orden inverso al que se abren. Por ejemplo: se abre `<alfa>`, se abre `<beta>` y se abre `<gamma>`. Se cerrarán en orden inverso al de apertura. Primero se cierra `</gamma>`, luego `</beta>` y, por último, `</alfa>`.

```
<alfa> <beta> <gamma>...</gamma> </beta> </alfa>
```

Sin embargo, si se hubiera cerrado `</beta>` antes que `</gamma>` no sería correcto.

```
<alfa> <beta> <gamma>...</beta> </gamma> </alfa>
```

- ✔ Los nombres de elementos y atributos son sensibles a mayúsculas/minúsculas.
- ✔ Los valores de los atributos deben aparecer entre comillas simples o dobles, pero del mismo tipo. Por ejemplo `<libro signatura="SIL001"/>` es correcto y `<libro signatura='SIL001"/>` no lo es al utilizar diferente tipo de comillas en la apertura y el cierre.
- ✔ No puede haber dos atributos con el mismo nombre asociados al mismo elemento.
- ✔ No se pueden introducir ni instrucciones de procesamiento ni comentarios en ningún lugar del interior de las etiquetas de apertura y cierre de los elementos. Por ejemplo, un comentario incorrecto sería `<persona <!-- comentario incorrecto --> >...</persona>`.
- ✔ No puede haber nada antes de la instrucción de procesamiento `<?xml ... ?>`.
- ✔ No puede haber texto antes ni después del elemento documento.
- ✔ No pueden aparecer los signos `<` ni `&` en el contenido textual de elementos ni atributos.

2.5 Visualización de un documento XML

En principio los documentos XML no incluyen marcas para la presentación, por lo que de forma nativa no tienen formato. Si se visualiza un documento XML en un navegador se visualizará tal cual, sin aplicar ningún formato.

La información contenida en un documento XML no está enfocada al usuario, sino que se emplea sobre todo en el intercambio de información entre sistemas. Existen varias maneras para representar visualmente los datos de un documento XML:

- ✔ Mediante una hoja de estilo CSS que indique al navegador cómo convertir cada

elemento del documento XML en un elemento visual.

- ✔ Mediante el uso de una hoja de transformaciones XSLT.
- ✔ Mediante el uso de un lenguaje de programación, como Java o JavaScript, que procese el documento XML.

Veremos los dos primeros métodos en un capítulo posterior.

3 Validación XML

Cuando se pretende validar los documentos XML es necesario haber definido previamente la estructura de su contenidos, es decir, que elementos va a tener, cuál será el contenido de los elementos y los atributos. Esta definición de la estructura puede hacerse con una definición de tipo de documento (DTD) o con un esquema. Vamos a ver ambos en detalle.

3.1 Definición de tipo de documento

Un DTD (*Document Type Definition*) es una descripción de la estructura de un documento XML. En esta descripción se tiene que especificar qué elementos tienen que aparecer, en qué orden, cuáles son optativos, cuáles obligatorios, qué atributos tienen los elementos, etc.

Es un mecanismo de validación de documentos que existía antes de la aparición de XML. Se usaba para validar documentos SGML, que es el precursor de todos los lenguajes de marcas actuales. Cuando apareció XML, se integró en su especificación como modelo de validación gramatical.

El DTD puede aparecer de diferentes formas:

- ✔ Interno ➜ Integrado en el propio documento XML.
- ✔ Externo ➜ En un archivo independiente con extensión `.dtd`. La opción del documento externo permitirá reutilizar el mismo DTD para distintos documentos XML, facilitando las posibles modificaciones de una manera centralizada. Cada documento XML que emplee el DTD tiene que hacer referencia a él para su validación.
- ✔ Mixto ➜ Puede tener una parte interna y otra externa. Es decir, se pueden definir una parte de la estructura dentro del documento DTD y hacer referencia al archivo `.dtd` que contiene otra parte de la estructura.

Siempre que se quiera declarar un DTD, se hará al comienzo del documento XML, justo después de la primera instrucción de procesamiento recomendada por W3C y que consiste en la declaración XML. Las reglas que lo constituyen son las que podrán aparecer a continuación de la declaración (dentro del propio documento XML) o en un archivo independiente.

3.1.1 Declaración del DTD

La declaración del DTD en un documento XML se realiza con `<!DOCTYPE>`. Es la instrucción donde se indica qué DTD se empleará para valida el XML y donde se define la estructura de sus elementos. Aparece al comienzo del documento XML. El primer dato que aparece es el nombre del elemento raíz del documento XML.

En función del tipo de DTD la sintaxis varía. Las características que definen el tipo son:

- ✔ Ubicación ➜ Dónde se localizan las reglas del DTD.
  - ➢ Interno ➜ Las reglas aparecen en el propio documento XML.
  - ➢ Externo ➜ Las reglas aparecen en un archivo independiente.
  - ➢ Mixto ➜ Combinación de los dos anteriores. Las reglas aparecen en ambos lugares. Las reglas internas tienen prioridad sobre las externas.
- ✔ Carácter ➜ Si es un DTD para uso privado o público.
  - ➢ Para uso privado ➜ Se identifica por la palabra `SYSTEM`.
  - ➢ Para uso público ➜ Se identifica por la palabra `PUBLIC`. Debe ir acompañado del FPI (*Formal Public Identifier*), una etiqueta que identifica al DTD de manera "universal".

Las distintas combinaciones son:

Sintaxis	**Tipo de DTD**
`<!DOCTYPE elemento_raíz [reglas]>`	Interno (implícitamente privado)
`<!DOCTYPE elemento_raíz SYSTEM URL>`	Externo y privado
`<!DOCTYPE elemento_raíz SYSTEM URL [reglas]>`	Mixto y privado
`<!DOCTYPE elemento_raíz PUBLIC FPI URL>`	Externo y público
`<!DOCTYPE elemento_raíz PUBLIC FPI URL [reglas]>`	Mixto y público

No puede haber una declaración interna y pública ya que no tiene sentido. Si un DTD es público no quema más remedio que sea independiente.

Parámetros

`elemento_raíz`

Nombre del elemento raíz del documento XML: aparece justo a continuación de la palabra DOCTYPE.

`SYSTEM | PUBLIC`

Declaración de privacidad/publicidad. Aparece a continuación del nombre del elemento raíz e indica si el DTD es de uso público (PUBLIC) o de uso interno de la organización que lo desarrolla (SYSTEM).

`FPI`

Sólo existe cuando el DTD es PUBLIC e indica el identificador por el que se conoce el DTD. Da información sobre la propia DTD y la organización que la ha creado.

`URL`

Indica la dirección web (URL) que referencia al archivo DTD. Solo existe cuando el DTD está en un archivo externo y la URL da su ubicación.

`reglas`

Consiste en las reglas que componen la estructura del documento XML.

FPI es una cadena de texto con un formato específico que se emplea para identificar unívocamente un producto, especificación o documento. Uno de sus usos más comunes es ser parte de una definición de tipo de documento, pero también son usados en otros formatos. Tiene la siguiente sintaxis:

```
{+|-}//Organización//Clase Descripción//Código de idioma
```

Parámetros

`+|-`

El primer carácter indica si la organización está registrada con ISO. El carácter – indica que la organización no está registrada por ISO. Si la organización está registrada se indica con +.

`Organización`

Organización responsable del DTD

`Clase`

Indica el tipo de documento que se está declarando. En el caso de una definición de tipo de documento sería DTD.

`Descripción`

Consiste en la descripción del lenguaje cuya estructura se define en el DTD.

`Código de idioma`

Es el código internacional del idioma en que esté escrita la DTD.

