1. Validación de documentos XML....3
2. DTD....5
2.1. Asociar una DTD a un documento XML....5
2.1.1-Declaración DTD interna....6
2.1.2-Declaración DTD externa....8
- DTD privadas....8
- DTD públicas....9
  - Reglas internas en declaraciones externas....10
2.2. Definición de esquemas con DTD....11
2.2.1-elementos....11
- contenidos genéricos....12
- Contenido de elementos....14
- contenido mezclado....18
2.2.2-atributos en DTD....19
- Atributos de ATTLIST....20
- Tipos de datos....21
2.2.3-Otros elementos....25
2.3. Limitaciones....25
2.3.1 La DTD no comprueba los tipos....25
2.3.2. Problemas en mezclar etiquetas y #PCDATA....26
2.3.3. Sólo acepta expresiones deterministas....26
3. Ejemplo de creación de una DTD....29
3.1. Solución....30
3.1.1-Asociar el archivo XML....30
3.1.2-Crear las reglas....30
4. Validar XML con una DTD....32

XML permite crear los lenguajes de marcas que queramos, cualquiera que sea el campo de actuación (se pueden crear las etiquetas necesarias para adaptarse a cualquier situación). Este es el punto fuerte de XML en relación a otros lenguajes de marcas: **se adapta a lo que queramos representar sin importar la complejidad que pueda tener**.

Pero los datos de los documentos XML normalmente deberán ser procesados por un programa de ordenador, y los programas de ordenador no dan tanta libertad como XML. Estos, no son muy buenos interpretando y entendiendo información que no se incluye en su diseño de proceso.

Supongamos que se ha desarrollado un programa para representar imágenes a partir de las etiquetas <dibujo>, <rectángulo> y <circulo>. Desde un punto de vista de la libertad que deja XML no habrá ningún problema para crear un archivo XML como este:

```
<dibujo >
    <rectángulo > 12,10,14,10 </ rectángulo >
    <línea > 13,13,25,25 </ línea >
</ dibujo >
```

El documento es perfectamente correcto desde un punto de vista de XML, pero el programa no sabrá qué hacer con la etiqueta <línea> ya que no ha sido programado para ese dato. Los programas normalmente se diseñan para procesar sólo tipos concretos de XML.

Por lo tanto, una de las cosas que tendrá que hacer un programa es comprobar que los datos del documento XML son correctos. Como esta tarea es muy pesada, se han definido **sistemas para comprobar que un documento XML contiene las etiquetas que debe contener y que están colocadas tal como se requiere**. El proceso de hacer estas comprobaciones se denomina **validación**.

1. Validación de documentos XML

El proceso de comprobar que unos archivos XML determinados siguen un determinado vocabulario, se denomina **validación** y los documentos XML que siguen las reglas del vocabulario se denominan **documentos válidos**. Hay que tener clara la diferencia entre lo que es un documento bien formado y un documento válido:

- Un documento es **bien formado** cuando sigue las reglas de XML.
- Un documento es **válido** si sigue las normas del vocabulario que tiene asociado.

> Un documento puede cumplir perfectamente con las reglas de creación de documentos XML y, por tanto, estar **bien formado**, pero en cambio no seguir las normas del vocabulario y, por tanto, no ser **válido**.

> Un documento puede ser bien formado, pero no válido, y en cambio, **si es válido** seguro que **es bien formado**.

La validación de documentos es un proceso corriente en lenguajes de marcas, ya que no siempre la persona que define cómo debe ser el documento, es la que genera los documentos (a veces los documentos llegarán de otros lugares, se generarán automáticamente ...).

> La **validación** es el proceso de comprobar que un documento **cumple con las reglas del vocabulario** en todos sus aspectos.

La validación permite asegurar que la información está exactamente en la forma en que debe estar. Esto es especialmente importante cuando se comparte información entre sistemas, ya que validar el documento es a**segurarse de que realmente la estructura de la información que se está pasando es la correcta**. Si se quiere hacer que dos programas situados en ordenadores diferentes colaboren, es necesario que la información que se pasan el uno al otro siga una estructura prefijada para enviarse mensajes.

Para forzar una determinada estructura en un documento, es necesario que haya alguna manera de definir aspectos como:

- en qué orden deben ir las etiquetas,
- cuáles son etiquetas correctas y cuáles no,
- en qué orden deben aparecer,
- qué atributos se pueden poner,
- qué contenido puede haber,
- etc.

XML hace esto mediante **lenguajes de definición de vocabularios** o lenguajes de **esquemas**.

Los lenguajes de definición de esquemas surgen por la necesidad de que los documentos XML sean procesados por programas (para extraer información, transformarlas, etc.), ya que los programas no son tan flexibles como XML y necesitan procesar datos con estructuras de documento cerradas.

Hay muchos lenguajes de definición de esquemas, pero los más utilizados en XML son:

- ***D**ocumento **T**ype **D**efinitions (DTD)*
- *W3C XML definition language*
- *relax NG*
- *Schematron*

> El proceso de validación se realiza mediante programas especiales llamados **procesadores o validadores** (en inglés se denominan *parsers*).

Los validadores, a menudo están en forma de bibliotecas para poder ser incorporados a los programas. No todos los procesadores pueden validar todos los lenguajes de definición de vocabularios y, por tanto, se deberán elegir en función del lenguaje que utilizamos.

Aunque los validadores sobre todo se utilizan como bibliotecas incorporadas en programas, también hay programas que permiten validar. Hay muchos programas que permiten validar documentos XML sin tener que programar, pero lo más habitual desde un punto de vista profesional suele ser utilizar algún editor XML especializado, tales como Oxygen XML Editor, EditiX XML Editor, Altova XMLSpy XML editor o Stylus Studio.