Por ejemplo, en los documentos XHTML el FPI es el siguiente:

```
"-//W3C//DTD HTML 4.01//EN"
```

Donde

- ✔ `-` ➜ La organización (W3C) no está registrada en ISO.
- ✔ `W3C` ➜ Es la organización que publica el documento.
- ✔ `DTD` ➜ Es la clase de documento.

- ✔ `XHTML 1.1` ➜ Es la descripción del lenguaje. La descripción consta a su vez de otras dos partes, la etiqueta que es `XHTML`, y la versión, que es `1.1`.
- ✔ `EN` ➜ El documento se ha publicado en inglés.

Veamos un ejemplo de declaración de DTD. Supongamos un documento XML cuyo elemento raíz se llama `<pedidos>`. Se trata de un DTD de uso privado (`SYSTEM`) y la ubicación de las reglas están en un archivo externo llamado `pedidos.dtd` que se encuentra en el mismo directorio que el archivo XML. Su declaración en el archivo XML sería la siguiente:

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE pedidos SYSTEM "pedidos.dtd">
…
```

Más adelante veremos como crear el archivo `pedidos.dtd`, el cual contiene las reglas con la estructura del documento XML. En este archivo no se incluye la declaración del tipo de documento (`<!DOCTYPE …`), puesto que sólo aparece en la cabecera del documento XML.

Conviene resaltar también que el atributo `standalone` de la instrucción de procesamiento `<?xml ... ?>` tiene el valor `no`, lo que significa que para el correcto procesado del documento necesitamos del uso de otros documentos externos (en este caso, de `pedidos.dtd`).

3.1.2 Componentes del DTD

Hay cuatro componentes que pueden formar un DTD:

- ✔ Elemento
- ✔ Atributo
- ✔ Entidad
- ✔ Notación

3.1.2.1 Elemento

Es una declaración de tipo de elemento. Indica la existencia de un elemento en el documento XML. Su sintaxis es:

```
<!ELEMENT nombre_elemento contenido>
```

Parámetros

`nombre_elemento`

El nombre del elemento en el documento XML. No se pondrán los delimitadores de su correspondiente etiqueta en el documento XML, es decir, < y >.

`contenido`

El contenido de un elemento puede indicar:

- Una regla, en cuyo caso será: `* ANY`. Se puede utilizar al construir el DTD para dejar la descripción de un elemento como válida en cualquier caso, eliminando cualquier comprobación sintáctica. Es un comodín que no debe aparecer en el DTD definitivo.
- `EMPTY` → Describe un elemento vacío.
- `#PCDATA` → Son datos en forma de cadena de caracteres (*Parsed Character DATA*) que puede ser texto, números, o cualquier otro formato pero sin contener etiquetas. Debe aparecer entre paréntesis.
- Elementos descendientes → El elemento contiene otros elementos. Se incluye entre paréntesis.
- Mixto → El elemento puede contener texto y otros elementos. En este caso el elemento #PCDATA estaría dentro del conjunto del resto de elementos. Se incluye entre paréntesis

Cuando el contenido del elemento son los elementos descendientes o mixto podemos especificar también en que orden aparecen y cuantas veces. Para ello utilizamos las siguientes reglas:

✔ Cardinalidad de elementos ➔ Indica el número de veces que puede aparecer un elemento o una secuencia de elementos. Para ello se emplean los siguientes símbolos que van a continuación del elemento sin espacios.

Símbolo	Significado
`?`	El elemento (o secuencia de elementos) puede aparecer 0 o 1 vez.
`*`	El elemento (o secuencia de elementos) puede aparecer 0 o n veces.
`+`	El elemento (o secuencia de elementos) puede aparecer 1 o n veces. No se puede utilizar con contenido mixto.

✔ Secuencias de elementos ➔ En la secuencia de elementos se emplean símbolos para indicar el orden en que deben aparecer, o si debe aparecer uno de ellos en exclusiva.

Símbolo	Significado
`A,B`	El elemento A aparece antes que B
`A|B`	Puede aparecer el elemento A o el B, pero no ambos.

Se pueden combinar el uso de símbolos de cardinalidad con la secuencia de elementos.

Veamos un ejemplo sencillo en el que solamente hay elementos contenidos. En un correo electrónico, se podría describir el elemento raíz `<email>` como una secuencia de elementos `<para>`, `<cc>` (optativo), `<cco>` (optativo), `<asunto>` y `<cuerpo>`. Las reglas del DTD serían las siguientes:

```
<!ELEMENT email (para, cc?, cco?, asunto, cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT cc (#PCDATA)>
```

```
<!ELEMENT cco (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
```

Otro ejemplo en el que incluye contenido mixto. Se quiere describir un elemento `<párrafo>` que simule el párrafo de un procesador de textos, de forma que pueda contener texto sin formato, texto en `<negrita>` o texto en `<cursiva>`. Estas dos últimas etiquetas podrán a su vez contener, bien texto sin formato, bien la otra etiqueta. Por tanto, `<párrafo>`, `<negrita>` y `<cursiva>` son elementos de contenido mixto. Las reglas que describen esto serían:

```
<!ELEMENT parrafo (negrita | cursiva | #PCDATA)*>
<!ELEMENT negrita (cursiva | #PCDATA)*>
<!ELEMENT cursiva (negrita | #PCDATA)*>
```

No se puede usar el cuantificador `+` con un contenido mixto, por eso se ha usado el `*`. Un documento XML válido con respecto a las reglas del anterior DTD:

```
<parrafo>
  Aquí un <cursiva>tema <negrita>importante</negrita></cursiva>
</parrafo>
```

3.1.2.2 Atributo

Cuando los elementos incluidos en la DTD, especificados según el epígrafe anterior, incluyen atributos, tenemos que añadir reglas para indicar la existencia de atributos de un elemento en el documento XML. La sintaxis es:

```
<! ATTLIST nombre_elemento
         nombre_atributo tipo atributo carácter por_defecto
         nombre_atributo tipo_atributo carácter por_defecto
… >
```

Parámetros

`nombre_elemento`

El nombre del elemento en el documento XML. Tiene que estar previamente definido con `<!ELEMENT …>`.

`nombre_atributo`

El nombre del atributo tiene que ser un nombre XML válido.

`tipo_atributo`

Indicamos el tipo de valor que tendrá el atributo. Puede ser uno de los siguientes:

- `CDATA` → Caracteres que no contienen etiquetas.
- `ENTITY` → El nombre de una entidad que también debe declararse en el DTD.
- `ENTITIES` → Una lista de nombres de entidades, separadas por espacios

- Enumerado → Una lista de valores separados por | y encerrados entre paréntesis, de entre los cuales el atributo debe tomar uno. Ver un ejemplo posterior.
- `ID` → Un identificador único. Se usa para identificar elementos, es decir, caracterizarlos de manera única. Por ello, dos elementos no pueden tener el mismo valor en atributos de tipo ID. Además, un elemento puede tener a lo sumo un atributo de tipo ID. El valor asignado a un atributo de este tipo debe ser un nombre XML válido.
- `IDREF` → Representa el valor de un atributo ID de otro elemento, es decir, para que sea válido, debe existir otro elemento en el documento XML que tenga un atributo de tipo ID y cuyo valor sea el mismo que el del atributo de tipo IDREF del primer elemento.
- `IDREFS` → Representa múltiples IDs de otros elementos, separados por espacios.
- `NMTOKEN` → Cualquier nombre sin espacios en blanco en su interior. Si hubiera espacios en blanco anteriores o posteriores, se ignorarán.
- `NMTOKENS` → Una lista de nombres, sin espacios en blanco en su interior separados por espacios. Si hubiera espacios en blanco anteriores o posteriores se ignorarán.
- `NOTATION` → Un nombre de notación, que debe estar declarada en el DTD.

`carácter`

El carácter del atributo puede ser:

- Un valor textual entre comillas, que representa un valor por defecto para el atributo.
- `#IMPLIED`, el atributo es de carácter opcional y no se le asigna ningún valor por defecto.
- `#REQUIRED`, el atributo es de carácter obligatorio, pero no se le asigna un valor por defecto.
- `#FIXED`, el atributo es de carácter obligatorio y se le asigna un valor por defecto que además es el único valor que puede tener el atributo.

`por_defecto`

El valor por defecto del atributo. Si se omite el atributo en el elemento tendrá este valor.

Lo habitual es utilizar un solo `ATTLIST` para declarar todos los atributos de un elemento, aunque se podría usar uno para cada atributo.

Cuando queremos que el atributo tome un valor de entre una lista usaremos un tipo de atributo enumerado. Por ejemplo, se quiere definir una regla que valide la existencia de un elemento `<semaforo>`, de contenido vacío, con un único atributo `color` cuyos posibles valores sean `rojo`, `naranja` y `verde`. El valor por defecto será `verde`.

```
<!ELEMENT semaforo EMPTY>
<!ATTLIST semaforo color (rojo | naranja | verde) "verde">
```

Veamos otro ejemplo. Se quiere representar un elemento `<empleado>` que tenga dos atributos, `idEmpleado` e `idJefe`. El primero será de tipo ID y carácter obligatorio, y el segundo de tipo `IDREF` y carácter optativo.

```
<!ELEMENT empleado (nombre, apellido) >
<!ATTLIST empleado
```

```
    idEmpleado ID #REQUIRED
    idJefe IDREF #IMPLIED>
```

Un fragmento de un documento XML válido con respecto a las reglas de este ejemplo podría ser el siguiente.

```
<empleados>
 <empleado idEmpleado="e_111">...</empleado>
 <empleado idEmpleado="e_222" idJefe="e_111"></empleado>
 …
</empleados>
```

Cada elemento `<empleado>` tiene un atributo `idEmpleado`, con un valor válido (nombre XML válido) y el segundo elemento `<empleado>` tiene también un atributo `idJefe`, cuyo valor ha de ser el mismo que el del atributo de tipo `ID` de otro elemento existente en el documento. En este caso, este atributo `idJefe` del segundo `<empleado>` vale igual que el atributo `idEmpleado` del primer `<empleado>`.

Por otro lado, un fragmento de XML no válido con respecto a las reglas anteriores sería aquél en el cual el atributo `idJefe` del primer `<empleado>` tenga un valor que no exista para ningún atributo de tipo ID de otro elemento del documento.

```
<empleados>
 <empleado idEmpleado="e_111" idJefe="e_333">...</empleado>
 <empleado idEmpleado="e_222" idJefe="e_111">...</empleado>
 …
</empleados>
```

En el siguiente ejemplo se quiere declarar un atributo de tipo `NMTOKEN` de carácter obligatorio. En el siguiente DTD, la declaración del elemento `<rio>` y su atributo `pais` es:

```
<!ELEMENT rio (nombre) >
<!ATTLIST rio pais NMTOKEN #REQUIRED>
```

En el siguiente fragmento XML, el valor del atributo `pais` es válido con respecto a la regla anterior.

```
<rio pais="EEUU">
  <nombre>Misisipi</nombre>
</rio>
```

En el siguiente documento XML, el valor del atributo `pais` no es válido con respecto a la misma regla ya que contiene espacios en su interior.

```
<rio pais="Estados Unidos">
  <nombre>Misisipi</nombre>
</rio>
```

3.1.2.3 Entidad

Una entidad es un elemento que puede emplearse como contenido textual de un elemento. Hay diferentes tipos de entidades, cada uno con su sintaxis propia. Son los siguientes:

- ✔ Referencia a entidades generales (internas o externas).
- ✔ Referencia a entidades parámetro (internas o externas).
- ✔ Entidades no procesadas (unparsed).

En el caso de la referencia a entidades generales internas, se utilizarán dentro del documento XML. Su sintaxis es:

```
<!ENTITY nombre_entidad definición_entidad>
```

Parámetros

`nombre_entidad`

Aquí indicamos el nombre de la entidad que se declara.

`definición_entidad`

Texto con el valor de la entidad. Va entre comillas dobles.

Veamos un ejemplo, en primer lugar, se declara una entidad en el DTD.

```
<!ENTITY rsa "República Sudafricana">
```

A continuación, se usa en el XML anteponiendo al nombre de la entidad el carácter ampersand (`&`) y a continuación un carácter punto y coma (`;`). El programa analizador del documento realizará la sustitución.

```
<país>
    <nombre>&rsa;</nombre>
</país>
```

Las referencia de tipo entidades generales externas están ubicadas en otros archivos. Su sintaxis es:

```
<!ENTITY nombre_entidad tipo_uso url_archivo>
```

Parámetros

`nombre_entidad`

Aquí indicamos el nombre de la entidad que se declara.

`tipo_uso`

Puede ser `SYSTEM` (privado) o `PUBLIC` (público).

`url_archivo`

Indica la URL con el archivo donde se encuentra definida la entidad.

Por ejemplo, se dispone de un archivo de texto, `autores.txt`, que contiene el siguiente texto plano **"`Miguel de Cervantes y William Shakespeare`"**.

Se crea un documento XML que hace referencia a ese archivo de texto, en forma de entidad externa. Al visualizar el documento, la referencia a la entidad general externa se sustituirá por el texto contenido en el archivo.

```
<?xml version="1.0"?>
<!DOCTYPE escritores [
    <!ELEMENT escritores (*PCDATA) >
    <!ENTITY autores SYSTEM "autores.txt"> ]>
<escritores>&autores;</escritores>
```

A continuación vienen las referencias a entidades parámetro, las cuales no pueden usarse en el el documento XML, solo en el propio DTD. Se pueden utilizar para agrupar ciertos elementos del DTD que se repitan mucho. Se diferencian las entidades parámetro de las generales, en que para hacer referencia a ellas, se usa el símbolo `%` en lugar de `&` tanto como para declararlas como para usarlas. Su sintaxis es:

```
<!ENTITY % nombre_entidad definición_entidad>
```

Parámetros

`nombre_entidad`

Aquí indicamos el nombre de la entidad que se declara.

`definición_entidad`

Se define el valor que posteriormente será sustituido donde aparezca.

Por ejemplo, se declara una entidad parámetro `dimensiones` y se referencia dentro del propio DTD.

```
<!ENTITY % dimensiones "alto CDATA #IMPLIED ancho CDATA #IMPLIED
profundo CDATA #IMPLIED">
<¡ELEMENT objeto (nombre) >
<!ATTLIST objeto codigo ID #REQUIRED %dimensiones;>
```

Este código es equivalente a haber escrito:

```
<!ELEMENT objeto (nombre)>
<!ATTLIST objeto
    codigo ID #REQUIRED
    alto CDATA #IMPLIED
    ancho CDATA #IMPLIED
    profundo CDATA #IMPLIED>
```

Las entidades de tipo referencia a entidades parámetro externas están ubicadas en otros archivos. Su sintaxis es:

```
<!ENTITY % nombre_entidad tipo_uso fpi url_archivo>
```

Parámetros

`nombre_entidad`

Aquí indicamos el nombre de la entidad que se declara.

`tipo_uso`

Puede ser `SYSTEM` (privado) o `PUBLIC` (público).

`fpi`

Solo en el caso de que el tipo de uso sea `PUBLIC`. Es el identificador público de la entidad. Su sintaxis es similar a la vista anteriormente para un DOCTYPE.

`url_archivo`

Indica la URL con el archivo donde se encuentra definida la entidad.

Las entidades no procesadas, referencian a datos que no deben ser procesados por el analizador XML, sino por la aplicación que lo use. Su sintaxis es:

```
<!ENTITY % nombre_entidad tipo_uso fpi valor_entidad NDATA tipo>
```

Parámetros

`nombre_entidad`

Aquí indicamos el nombre de la entidad que se declara.

`tipo_uso`

Puede ser `SYSTEM` (privado) o `PUBLIC` (público).

`fpi`

Solo en el caso de que el tipo de uso sea `PUBLIC`. Es el identificador público de la entidad.

`valor_entidad`

Indica la URL con el archivo donde se encuentra definida la entidad.

Por ejemplo, vamos a declarar una notación de nombre `JPG` para el tipo MIME. Se declara una entidad no procesada de nombre `mediterraneo`, asociada al archivo de imagen `mediterraneo.jpg`. Por último, se declara un elemento `<mar>`, que cuenta con un atributo `imagen` que es del tipo `ENTITY` recién declarado.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg" NDATA JPG>
```

```
<!ELEMENT mar (nombre)>
<!ATTLIST mar imagen ENTITY #IMPLIED>
```

En el XML, el valor del atributo `imagen` del elemento `<mar>` es la entidad no procesada declarada en el DTD:

```
<mares>
    <mar imagen="mediterraneo">
         <nombre>Mediterráneo</nombre>
    </mar>
    …
</mares>
```

Podemos extender el ejemplo anterior permitiendo incluir múltiples imágenes como valor del atributo `imagen` del elemento `<mar>`.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo1 SYSTEM "mediterraneo1.jpg" NDATA JPG>
<!ENTITY mediterraneo2 SYSTEM "mediterraneo2.jpg" NDATA JPG>
<!ELEMENT mar (nombre)>
<!ATTLIST mar imagen ENTITIES #IMPLIED>
```

Y en el documento XML, el valor del atributo imagen del elemento `<mar>` son las dos entidades no procesadas declaradas en el DTD:

```
<mares>
    <mar imagen="mediterraneo1 mediterraneo2">
         <nombre>Mediterráneo</nombre>
    </mar>
    …
</mares>
```

3.1.2.4 Notación

Este es un elemento avanzado en el diseño de DTDs. Es una declaración del tipo de atributo `NOTATION`. Una notación se usa para especificar un formato de datos que no sea XML. Se usa con frecuencia para describir tipos MIME, como `image/gif` o `image/jpg`.

Se utiliza para indicar un tipo de atributo al que se le permite usar un valor que haya sido declarado como notación en el DTD. Su sintaxis es:

```
<!NOTATION nombre_notación SYSTEM "identificador_externo”>
```

Parámetros

`nombre_notacion`

Aquí indicamos el nombre de la notación que se declara.

`SYSTEM`

Para uso privado

“identificador_externo”

Solo en el caso de que el tipo de uso sea PUBLIC. Es el identificador público de la entidad.

La sintaxis del atributo que usa la notación sería:

```
<!ATTLIST nombre_elemento nombre_atributo NOTATION
valor_defecto>
```

Por ejemplo, se declaran tres notaciones que corresponden a otros tantos tipos MIME de imágenes (gif, jpg y png). También se declara un elemento <mar> y sus atributos imagen y formato_imagen, este último referenciando las notaciones recién creadas. Por último, se declara una entidad no procesada que se asocia a un archivo de imagen.

```
<!NOTATION GIF SYSTEM "image/gif ">
<!NOTATION JPG SYSTEM "image/jpeg" >
<!NOTATION PNG SYSTEM "image/png">
<!ELEMENT mar (nombre) >
<!ATTLIST mar
    imagen ENTITY #IMPLIED
    formato_imagen NOTATION (GIF | JPG | PNG) #IMPLIED>
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg">
```

Y en el documento XML, el valor del atributo imagen del elemento <mar> es la entidad no procesada declarada en el DTD, y el valor del atributo formato _imagen el de una de sus alternativas válidas, la notación JPG:

```
<mares>
    <mar imagen="mediterraneo” formato_imagen="JPG">
    <nombre>Mediterráneo</nombre>
</mar>
    …
</mares>
```

3.1.3 Secciones condicionales

Permiten incluir o excluir reglas en un DTD en función del resultado de una condición. Sólo se pueden ubicar en DTDs externos. Su uso tiene sentido al combinarlas con referencias a entidades parámetro.

Las secciones condicionales son IGNORE e INCLUDE, teniendo la primera precedencia sobre la segunda. Por ejemplo, supónganse dos estructuras diferentes para un mensaje:

- ✔ Una sencilla que incluye emisor, receptor y contenido.
- ✔ Otra extendida que incluye los datos anteriores junto con el título y el número de palabras.

Se quiere diseñar un DTD que, en función del valor de una entidad parámetro, incluya una estructura de mensaje o la otra. Por defecto se incluirá la larga. El DTD, que ha de ser

externo, tendrá la siguiente estructura:

```
<!-- Mensaje corto -->
<![IGNORE[
    <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo -->
<![INCLUDE [
    <!ELEMENT mensaje (titulo, emisor, receptor, contenido,
palabras)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT palabras (#PCDATA)>
]]>

<!-- Declaración de elementos y atributos comunes →
<!ELEMENT emisor (*PCDATA)>
<!ELEMENT receptor (+PCDATA)>
<!ELEMENT contenido (*PCDATA>
```

Al añadir las referencias a entidad parámetro, el DTD quedaría:

```
<!ENTITY %corto "IGNORE">
<!ENTITY %largo "INCLUDE">

<!-- Mensaje corto →
<![%corto[
    <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo →
<![%largo[
    <!ELEMENT mensaje (titulo, emisor, receptor, contenido,
palabras)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT palabras (#PCDATA)>
]]>

<!-- Declaración de elementos y atributos comunes →
<!ELEMENT emisor (#PCDATA)>
<!ELEMENT receptor (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>
```

Para cambiar el tipo de mensaje que se va a incluir, basta con modificar la asociación de valores de `%corto` a `INCLUDE` y de `%largo` a `IGNORE`, de la forma:

```
<!ENTITY %corto "INCLUDE">
<!ENTITY %largo "IGNORE”>
```

Se podrían dejar la declaración de las entidades en la DTD interna al documento XML, donde se determinaría qué tipo de mensaje se quiere incluir de manera específica para ese documento. Sería así