2. DTD

El DTD (**D**ocument **T**ype **D**efinitions), es un **lenguaje de definición de esquemas** que ya existía antes de la aparición de XML (se utilizaba en SGML). Al estar pensado para funcionar con SGML se podía utilizar en muchos de los lenguajes de marcas basados en él, como *XML* o *HTML*.
Cuando se definió XML se aprovechó para hacer una versión simplificada de DTD (el lenguaje de especificación de esquemas original). Así al utilizar esta versión original de DTD se permitiría mantener la compatibilidad con SGML, y por tanto se podría referenciar DTD en la especificación de XML (W3.Org/TR/REC-xml/).

El **objetivo principal de DTD** es proveer un mecanismo para validar las estructuras de los documentos XML y **determinar si el documento es válido o no**. Pero este no será la única ventaja que nos aportarán las DTD, sino que también es una ventaja a la hora de compartir información entre organizaciones, ya que si alguien tiene nuestra DTD nos puede enviar información en nuestro formato pudiéndola procesar con nuestro programa.

Durante mucho tiempo las DTD fueron el sistema más usado en XML, pero actualmente ha sido superado por ***XML Schemas***. DTD es todavía muy usado, sobre todo porque es mucho más sencillo.

2.1. Asociar una DTD a un documento XML

Para poder validar un documento XML se debe especificar cuál es el documento de esquemas que se utilizará. Si el lenguaje que se utiliza es DTD hay varias maneras de especificarlo, pero en general siempre implicará añadir una declaración *DOCTYPE* dentro del documento XML a validar.

Lo más habitual es añadir una referencia al DTD dentro del documento XML, mediante la etiqueta especial ***<!DOCTYPE***.

> Como la declaración de XML es opcional (si se declara debe ser lo primero que aparezca en un documento XML), la etiqueta DOCTYPE deberá ir siempre tras él.

Por lo tanto, sería **incorrecto** hacer:

```
<! DOCTYPE ...>
<? xml  version = "1.0" ?>
```

Pero, por otro lado, la etiqueta no puede aparecer en ningún otro lugar que no sea justo después de la declaración XML (no se puede incluir la etiqueta DOCTYPE una vez haya empezado el documento). No sería correcto:

```
<? xml  version = "1.0" ?>
<documento >
<!DOCTYPE ...>
</documento >
```

Ni tampoco el final:

```
<? xml  version = "1.0"  ?>
<documento >
</documento >
<!DOCTYPE ...>
```

Sí sería correcto:

```
<? xml  version = "1.0"  ?>
<!DOCTYPE ...>
<documento >
</documento >
```

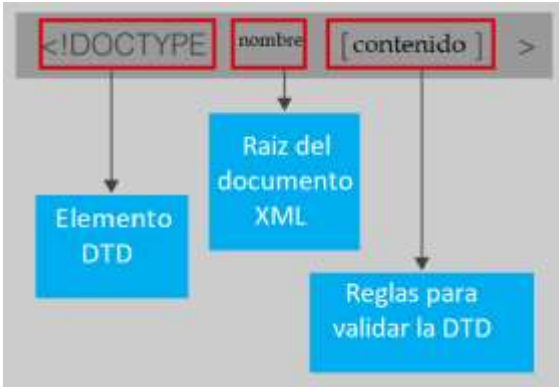
Hay dos maneras de incorporar DTD en un documento XML:

- declaración interna
- declaración externa

2.1.1-Declaración DTD interna

En las declaraciones DTD internas, las reglas de la DTD **están incorporadas en el documento XML**.

Declaración DOCTYPE interna



Por ejemplo, si se copian las líneas siguientes en un documento XML llamado prueba.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clase [
   <!ELEMENT clase (profesor, alumnos)>
   <!ELEMENT profesor (#PCDATA) >
   <!ELEMENT alumnos (nombre*) >
   <!ELEMENT nombre (#PCDATA) >
 ]>
<clase>
  <profesor > Marcelino Paz </profesor>
  <alumnos >
     <nombre> Federico Paz </nombre>
  </alumnos >
</clase>
```

Las declaraciones de DTD internas no se usan mucho porque tienen una serie de problemas que no las hacen ideales:

- Al tener la definición del esquema dentro del documento XML **no es fácil compartir** las declaraciones con otros documentos.
- Si se tienen muchos documentos XML, para cambiar ligeramente la declaración se tendrán que hacer **cambios** en todos.

Es más recomendable tener la DTD en un documento aparte y utilizarlo para validar todos los XML.

2.1.2-Declaración DTD externa

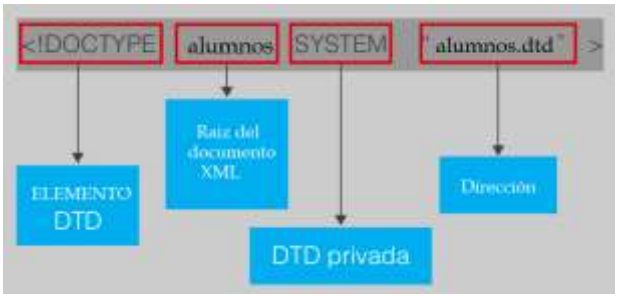
Para hacer una declaración externa también se utiliza la etiqueta ***DOCTYPE*** pero el formato es ligeramente diferente y prevé dos posibilidades:

- DTD privada
- DTD pública

- **DTD privadas**

Las definiciones de DTD privadas son las más corrientes, ya que, aunque se defina el vocabulario como privado, no hay nada que impida que el archivo se comparta o se publique en Internet. La definición de una DTD privada se hace de la manera:

Definición de una DTD privada en un documento XML



El campo de dirección especifica dónde está y qué nombre tiene el archivo que contiene las reglas. Se puede especificar utilizando una ***URI*** (***U***niform ***R***esource ***I***dentifier). Por tanto, se puede definir la DTD por medio de un nombre que por definición **es único**.

> Una ***URI*** es una cadena de caracteres que se utiliza para hacer **referencia única a un recurso** local, dentro de una red o en Internet.

- En el campo de dirección se pueden definir especificando el camino en el árbol de directorios de la máquina:

```
<!DOCTYPE clase SYSTEM “C:\dtd\classe.dtd">
```

- También se puede definir una DTD por medio de una dirección de Internet:

```
<! DOCTYPE clase SYSTEM "http://www.ioc.cat/clase.dtd">
```

Podemos definir dentro de un fichero XML una DTD que tenga el elemento raíz <alumnos> de este modo:

```
<!DOCTYPE alumnos SYSTEM “alumnos.dtd">
```

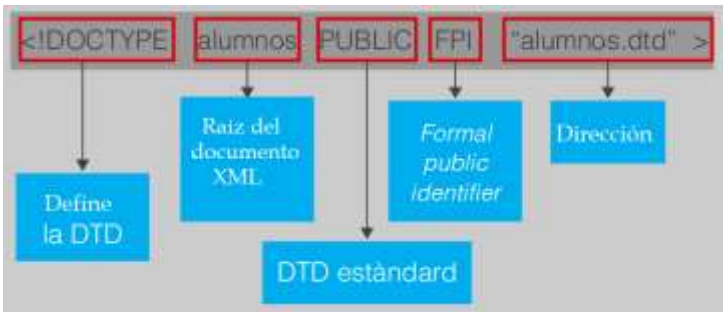
Una vez definido sólo hay que crear el archivo alumnos.dtd en el lugar adecuado con las reglas que definen el vocabulario:

```
<?xml  version = "1.0"  encoding = "UTF-8" ?>
<!ELEMENT clase (profesor, alumnos)>
<!ELEMENT profesor (#PCDATA)>
<!ELEMENT alumnos (nombre *)>
<!ELEMENT nombre (#PCDATA)>
```

- **DTD públicas**

Las definiciones ***PUBLIC*** están reservadas para DTD que estén definidas por organismos de estandarización, ya sean oficiales o no. En la definición de una DTD pública se añade un campo extra que hace de identificador del organismo de estandarización.

Definición de una DTD pública en un documento XML



La mayoría de los campos son idénticos a los de la DTD privada, pero en este caso se ha añadido un campo más, que es un identificador. El identificador puede estar en cualquier formato, pero lo más corriente es utilizar el formato **FPI** (**F**ormal **P**ublic **I**dentifiers).

El FPI se define por medio de un grupo de cuatro cadenas de caracteres separadas por dobles barras. En cada posición se especifica:

"Símbolo // Nombre del responsable de la DTD // Documento descrito // Idioma"

Definición de un FPI

campo	valores posibles
primero	Si el estándar no ha sido aprobado habrá un '-', mientras que si ha sido aprobado por un organismo no estándar tendremos un '+'. Y en cambio, si ha sido aprobado por un organismo de estándares, habrá una referencia.
Nombre del responsable	Quién es la organización o la persona responsable de definir el estándar.
documento descrito	Contiene un identificador único del documento descrito.
idioma	El código ISO del idioma en que está escrito el documento.

Por ejemplo, esta sería una declaración correcta:

```
<! DOCTYPE clase PUBLIC "-//JA//Clase1.0//ES" "http://www.juntadeandalucia.es/classe.dtd">
```

Un ejemplo de DTD pública que se utiliza mucho es la declaración de HTML 04:01:

```
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "Http://www.w3.org/TR/html4/strict.dtd">
```

- **Reglas internas en declaraciones externas**

Una definición externa también puede contener un apartado opcional que permita especificar un subconjunto de reglas internas.

```
<! DOCTYPE nombre SYSTEM "archivo.dtd" [reglas]>
<! DOCTYPE clase PUBLIC "-//JA//Clase1.0//ES" "archivo.dtd" [reglas]>
```

Si no hay reglas no es necesario especificar este apartado, se puede dejar en blanco. Por lo tanto, sería correcto definir la DTD de este modo:

```
<! DOCTYPE alumnos SYSTEM "alumnos.dtd" []>
```

Como de esta:

```
<! DOCTYPE alumnos SYSTEM "alumnos.dtd">
```

2.2. Definición de esquemas con DTD

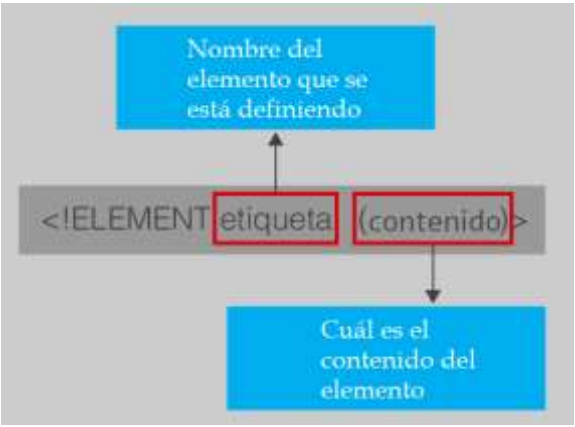
La parte más importante de un sistema de definición de vocabularios es definir cómo se hace para determinar el orden en que los elementos pueden aparecer y qué contenido pueden tener, qué atributos pueden tener las etiquetas, etc.

Esto, en DTD, se hace definiendo una serie de reglas por medio de un sistema de etiquetas predefinidas. A diferencia de lo que ocurre en XML, al hacer una definición en DTD las etiquetas están ya definidas. Para poder definir elementos y atributos se deberán usar etiquetas concretas.

2.2.1-elementos

Del mismo modo que los elementos son la base de XML, la definición de estos elementos es la base de la definición de archivos DTD. Siempre se deberán definir todos los elementos que componen el vocabulario y además especificar el contenido

Definición de una DTD



Es muy sencillo definir elementos con DTD; simplemente se define el nombre del elemento y cuál es el contenido.

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

Como en XML, los contenidos pueden ser datos o bien otros elementos. Podemos definir tres grandes grupos de contenidos en una DTD:

- Contenidos genéricos
- Contenido de elementos
- Contenido mezclado

- **contenidos genéricos**

Hay tres contenidos genéricos que se pueden utilizar para definir elementos: ANY, EMPTY y #PCDATA

Contenidos genéricos en una DTD

valor	significado
ANY	El contenido del elemento puede ser cualquier cosa.
EMPTY	El elemento no tiene contenido.
#PCDATA	El contenido de la etiqueta pueden ser datos.

Los elementos que estén definidos con ***ANY*** pueden contener cualquier cosa en su interior. Tanto etiquetas, como datos, o incluso una mezcla de las dos cosas.

Si se define persona de esta manera:

```
<!ELEMENT persona ANY >
```

- validará elementos que contengan datos

```
<persona> Federico Paz </persona >
```

- como elementos que contengan otros elementos

```
<persona>
   <nombre> Federico </nombre>
   <apellido> Paz </apellido>
</persona>
```

Por tanto, el contenido definido con ANY es muy potente pero también hace que se pierda el control de las cosas que se pueden validar, ya que lo validará prácticamente todo.

A veces hay elementos en XML que no es necesario que tengan un valor que aporte alguna información, ya que el nombre de la etiqueta puede tener algún tipo de significado semántico. Veamos en el siguiente ejemplo el elemento etiqueta <profesor>

```
</personas >
   <persona >
      <nombre > Pedro Martín </nombre >
      <profesor/>
   </persona >
   <persona >
      <nombre > Marcelino Paz </nombre >
   </persona >
</personas >
```

La etiqueta <profesor/> no necesita tener ningún contenido porque está implícito que la persona a la que se asigne la etiqueta será un profesor.
Los contenidos definidos con ***EMPTY*** están pensados para las etiquetas que no tendrán ningún contenido en su interior. Por lo tanto, el elemento <profesor/> del ejemplo anterior se definiría así:

```
<!ELEMENT profesor EMPTY >
```

Esta definición sirve tanto para los elementos que se definen utilizando el sistema de una sola etiqueta <profesor/> como para los que definen las etiquetas de apertura y cierre <profesor> </profesor>

El contenido genérico ***#PCDATA*** seguramente es el más usado para indicar que una etiqueta sólo tiene datos en su contenido.
Si partimos del ejemplo siguiente:

```
<persona >
   <nombre > Marcelino </nombre >
   <apellido > Paz </nombre >
</persona >
```

Se puede usar #PCDATA para definir el contenido de los elementos <nombre>y <apellido>porque en su interior sólo tienen datos.

```
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT apellido (#PCDATA) >
```

Pero no se puede usar, en cambio, para definir el elemento <persona>, ya que en su interior no sólo hay datos, sino que tiene los elementos etiqueta como son <nombre>y <apellido>.

- **Contenido de elementos**

Una de las situaciones habituales en un documento XML es que un elemento dentro de su contenido tenga otros elementos. Tenemos varias posibilidades para definir un contenido de elementos dentro de una DTD:

- Secuencias de elementos
- Alternativas de elementos
- Modificadores

Si miramos el ejemplo siguiente:

```
<persona >
   <nombre > Pedro </nombre >
   <apellido > Martínez </apellido >
</persona >
```

<persona> es un elemento que tiene como contenido una secuencia de otros elementos.

En una DTD, se soluciona esta situación definiendo explícitamente cuáles son los hijos del elemento, separándolos por comas.

```
<!ELEMENT persona (nombre,apellido) >
```

Nunca hay que olvidar que las secuencias tienen un orden explícito y, por tanto, sólo validarán si el orden en que se definen los elementos es idéntico al orden en que aparecerán en el documento XML.

Veamos el siguiente documento:

```
<comida>
  <almuerzo> Arroz </almuerzo>
  <cena> Sopa </cena>
</comida>
```

Si el elemento <comida> se define con la secuencia (almuerzo,cena), será válido, ya que los elementos que contiene están en el mismo orden que en el documento.

```
<!ELEMENT comida (almuerzo,cena) >
```

Pero no validaría, en cambio, si definiéramos la secuencia al revés (cena,almuerzo).

```
<!ELEMENT comida (cena,almuerzo)
```

Si diseñáramos un XML para definir las fichas de personal de una empresa podríamos tener una etiqueta <personal> y luego definir el cargo con otra etiqueta. El presidente se podría definir así:

```
<personal >
  <presidente > Juan María Flores < presidente >
</personal >
```

Y uno de los trabajadores así:

```
<personal >
   <trabajador > Pedro Vila </trabajador >
</personal >
```

Podemos hacer que una DTD valide ambos casos utilizando el ***operador de alternativa (|)***; nos permite que se pueda elegir entre una de las opciones que se le ofrecen a la hora de validar un documento XML.

Podemos validar los dos documentos XML anteriores definiendo <personal> de la siguiente manera:

```
<!ELEMENT personal (trabajador | presidente) >
```

No hay ninguna limitación definida para poner alternativas; podemos poner tantas como nos hagan falta.