```
<?xml verion="1.0"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd" [
    <!ENTITY %corto "INCLUDE">
    <!ENTITY %largo "IGNORE">
]>
<mensaje>
…
</mensaje>
```

3.1.4 Inconvenientes de los DTD

Algunas limitaciones de los DTD son:

- ✔ Un DTD no es un documento XML, luego no se puede verificar si está bien formado.
- ✔ No se pueden fijar restricciones sobre los valores de elementos y atributos, como su tipo de datos, su tamaño, etc.
- ✔ No soporta espacios de nombres.
- ✔ Sólo se puede enumerar los valores de atributos, no de elementos.
- ✔ Sólo se puede dar un valor por defecto para atributos, no para elementos.
- ✔ Existe un control limitado sobre las cardinalidades de los elementos, es decir, no se puede concretar el número de veces que pueden aparecer.

3.2 Esquema

Un esquema XML (XML Schema) es un conjunto de reglas utilizadas para la definición de la estructura del contenido de un documento XML y su posterior validación. Los esquemas se guardan en archivos con extensión `.xsd` y su propuesta está basado en las especificaciones de W3C.

Los esquemas como forma de definir la estructura del contenido de un documento XML han surgido recientemente y están pensados para reemplazar a las DTD. Estas resultan demasiado limitadas ya que no permiten definir tipos de datos y los únicos elementos terminales son los datos textuales. Los esquemas XML permiten el uso de varios tipos para sus datos.

Con frecuencia, los documentos XML están destinados a ser insertados o extraídos de una base de datos. Conocer los tipos de datos que se manipulan permite llevar a cabo de manera más eficaz determinados procesos (como las inserciones o las extracciones). En el mejor de los casos, los tipos de datos de los documentos XML se corresponden con los tipos utilizados por la base de datos. En este sentido, los esquemas son una herramienta esencial para la integración de la solución XML. Además, la inclusión de la especificación del tipo en la gramática no es su única ventaja: los esquemas son más fáciles de leer y entender porque están escritos en lenguaje XML.

Un esquema XML añade flexibilidad y potencia a la hora de definir la estructura de los datos. Presentan varias ventajas sobre los DTDs:

- ✔ Los esquemas usan sintaxis XML ➔ En otras palabras, un esquema XML es un documento XML, al contrario de los DTD. Esto significa que se puede procesar un esquema igual que cualquier otro documento XML y podría ser validado.
- ✔ Los esquemas XML soportan tipos de datos ➔ Los esquemas XML soportan todos los tipos originales de los DTDs (IDREF, ID...). Además, soportan tipos de datos enteros, números en punto flotante, fechas, horas, cadenas de texto, URLs y otros tipos de datos útiles para el procesado y validación de datos.
- ✔ Los esquemas XML son extensibles ➔ Además de los tipos de datos definidos en la especificación de esquemas XML, se pueden crear tipos de datos propios y se pueden derivar nuevos tipos de datos a partir de otros.
- ✔ Los esquemas XML soportan espacios de nombres ➔ A diferencia del DTD, el XSD soporta la definición de los espacios de nombres o NAMESPACES, que discriminan los distintos elementos por un prefijo, y que previenen errores de coincidencia en nombres de elementos XML.
- ✔ Los esquemas XML tienen mayor poder de expresión ➔ Por ejemplo, con esquemas XML se puede definir que el valor del un atributo `<estado>` no puede tener una longitud mayor de 2 caracteres, o que el valor de un elemento `<codigo-postal>` debe cumplir la expresión regular `[0-9]{5}(-[0-9]{4})?`. No se puede hacer ninguna de estas cosas con los DTDs.
- ✔ Permiten concretar con precisión la cardinalidad de un elemento, es decir, las veces que puede aparecer en un documento XML.

Un documento DTD describe la jerarquía de los elementos que componen el documento XML. Las hojas del árbol jerárquico representan los elementos terminales, aquellos que contienen la información. Las DTD atribuyen necesariamente el tipo `#PCDATA` a los valores de estos elementos. A los atributos no se les puede especificar ningún tipo de datos. En un esquema, los elementos terminales y los atributos de los elementos tienen asignado un tipo de datos. Podremos utilizar los tipos predefinidos por la gramática o crear unos nuevos de acuerdo con las necesidades del escenario de aplicación.

Otro inconveniente de las DTD es que se construyen con un lenguaje propio que es necesario entender. La lectura de una gramática compleja descrita por un documento DTD es a veces difícil. Los esquemas especifican las gramáticas de los documentos XML en lenguaje XML. El documento resultante es mucho más fácil de leer y a menudo más comprensible.

3.2.1 Estructura de un esquema

El esquema es un documento XML, por lo tanto tiene un prólogo para definir la versión del lenguaje XML y el juego de caracteres utilizado. Su estructura general es:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<!-- Definición del esquema →

</xsd:schema>
```

El elemento `schema` hace referencia al espacio de nombres XMLSchema publicado por el W3C en `http://www.w3.org/2001/XMLSchema`. El uso de prefijo de espacio de nombres es opcional, aunque generalmente se usa `xs` o `xsd`. A continuación se incluyen las etiquetas para definir los elementos, atributos y, en caso necesario, la definición de tipos de datos personalizados.

3.2.2 Definición de elementos

La definición de los elementos es diferente si este contiene otros elementos o si es un elemento final que solo contiene datos. En el primer caso la sintaxis es la siguiente:

```
<xsd:element name="nombre_elemento"
    minOccurs="minimo" maxOccurs="máximo">
 <xsd:complexType>
  <xsd:sequence>

    Definición de elementos simples

  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Posee el tipo `complexType` como cualquier elemento que incluya atributos u otros elementos. El tipo `simpleType` está reservado a los elementos que no poseen ni atributos, ni otros elementos.

Las listas de subelementos se describen en el interior de un elemento de secuencia. Si alguno de estos subelementos contuviera atributos u otros elementos, entonces habría que volver a incluir una definición de elemento como la anterior.

Con `minOccurs` y `maxOccurs` se indican cuantos elementos como mínimo y como máximo puede contener. Si `maxOccurs` tiene el valor `unbounded` entonces el número de elementos máximos es indeterminado, se pueden incluir tantos como se desee. El valor por defecto de ambos es de 1.

En el caso de elementos que no contienen atributos u otros elementos se elegirá un elemento de tipo simple con la siguiente sintaxis.

```
<xsd:element name="nombre_elemento" type="tipo_datos"
             minOccurs="minimo" maxOccurs="máximo"
             default="valor"
             fixed="valor"/>
```

El atributo `type` debe tener un valor con alguno de los tipos de datos predefinidos del estándar o un nombre de atributo personalizado. Los tipos predefinidos más habituales son los siguientes:

- ✔ `xsd:string` ➜ Cadena de caracteres.
- ✔ `xsd:date` ➜ Fecha en formato `YYYY-MM-DD`.
- ✔ `xsd:dateTime` ➜ Fecha y hora en formato `YYYYMM-DDThh:mm:ss`.
- ✔ `xsd:boolean` ➜ Un valor booleano `true` o `false`.
- ✔ `xsd:decimal` ➜ Número decimal de precisión arbitraria.
- ✔ `xsd:double` ➜ Número en punto flotante de doble precisión.
- ✔ `xsd:float` ➜ Número en punto flotante de simple precisión.

Hay algunos más, pero estos son los más habituales. De cada uno de estos tipos primitivos se pueden obtener tipos derivados.

Con `default` podemos establecer un valor por defecto. Si el elemento se incluye en el documento XML sin ningún contenido equivaldría a haber puesto como contenido el valor indicado en el atributo `default`.

En el caso de que el elemento tiene un valor fijo se indicaría con el atributo `fixed`.

3.2.3 Definición de atributos

La declaración de atributos se incluyen en los elementos de tipo `complexType` y se hace usando la siguiente sintaxis:

```
<xsd:attribute name="nombre_atributo" type="tipo_datos"
               default=”valor”
               fixed=”valor”/>
```

El tipo de datos es alguno de los tipos de datos vistos en el epígrafe anterior.

3.2.4 Métodos de diseño

Hay varias maneras de abordar el diseño de un esquema XML. Usaremos una u otra, o una combinación de varias, dependiendo de factores tales como la complejidad, extensión y el tipo de documentos que estamos definiendo (por ejemplo, sin son documentos donde predomina una colección de datos estructurados, o son documentos con mucho texto libre).

A modo de ejemplo vamos a crear el esquema del siguiente documento XML.

```
<revista codigo="0954651235">
  <nombre>
    El XML libre
  </nombre>
  <fechadesalida>18-06-2002</fechadesalida>
  <artículo>
    <título>Los esquemas XML</título>
    <autor> Sébastien L. </autor>
    <númerodepalabras>500</númerodepalabras>
    <texto>
```

```
      Contenido del artículo
    </texto>
  </artículo>
  <artículo>
    <título>XML y DTD</título>
    <autor> Yasmine S.  </autor>
    <autor> Lourdin H. </autor>
    <númerodepalabras>260</númerodepalabras>
    <texto>
      Contenido del artículo
    </texto>
  </artículo>
</revista>
```

3.2.4.1 Diseño anidado o "muñecas rusas"

Se llama así porque se anidan declaraciones de elementos unas dentro de otras. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Consiste en, partiendo de un documento XML, seguir la estructura del mismo e ir definiendo los elementos que aparecen en el mismo de forma secuencial, incluyendo la definición completa de cada elemento en el mismo orden en el que aparecen en el documento instancia. Este método de diseño es muy sencillo, pero puede dar lugar a esquemas XML difíciles de leer y mantener cuando los documentos son complejos. Además produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones. Es el método de diseño menos recomendable.

Empezamos con el prólogo

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

A continuación para la etiqueta de apertura del elemento `revista` se crea un elemento con el mismo nombre. Al tener atributos y subelementos tiene que ser de tipo `complexType`.

Las listas de subelementos se describen en el interior de un elemento `sequence`. Por lo tanto, después de la declaración del esquema puede haber lo siguiente:

```
<xsd:element name="revista">
 <xsd:complexType>
  <xsd:sequence>
```

El elemento `sequence` (llamado compositor `sequence`) define una lista ordenada de subelementos.

Después, los elementos `nombre` y `fechadesalida` que no tienen atributos, ni subelementos, se describen como tipos simples. Para el primer elemento, se ha elegido el tipo `xsd:string`. El segundo elemento tendrá el tipo `xsd:date`. Estos son tipos predefinidos por el espacio de nombres XMLSchema del W3C:

```
<xsd:element name="nombre" type="xsd:string"/>
<xsd:element name="fechadesalida" type="xsd:date"/>
```

A continuación, el elemento `articulo`, el cual es complejo. En este caso como puede haber varios elementos artículo vamos a incluir cardinalidad.

```
<xsd:element name="articulo" minOccurs="0"
maxOccurs="unbounded">
 <xsd:complexType>
   <xsd:sequence>
```

El atributo `minOccurs` representa el número mínimo de ocurrencias y `maxOccurs` el número máximo. `unbounded` significa que el elemento puede repetirse tantas veces como se desee. Si se omiten estos atributos tienen un valor de 1 por defecto.

Ahora viene la declaración de los componentes del elemento artículo:

```
<xsd:element name="título" type="xsd:string"/>
<xsd:element name="autor" type="xsd:string"  minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element name="númerodepalabras" type="xsd:string"/>
<xsd:element name="texto" type="xsd:string"/>
```

La descripción del elemento artículo finaliza con el cierre de las etiquetas de los elementos `sequence`, `complexType` y `element`:

```
</xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Finalmente, se cierra la secuencia del elemento `revista`.

```
</xsd:sequence>
```

Una vez indicados los elementos del elemento `revista` podemos declarar su atributo `código`.

```
<xsd:attribute name="código" type="xsd:string"/>
```

A continuación se finaliza el esquema, aquí se cierra el elemento `revista`:

```
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Este primer estilo de esquema e similar a la estructura del documento de datos XML, cada elemento o atributo del documento se describe en el mismo orden en el que aparecen. A continuación, se presenta el esquema completo correspondiente al documento XML de ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="revista">
  <xsd:complexType>
    <xsd:sequence>

      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="fechadesalida" type="xsd:date"/>

      <xsd:element name="artículo"
          minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="título" type="xsd:string"/>
          <xsd:element name="autor" type="xsd:string"
              minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element
               name="númerodepalabras" type="xsd:string"/>
          <xsd:element name="texto" type="xsd:string"/>

        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  </xsd:sequence>

  <xsd:attribute name="código" type="xsd:string"/>
 </xsd:complexType>
</xsd:element>

</xsd:schema>
```

3.2.4.2 Diseño por clonación

Aunque el diseño anidado es muy fácil de implementar, puede convertirse rápidamente en algo difícil de leer o de mantener si la estructura del documento se vuelve más compleja. Se aleja mucho de la organización de las DTD, ya que se complican las operaciones de conversión manual en el esquema.

El estilo de esquema por clonación incorpora los conceptos que se han abordado en la construcción anterior. Sin embargo, la manera de construir el esquema es diferente. Al igual que las DTD, consiste en un simple catálogo de elementos presentes en el documento y especifica, para cada uno de ellos, la lista de atributos y elementos. Los elementos terminales y los atributos se declaran en primer lugar. A continuación, aparecen las declaraciones de secuencias y de tipos complejos y se hace referencia a los elementos y atributos anteriormente declarados. El uso de referencias es similar a la clonación de objetos, de ahí el nombre de este tipo de construcción. El elemento o atributo se define y se duplica en el esquema mediante el uso del atributo `ref`.

La construcción de este tipo de esquema es sencilla. A continuación, se presenta el esquema completo correspondiente al documento XML presentado anteriormente siguiendo

una construcción por clonación.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<!-- definición de los elementos simples -->

<xsd:element name="nombre" type="xsd:string"/>
<xsd:element name="fechadesalida" type="xsd:date"/>
<xsd:element name="título" type="xsd:string"/>
<xsd:element name="autor" type="xsd:string"/>
<xsd:element name="númerodepalabras" type="xsd:string"/>
<xsd:element name="texto" type="xsd:string"/>

<!-- definición de los atributos -->

<xsd:attribute name="código" type="xsd:string"/>

<!-- definición de los elementos de tipo complejo -->

<xsd:element name="artículo">
  <xsd:complexType>
    <xsd:sequence>
      <!-- Se hace referencia al elemento simple mediante el
atributo "ref" -->
      <xsd:element ref=título/>
      <xsd:element ref="autor"
          minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="númerodepalabras"/>
      <xsd:element ref="texto"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="revista">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="nombre"/>
      <xsd:element ref="fechadesalida"/>
      <xsd:element ref="artículo"
          minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
   <xsd:attribute ref="código"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

3.2.5 Definición de tipos personalizados

Los esquemas ofrecen la posibilidad de definir tipos de datos personalizados. La creación de tipos es el tercer método de construcción de esquemas, después de los métodos en muñecas rusas y por clonación. Un tipo puede definir datos simples o complejos. Los tipos permitirán dividir su esquema en partes atómicas y enlazarlas entre ellas.

Los tipos de datos simples son los elementos `simpleType` y los tipos de datos complejos los elementos `complexType`. Estas definiciones deben hacerse fuera de cualquier definición de elemento o atributo.

A menudo, la creación de un tipo nuevo consiste en derivar un tipo a partir de otro, estableciendo alguna restricción a los valores aceptados. Por ejemplo, para definir un tipo de datos correspondiente a una cadena de caracteres que acepta un máximo de 32 caracteres, se crea el siguiente tipo:

```
<xsd:simpleType name="nombreTipo">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="32"/>
  </xsd:restriction>
</xsd:simpleType>
```

El elemento `simpleType` indica que el tipo creado es simple. El elemento `restriction` permite establecer una restricción sobre un tipo predefinido especificado en el atributo `base`. En este ejemplo es `string`. El elemento `maxLength` se denomina una faceta. Permite especificar una restricción en el tamaño de la unidad correspondiente al tipo derivado.

La faceta `pattern` en ocasiones resulta muy útil. Permite introducir una expresión regular para definir el dominio de aceptación de un valor. Por ejemplo, en el documento XML de la revista, el primer atributo empleado es un código. Vamos a definir el tipo de este código utilizando esta faceta de la siguiente manera:

```
<xsd:simpleType name="codigoTipo">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{10}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Por lo tanto, el código se define como una cadena de diez caracteres numéricos.

Los tipos complejos se definen de la misma forma. A continuación, se presenta el esquema XML completo para este ejemplo utilizando los tipos personalizados:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<!-- definición de los tipos simples -->
<xsd:simpleType name="nombreTipo">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="32"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="fechaTipo">
  <xsd:restriction base="xsd:date"/>
</xsd:simpleType>
```

```
<xsd:simpleType name="textoTipo">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="códigoTipo">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{10}"/>
  </xsd:restriction>
</xsd:simpleType>

<-- definición de los tipos complejos -->

<xsd:complexType name="artículoTipo">
  <xsd:sequence>
    <xsd:element name="título" type="nombreTipo"/>
    <xsd:element name="autor" type="nombreTipo"
        minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="númerodepalabras" type="nombreTipo"/>
    <xsd:element name="texto" type="textoTipo"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="revistaTipo">
  <xsd:sequence>
    <xsd:element name="nombre" type="nombreTipo"/>
    <xsd:element name="fechadesalida" type="fechaTipo"/>
    <xsd:element name="artículo" type="artículoTipo"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="código" type="códigoTipo" use="required"/
>
</xsd:complexType>

<!-- Creación o instanciación del elemento revista -->
<xsd:element name="revista" type="revistaTipo"/>

</xsd:schema>
```

3.2.6 Vinculación del esquema al documento XML

Una vez tenemos definido el esquema tenemos que vincularlo al documento XML para que éste pueda validarse. Para ello, el documento XML debe hacer referencia a un espacio de nombres el cuál sería `http://www.w3.org/2000/10/XMLSchema-instance`. Por lo general, se asocia con el prefijo `xsi`.

Existen dos tipos de vínculos que se corresponden con dos sintaxis diferentes. En primer lugar, tomemos un esquema en el que no se define el espacio de nombres. En este caso, el vínculo para el ejemplo anterior se establece utilizando la siguiente sintaxis:

```
<?xml version="1.0" encoding="utf-8" ?>
<revista código="0836217462"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:noNamespaceSchemaLocation="prensa.xsd">

    …

</revista>
```

Vemos que el archivo en el que hemos guardado el esquema es `prensa.xsd`.

Sin embargo, para vincular un documento a un esquema en el que se define un espacio de nombres, se debe utilizar la siguiente sintaxis:

```
<revista código="0836217462"
xmlns="http://www.ejemploesquema.com/
espaciodenombres/revistas/"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.ejemploesquema.com/espaciodenombr
es/revistas prensa.xsd">
```

4 Espacio de nombres

Al poder reutilizar las definiciones de documentos o esquemas en múltiples documentos XML nos podemos encontrar la necesidad de utilizar varios de estos vocabularios y en aparezcan en cada uno de ellos dos elementos con significado diferente pero con el mismo nombre.

Un ejemplo sería un documento XML que contuviera referencias a un cliente y a un producto solicitado por éste. Tanto el elemento que representa el cliente como el que representa el producto pueden tener un elemento hijo llamado `numero_ID`. Las referencias al elemento `numero_ID` podrían ser ambiguas ya que son elementos con igual nombre pero significado distinto.

Por tanto, estos documentos XML que contendrán elementos de múltiples fuentes independientes entre sí, pueden tener problemas de reconocimiento y colisión. Puede ser que estemos utilizando dos vocabularios que utilizan el mismo tipo de elemento o nombre de atributo.

Un espacio de nombres XML es una recomendación W3C para proporcionar elementos y atributos con nombre único en un documento XML. Un documento XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML. Si a cada uno de estos vocabularios se le da un espacio de nombres, se resuelve la ambigüedad existente entre elementos o atributos que se llamen igual. Los nombres de elementos dentro de un espacio de nombres deben ser únicos.

Un espacio de nombres XML no necesita que su vocabulario sea definido, aunque es una buena práctica utilizar un DTD o un esquema XML para definir la estructura de datos en la ubicación URI del espacio de nombres.

4.1 Declaración de un de espacio de nombres

Para definir un espacio de nombres al que pertenece un elemento se emplea, dentro de la etiqueta de inicio de ese elemento, el atributo XML reservado `xmlns`, cuyo valor debe ser un identificador uniforme de recurso, que es el nombre del espacio de nombres. El formato genérico de una declaración de un espacio de nombres es el siguiente:

```
xmlns:<prefijo>=<URI>
```

El `prefijo`, que es opcional, es un alias que identifica al espacio de nombres. El prefijo de un espacio de nombres es totalmente arbitrario; lo que define e identifica un espacio de nombres es, en realidad, el URI. Hay que destacar que el URI no se interpreta realmente como una dirección; se trata como una cadena de texto por el analizador XML. Aquí no se trata de utilizar un URI como enlace, ni tiene por qué tener contenido, los URI sólo se utilizan para que el nombre sea único. Tampoco es una referencia a la definición de la estructura del contenido del documento XML. Siguiendo con el ejemplo anterior podríamos tener el siguiente documento XML en el que se definen dos espacios de nombres en el elemento `cliente`.

```
<?xml version="1.0" encoding="utf-8"?>
<cli:cliente xmlns:cli='http://www.ejemplo.com/cliente'
             xmlns:ped='http://www.ejemplo.com/pedido'>
    <cli:numero_ID>1234</cli:numero_ID>
    <cli:nombre>José Pérez</cli:nombre>
    <cli:telefono>957000000</cli:telefono>
    <ped:pedido>
      <ped:numero_ID>56789</ped:numero_ID>
      <ped:articulo>Caja de herramientas</ped:articulo>
      <ped:precio>187,90</ped:precio>
    </ped:pedido>
</cli:cliente>
```

El prefijo se emplea para cualificar los elementos

El prefijo se emplea para cualificar los elementos que pertenecen al espacio de nombres. Si no colocamos el prefijo, entonces el nombre del espacio de nombres es indicado por el nombre del elemento que tiene el atributo `xmlns`.

El alcance de la declaración de un prefijo de espacio de nombres comprende desde la etiqueta de inicio de un elemento XML, en la que se declara, hasta la etiqueta final de dicho elemento XML. En las etiquetas vacías, correspondientes a elementos sin hijos, el alcance es la propia etiqueta.

4.2 Espacio de nombres por defecto

Cuando la declaración del espacio de nombres no especifica el carácter dos puntos y el prefijo del espacio de nombres, se está definiendo un espacio de nombres por defecto. Un espacio de nombres por defecto se aplica al elemento donde está declarado (si ese elemento no posee prefijo de espacio de nombres), y a todos los elementos sin prefijo dentro del contenido del ese elemento.

Si el atributo `xmlns` es una cadena vacía, entonces se considera que los elementos sin prefijo pertenecientes al ámbito de la declaración no están en ningún espacio de nombres. Por ejemplo,

```
<?xml version="1.0" encoding="utf-8"?>
<cliente xmlns='http://www.ejemplo.com/cliente'
         xmlns:ped='http://www.ejemplo.com/pedido'>
    <numero_ID>1234</cli:numero_ID>
    <nombre>José Pérez</cli:nombre>
    <telefono>957000000</cli:telefono>
    <ped:pedido>
      <ped:numero_ID>56789</ped:numero_ID>
      <ped:articulo>Caja de herramientas</ped:articulo>
      <ped:precio>187,90</ped:precio>
    </ped:pedido>
</cliente>
```

Vemos que el elemento `cliente` tiene un espacio de nombres por defecto ya que el primer atributo `xmlns` no contiene un prefijo. Todos los elementos dentro de cliente que no tienen prefijo (`numero_ID`, `nombre`, `telefono`) no tienen prefijo y pertenecen al espacio de nombres de `cliente`. El elemento `pedido` y sus elementos hijos tienen prefijo y pertenecen a otro espacio de nombres.

4.3 Unicidad de los atributos

Ningún elemento XML puede tener dos atributos con el mismo nombre expandido. El siguiente caso violaría esta restricción, ya que los dos atributos del elemento tendrían el mismo nombre expandido.

```
<?xml version="1.0"?>
<raiz xmlns:ns1="http://www.ejemplo1.com/ns"
      xmlns:ns2="http://www.ejemplo2.com/ns">
  <elemento ns1:atributo="valor1"  ns2:atributo="valor2" />
</raiz>
```

4.4 Espacio de nombres en DTD

Los espacios de nombre nacieron con posterioridad a las DTDs y por eso estas no dan soporte a los espacios de nombre. Si combinamos varios documentos XML con sus correspondientes DTD y queremos mezclar ambas páginas, habrá elementos con el mismo nombre. La única solución es que el autor de cada documento especifique qué DTD usar cuando se valide un elemento dado.

Es decir, en el elemento se especifica qué DTD usar, por lo que esto sólo tiene sentido si se usa más de un DTD. Entonces se usarían ambos DTD y utilizaríamos los espacios de nombre para distinguir aquellos elementos en los que no esté claro a qué DTD pertenecen. De esta forma, hay que definir los espacios de nombre usados para poderlos validar con DTDs.

Siguiendo con nuestro ejemplo ser haría de la siguiente manera. Supongamos el

siguiente DTD en el archivo `cliente.xsd`.

```
<!DOCTYPE cli:cliente [
<!ELEMENT cli:cliente
          (cli:numero_ID cli:nombre cli:telefono ped:pedido)>
<!ELEMENT ped:pedido (ped:numero_ID, ped:articulo, ped:precio) >
<!ELEMENT cli:numero_ID (#PCDATA)>
<!ELEMENT cli:nombre (#PCDATA)>
<!ELEMENT cli:telefono (#PCDATA)>
<!ELEMENT ped:pedido (#PCDATA)>
<!ELEMENT ped:articulo (#PCDATA)>
<!ELEMENT ped:precio (#PCDATA)>
<!ATTLIST cli:cliente
    xmlns:cli CDATA #FIXED "http://www.ejemplo.com/cliente"
    xmlns:ped CDATA #FIXED "http://www.ejemplo.com/pedido">

]>
```

Vemos que los elementos definidos contienen su prefijo y en la lista de atributos se han incluido atributos con sus respectivos prefijos `xmlns` los cuales tienen el valor fijo de la URI de cada espacio de nombres.

4.5 Espacios de nombres en esquemas

El uso de espacios de nombres en los esquemas emplea una solución más elegante que en las DTDs. Consiste en introducir en el elemento raíz del esquema una referencia al espacio de nombres mediante el atributo `xmlns`.

Además, podemos hacer que el esquema que estamos creando tenga asociado un espacio de nombres propio. Para ello se puede utilizar el atributo `targetNamespace` del elemento `schema` que crea un espacio de nombres al que pertenecen los elementos que se definen en el esquema.

También se puede especificar si los elementos y los atributos declarados en él deben estar certificados por un espacio de nombres, ya sea explícitamente mediante un prefijo o implícitamente de forma predeterminada, cuando se utilicen en un documento XML. Para ello se pueden utilizar los atributos `elementFormDefault` y `attributeFormDefault` del elemento `<xsd:schema>`. Los posibles valores de estos atributos son:

- ✔ `qualified` ➜ En los documentos XML que referencien este esquema, los elementos (en el caso de `elementFormDefault`) o atributos (en el caso de `attributeFormDefault`) deben estar cualificados con un prefijo.
- ✔ `unqualified` ➜ Este es el valor por defecto. Indica que los elementos/atributos no necesitan estar prefijados en el documento XML.

Sigamos con nuestro ejemplo. Supongamos que el documento XML con el pedido de un cliente tuviera el el siguiente esquema definido en el archivo `clientes.xsd` para los elementos `cliente` y sus elementos hijos.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:cli="http://www.ejemplo.com/cliente"
    targetNamespace="http://www.ejemplo.com/cliente"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">

  <xsd:element name="cliente">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="numeroID" type="xs:string"/>
        <xsd:element name="nombre" type="xs:string"/>
        <xsd:element name="telefono" type="xs:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Por otro lado también disponemos del esquema `pedido.xsd` para el elemento `pedido` y sus elementos hijos.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ped="http://www.ejemplo.com/pedido"
    targetNamespace="http://www.ejemplo.com/pedido"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">

  <xsd:element name="pedido">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="numeroID" type="xs:string"/>
        <xsd:element name="articulo" type="xs:string"/>
        <xsd:element name="precio" type="xs:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Ahora tendríamos que incorporar los dos esquemas al documento XML del ejemplo anterior. Sería de la siguiente manera.

```
<?xml version="1.0" encoding="utf-8"?>
<cli:cliente
    xmlns:cli='http://www.ejemplo.com/cliente'
    xmlns:ped='http://www.ejemplo.com/pedido'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.ejemplo.com/cliente
cliente.xsd http://www.ejemplo.com/pedido pedido.xsd'>

    <cli:numero_ID>1234</cli:numero_ID>
    <cli:nombre>José Pérez</cli:nombre>
    <cli:telefono>957000000</cli:telefono>d
```

```
    <ped:pedido>
      <ped:numero_ID>56789</ped:numero_ID>
      <ped:articulo>Caja de herramientas</ped:articulo>
      <ped:precio>187,90</ped:precio>
    </ped:pedido>
</cli:cliente>
```

Los espacios de nombre nacieron con posterioridad a los DTDs y por eso las DTDs no dan soporte a los espacios de nombre. Si combinamos varios documentos con sus correspondientes DTD, y queremos mezclar ambas páginas, habrá elementos que tengan el mismo nombre. La única solución es que el autor de cada documento especifique qué DTD usar cuando se valide un elemento dado. Es decir, que para elemento se especifica qué DTD usar, por lo que esto sólo tiene sentido si se usa más de un DTD. Entonces se usarían ambos DTD y utilizaríamos los espacios de nombre para distinguir aquellos elementos en los que no esté claro a qué DTD pertenecen. De esta forma, hay que definir los espacios de nombre usados para poderlos validar con DTDs:

5 Bibliografía

WIKIPEDIA, *Espacio de nombres XML*. *[a*cceso septiembre 2020]. Disponible en <https://es.wikipedia.org/wiki/Espacio_de_nombres_XML>

WIKIPEDIA, *Lenguaje de marcado. [a*cceso septiembre 2020]. Disponible en <https://es.wikipedia.org/wiki/Lenguaje_de_marcado#Historia>

TODOXML, *Documentos XML bien formados [a*cceso septiembre 2020]. Disponible en <https://sites.google.com/site/todoxmldtd/referencia/referencia-de-xml/07-xml-bien-formados#:~:text=Un%20documento%20XML%20bien%20formado,formado%20con%20un%20documento%20v%C3%A1lido.>

ARRANZ,D., *Apuntes de HTML*. *[a*cceso septiembre 2020]. Disponible en <https://www.dsi.uclm.es/personal/MiguelFGraciani/mikicurri/Docencia/LenguajesInternet0910/web_LI/Teoria/XML/Programaci%C3%B3n%20en%20castellano_%20Apuntes%20de%20XML.%20Escribir%20XML.htm>

CASTRO,J.M. y RODRÍGUEZ, J.R., *Lenguajes de Marcas y Sistemas de Gestión de Información*. Ed Garceta 2012 – ISBN: 978-84-1545-217-1

Libro Garceta

Libro Práctico XML de ENI

Web TodoXML

https://web.archive.org/web/20080813025523/http://www.programacion.com/tutorial/joa_xml/21/