```
<!ELEMENT personal (trabajador | presidente | informático | gerente >
```

También se permite especificar la definición con ***EMPTY*** para indicar que el valor puede aparecer o no:

```
<!ELEMENT alumno (delegado | EMPTY) >
```

Combinar el operador de alternativa con *PCDATA* es más complejo.

> No hay nada que impida mezclar las secuencias y las alternativas de elementos a la hora de definir un elemento con DTD. Por lo tanto, se pueden hacer las combinaciones que hagan falta entre secuencias y alternativas.

Si tenemos un XML que define la etiqueta <circulo> a partir de sus coordenadas del centro y del radio o el diámetro, podemos definir el elemento combinando las secuencias con una alternativa. La etiqueta <circulo> contendrá siempre dentro de sí las etiquetas <x>, <y> y después puede contener también <radio> o <diametro>:

```
<!ELEMENT círculo (x, y, (radio | diametro)) >
```

> Combinar secuencias y alternativas permite saltarse las restricciones de orden de las secuencias.

El siguiente ejemplo nos permite definir una persona de nombre y apellido en cualquier orden:

```
<!ELEMENT persona ((apellido, nombre) | (nombre, apellido)) >
```

En los casos en que las etiquetas se repitan un número indeterminado de veces, será imposible especificar todas ellas en la definición del elemento. **Los modificadores** sirven para especificar cuántas instancias de los elementos hijos puede haber en un elemento.

Modificadores aceptados en el contenido de los elementos

modificador	significado
?	Indica que el elemento tanto puede que sea como no.
+	Se utiliza para indicar que el elemento tiene que salir una vez o más.
*	Indica que puede que el elemento esté repetido un número indeterminado de veces, o bien no estar allí.

Si queremos especificar que una persona pueda ser identificada por medio del nombre y uno o más apellidos podríamos definir el elemento <persona> de esta manera:

```
<!ELEMENT persona (nombre, apellido +) >
```

Esta expresión indica que apellido tiene que salir al menos una vez.

Nos validará este ejemplo:

```
<persona>
   <nombre > Juan </nombre >
   <apellido > Pérez </apellido >
</persona >
```

Y también este otro:

```
<persona >
   <nombre > Juan </nombre >
   <apellido > Pérez < apellido >
   <apellido > García </apellido >
</ persona >
```

Esta no es la mejor manera de definir lo que queremos, ya que también nos aceptará personas con 3 apellidos, 4 apellidos, o más; el modificador ideal para forzar que sólo pueda haber uno o dos apellidos sería "***?***", empleándolo como sigue:

```
<!ELEMENT persona (nombre, apellido, apellido?) >
```

El modificador "*****" aplicado al mismo ejemplo nos permitiría aceptar que alguna persona no tuviera apellidos o que tenga un número indeterminado:

```
<!ELEMENT persona (nombre, apellido*) >
```

Los modificadores aplicados detrás de un paréntesis implican aplicarlos a todos los elementos de dentro de los paréntesis:

```
<!ELEMENT escritor (( libro, fecha)* | artículos+ )  >
```

- **contenido mezclado**

Se pensaron para poder definir contenidos que contengan texto que se va a ir intercalando con otros elementos como, por ejemplo:

```
<carta >
    Estimado
    <empresa > Hierros Puig </empresa>
    : Sr.
    <director > Manel García </director >
    Le envío esta carta para comunicarle que le hemos enviado su pedido
    <pedido > 145 </pedido >
    a la dirección que nos proporcionó.    Atentamente,
    <empresa > Ferretería, SA </empresa >
</carta >
```

El *contenido mezclado* se define usando el tipo ***#PCDATA*** (siempre en primer lugar) y luego se añaden los elementos con la ayuda del ***operador de alternativa, "|"*** y se termina todo el grupo con ***el modificador "*"***."

```
<!ELEMENT carta (#PCDATA | empresa | director | pedido) * >
```

> Hay que tener en cuenta que este contenido sólo sirve para controlar que los elementos puedan aparecer, pero no hay ninguna manera de controlar en qué orden aparecerán los diferentes elementos.

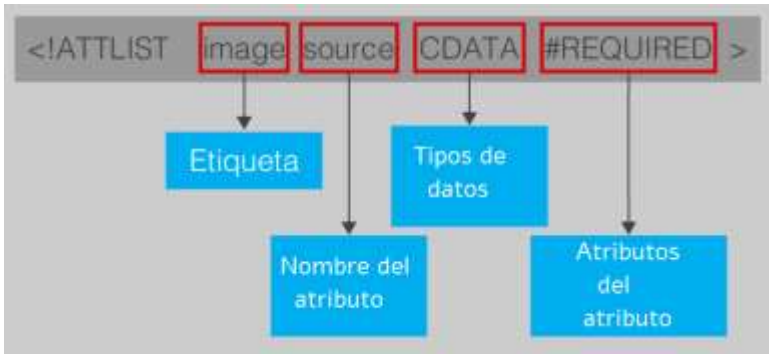
Vamos a definir una DTD que valide el ejemplo presentado al principio de este apartado de la siguiente manera:

```
<!ELEMENT carta (#PCDATA | empresa | director) *>
<!ELEMENT empresa (#PCDATA)>
<!ELEMENT director (#PCDATA)>
```

2.2.2-atributos en DTD

En las DTD se especificarán explícitamente, cuáles son los atributos de cada etiqueta. La declaración de atributos se hace con la etiqueta especial ***ATTLIST***.

Definición de un atributo en DTD



Los dos primeros valores de la definición del atributo son el nombre de la etiqueta y el nombre del atributo, ya que al definir un atributo siempre se especifica a qué etiqueta pertenece. Una de las críticas que se han hecho a las DTD ha sido, que **no se pueden hacer atributos genéricos**.

> Si alguien quiere definir un atributo que sea compartido por todos los elementos de su vocabulario debe ir especificando el atributo para todos y cada uno de los elementos.

Para definir un atributo llamado nombre perteneciente al elemento <persona> lo podemos hacer de esta manera:

```
<!ATTLIST persona nombre CDATA #IMPLIED >
```

Si un elemento necesita varios atributos tenemos que especificar todos los atributos en varias líneas *ATTLIST*. Por ejemplo, definimos los atributos nombre y apellido del elemento <persona> de esta manera:

```
<!ATTLIST persona nombre CDATA #REQUIRED >
<!ATTLIST persona apellido CDATA #REQUIRED >
```

Se puede hacer la definición de los dos atributos con una sola referencia *ATTLIST*:

```
<!ATTLIST persona nombre CDATA #REQUIRED
                 apellido CDATA #REQUIRED >
```

Las dos declaraciones anteriores nos permitirían validar los atributos del elemento <persona>de este ejemplo:

```
<persona nombre = "Federico" apellido = "Pérez" />
```

- **Atributos de ATTLIST**

Los elementos *ATTLIST* también pueden tener atributos que permiten definir características sobre los atributos. Estos atributos se pueden ver en la tabla.

Atributos de ATTLIST

atributo	significado
#IMPLIED	El atributo es opcional.
#REQUIRED	El atributo es obligatorio. El elemento debe tener definido o no validará.
#FIXED	Se utiliza para definir atributos que tienen valores constantes e inmutables. Se debe especificar, ya que es permanente.
#DEFAULT	Permite especificar valores predeterminados en los atributos.

Por lo tanto, si se define un atributo de este modo:

```
<! ATTLIST equipo posición ID #REQUIRED >
```

está obligando a que cuando en un documento XML se defina el elemento <equipo>, se especifique obligatoriamente el atributo posición y que además su valor no se repita en el documento, ya que es de tipo ID (más adelante veremos los tipos de datos para atributos).

En cambio, definiendo el atributo DNI de <persona> con **#IMPLIED** se permitirá que al crear el documento XML del atributo DNI aparezca o no.

```
<!ATTLIST persona dni NMTOKEN #IMPLIED >
```

Al definir un atributo como **#FIXED** o, como **#DEFAULT,** se le debe especificar el valor.

Este valor se especifica a continuación del atributo y debe ir entre comillas:

```
<! ATTLIST documento versión CDATA #FIXED "1.0" >
<! ATTLIST documento codificación NMTOKEN #DEFAULT "UTF-8" >
```

Los atributos de tipo ***#FIXED*** deben tener el valor especificado en la definición y éste no se puede cambiar, mientras que los valores definidos con ***#DEFAULT*** sí pueden ser cambiados.

- **Tipos de datos**

El tercer parámetro de una declaración ATTLIST sirve para definir qué tipos de datos puede tener un atributo. Los atributos en DTD no usan los tipos de datos de los elementos a los que pertenecen, sino que se definen los suyos propios.

Los tipos de datos de los atributos se pueden ver en la tabla siguiente:

Tipos de datos de los atributos de una DTD

tipo	significado
CDATA	Puede contener cualquier cadena de caracteres aceptable. Se puede utilizar para precios, URL , direcciones de correo electrónico, etc.
enumeraciones	Se utilizan para definir que el atributo debe tener uno de los valores especificados.
ID	El atributo se podrá utilizar como identificador de un elemento. Su valor será único.
IDREF o IDREFS	El valor son referencias a un ID que debe existir. El plural es para definir que es una lista.
ENTITY o entidad	Las entidades permiten definir constantes para el documento y, por tanto, el valor del atributo debe ser una entidad.
NMTOKEN o NMTOKENS	Especifican cadenas de caracteres que sólo tengan caracteres permitidos para XML. Por tanto, no se permiten los espacios.
Notation	Permite que el atributo sea de una notación declarada anteriormente.

El tipo ***CDATA*** es un tipo de datos prácticamente idéntico al tipo *#PCDATA* de las etiquetas. En un *CDATA* se puede poner cualquier dato en formato de texto tanto si tiene espacios como si no los tiene.

Por lo tanto, la declaración siguiente:

```
<!ATTLIST empresa nombre CDATA #REQUIRED >
```

permitiría definir etiquetas empresa como estas:

```
<empresa nombre = "Microsoft Corporation" />
<empresa nombre = "6tems" />
```

Es importante recordar que los números también validarían con un tipo CDATA porque interpretados en forma de texto no dejan de ser simplemente caracteres:

```
<empresa nombre = "12" />
```

Los atributos también pueden ser especificados definiendo cuáles pueden ser los valores correctos para el atributo. Estos valores se especifican literalmente con el operador de selección:

```
<!ATTLIST módulo inicio (septiembre | febrero) # REQUIRED >
```

Por tanto, según la definición, el atributo inicio debe existir y sólo puede tener los valores "setiembre" o "febrero".

Se pueden poner múltiples opciones en una ***enumeración***. Por ejemplo, podemos comprobar que el valor del atributo sea un número entre 1 y 12 de esta manera:

```
<! ATTLIST módulo número (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12) >
```

El tipo de datos ***ID*** sirve para definir atributos que se puedan usar como identificadores de un elemento dentro del documento. Hay que tener en cuenta que:

- Los valores asignados **no se pueden repetir** dentro del documento. Esto los hace ideales para identificar unívocamente los elementos de un documento.
- Los valores **deben empezar por una letra o un subrayado**.

Los valores del atributo posición de la definición siguiente no se podrán repetir dentro del mismo documento:

```
<! ATTLIST equipo posición ID #REQUIRED >
```

Con la declaración anterior si validamos el siguiente documento, se generaría un error de validación en el atributo posición, ya que se repite el valor "primero" en los dos elementos.

```
<clasificación >
  <equipo posición = "primero" > FC Barcelona </ equipo >
  <equipo posición = "primero" > Real Madrid </ equipo >
</clasificación >
```

Para que se valide este documento hay que asegurarse de que los atributos ID no tengan el mismo valor:

```
<clasificación >
  <equipo posición = "primero" > FC Barcelona </equipo >
  <equipo posición = "segundo" > Real Madrid </equipo >
</clasificación>
```

Los tipos ***IDREF*** y ***IDREFS*** se utilizan para definir valores de atributos que hacen referencia a elementos que tengan un valor de tipo *ID*.

Sólo tienen sentido en vocabularios que tengan atributos con valor ID.

IDREF se utiliza para hacer referencia a un valor ID:

```
<! ATTLIST receta id ID #REQUIRED >
<! ATTLIST ingrediente ref IDREF #REQUIRED
```

Con la declaración que se acaba de especificar se puede validar un documento como el siguiente:

```
<libro-cocina >
   <recetas >
      <receta id = "recepta1" > Patatas fritas </receta >
      <receta id = "recepta2" > Patatas hervidas </receta >
   </recetas >
   <ingredientes >
      <ingrediente ref = "recepta1" > Aceite </ingrediente >
      <ingrediente ref = "recepta2" > Agua </ingrediente >
   </ingredientes >
</libro-cocina >
```

IDREFS permite especificar una lista de valores *ID* en el mismo atributo. Por ejemplo, si se define el atributo ingrediente como tipo *IDREFS*:

```
<! ATTLIST ingrediente ref IDREFS #REQUIRED >
Se podría poner una lista de ID como valor del atributo:
<ingrediente ref = "recepta1 recepta2" > Patatas </ ingrediente >
```

Los tipos ***NMTOKEN*** permiten especificar que los atributos pueden tener cualquier carácter aceptado por el XML. Así, la declaración siguiente:

```
<! ATTLIST hombre nacimiento NMTOKEN #REQUIRED >
```

permitirá validar este elemento:

```
<hombre nacimiento = "1970" />
```

Pero no lo hará con este otro, porque tiene un espacio:

```
<hombre nacimiento = "siglo XX" />
```

El valor en plural ***NMTOKENS*** permite especificar una lista de valores en vez de uno solo:

```
<coordenadas posición = "x y"/>
```

Para permitir valores que han sido declarados como “Notation” se utiliza la etiqueta “***<!NOTATION”***. Se suele usar para especificar datos no-XML como las declaraciones tipo MIME:

```
<!Notation GIF SYSTEM "image /gif" >
<!Notation JPG SYSTEM "image /jpeg" >
<!Notation PNG SYSTEM "image /png" >
<!ATTLIST persona
  photo_type Notation (GIF | JPG | PNG) #IMPLIED >
```

El tipo ***ENTITY*** indica que el valor es una referencia a un valor externo que no se debe procesar. En general suelen contener valores binarios externos. Utilizar ENTITY implicará haber declarado la entidad con *<!ENTITY*:

```
<!ATTLIST persona foto ENTITY #IMPLIED
<!ENTITY pere SYSTEM "Pere.jpg">
```

La especificación permitirá que se defina el atributo persona con el valor de la entidad y que automáticamente sea asociado a la imagen:

```
<persona nombre = "pere" />
```

2.2.3-Otros elementos

<!ELEMENT> y ***<!ATTLIST>*** no son las únicas que se pueden usar para declarar DTD. Hay algunas más que en determinados casos se pueden utilizar también

Otras etiquetas posibles en una DTD

etiqueta	Se utiliza para ...
<! ENTITY>	Definir referencias a archivos externos que no se deben procesar.
<! Notation>	Incluir datos que no sean XML en el documento.
<! INCLUDE>	Hacer que partes de la DTD añadan al procesamiento.
<! IGNORE>	Hacer que partes de la DTD sean ignoradas por el procesador.

2.3. Limitaciones

Una de las críticas que se ha hecho al uso de **DTD** para definir lenguajes XML, es que **no está basado en XML**. La DTD no sigue el estándar para definir los documentos de XML y, por tanto, para usarlo, hay que aprender un nuevo lenguaje. Si la DTD estuviera basada en XML no sería necesario conocer de qué manera se deben definir los elementos, marcar las repeticiones, los elementos huecos, etc., ya que se haría con elementos propios de XML.

Aun así, el hecho de que DTD no sea un lenguaje XML es un problema menor si se tienen en cuenta el resto de limitaciones que tiene:

- No comprueba los **tipos**
- Presenta problemas en **mezclar** etiquetas y #PCDATA
- Sólo acepta **expresiones deterministas**.

2.3.1 La DTD no comprueba los tipos

Uno de los problemas más importantes que nos encontraremos a la hora de usar DTD para definir nuestro vocabulario es que no tiene ninguna manera de comprobar los tipos de datos que contienen los elementos. A menudo los nombres de los elementos ya determinan que el contenido que habrá será de un tipo determinado (un número, una cadena de caracteres, una fecha, etc.) pero la DTD no deja que se le especifique qué tipo de datos se quiere poner.

Por lo tanto, si alguien rellena con cualquier cosa un elemento <día>, el documento será

válido a pesar de que el contenido no sea una fecha:

```
<día > chocolate </día >
```

Al no poder comprobar el tipo del contenido, una limitación importante añadida es que no hay ninguna manera de poder poner restricciones. Por ejemplo, no podemos especificar que una fecha esté entre los años 1900 y 2012.

2.3.2. Problemas en mezclar etiquetas y #PCDATA

Una limitación más compleja de ver es que no se pueden mezclar etiquetas y #PCDATA en expresiones si el resultado no es lo que se conoce como " *contenido mezclado* ".
Un ejemplo de ello consistiría en hacer una definición de un ejercicio como un enunciado de texto, y que tenga varios apartados que también contengan texto.

```
<ejercicio >
  Lea el texto "Validación de documentos XML" y responde a las siguientes preguntas:
  <apartado numero = "1" > ¿Qué significan las siglas XML? </apartado >
  <apartado numero = "2" > ¿Qué es un DTD? </apartado >
</ejercicio >
```

Lo más sencillo sería mezclar un *#PCDATA* y la etiqueta <apartado>, pero es incorrecto:

```
<!ELEMENT ejercicio (#PCDATA | apartado *)>
```

Además, no se permite que se hagan declaraciones duplicadas de elementos, o sea que tampoco lo podemos arreglar con:

```
<!ELEMENT ejercicio (#PCDATA)>
<!ELEMENT ejercicio (apartado *)>
```

La única manera de combinarlo sería utilizar la fórmula del *contenido mezclado*:

```
<!ELEMENT ejercicio (#PCDATA | apartado) *>
```

2.3.3. Sólo acepta expresiones deterministas

Otra de las limitaciones de las DTD es que obliga a que las expresiones tengan que ser siempre deterministas. Veamos el documento DTD:

```
<!ELEMENT clase (profesor | alumnos)>
<!ELEMENT profesor (nombre, apellidos)>
<!ELEMENT alumnos (alumno*)>
<!ELEMENT alumno (nombre, apellido)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
```

Cuando se analiza un archivo XML se define cuál es la raíz de la DTD. Si consideramos que la raíz es el elemento <clase>, el proceso de validación comenzará en la primera línea que nos dice que para que el documento sea válido, después de la raíz debe haber un elemento <profesor>, o un elemento <alumnos>. Si el que llega es un <profesor> el procedimiento de validación pasará a evaluar la segunda definición y si llega un <alumnos> pasará a la tercera definición. Si seguimos con la evaluación veremos que realmente el validador, cuando se encuentra con una alternativa, acaba siempre yendo a una determinada expresión. Esto es porque la DTD analizada es ***determinista***.

En el ejemplo de abajo***, el validador debe poder elegir, al leer el primer elemento, con cuál de las alternativas debe quedarse***. Si me llega un elemento <nombre> me quedo con la alternativa de la izquierda, (nombre, apellido) y si me llega un <alias> me quedo con la de la derecha, (alias, nombre, apellido).

```
<!ELEMENT persona (nombre, apellido | alias, nombre, apellido)>
```

Si en algún momento alguien crea un documento que no sea determinista la validación fallará. Esto ocurrirá si en algún momento el validador se encuentra que no puede decidir cuál es la alternativa correcta.

```
<!ELEMENT terrestre (persona, hombre | persona, mujer)>
```

Cuando desde el elemento <terrestre> nos llegue un elemento <persona>, el procesador no tendrá ninguna manera de determinar si estamos haciendo referencia al elemento <persona> de la expresión de la derecha (persona, hombre), o bien el de la izquierda (persona, mujer), y por tanto fallará porque tiene dos opciones posibles. Es una expresión **no determinista.**

A menudo las expresiones no deterministas las podemos expresar de otra manera, de modo que se conviertan deterministas. El ejemplo anterior se podía haber escrito de una

manera determinista para que al llegar un elemento <persona> no haya dudas.

```
<!ELEMENT terrestre (persona, (hombre | mujer))>
```

Así forzamos a que siempre aparezca un elemento <persona> y después habrá la alternativa entre un <hombre> o una <mujer>; ahora es ya determinista.

Las expresiones no deterministas son más corrientes de lo que parece, ya que los modificadores las pueden provocar y puede que no lo parezcan. Si se quisiera escribir una expresión que determine un libro a partir del autor o del título, en cualquier orden:

```
<!ELEMENT libro (autor?, titulo? | titulo?, autor?)>
```

Pero la expresión anterior es incorrecta, ya que si el validador se encuentra un <autor>, no sabe si es el autor del lado derecho de la condición (autor?, titulo?), o bien es el del lado izquierdo que no tiene título.

La expresión **determinista** idéntica a la anterior sería:

```
<!ELEMENT libro (autor, título? | titulo, autor? | EMPTY)>
```

> Las DTD se siguen usando porque son sencillas de crear, pero hay que recordar siempre las limitaciones que tienen, y tener presente que no siempre se adaptarán perfectamente a lo que se quiere hacer.

3. Ejemplo de creación de una DTD

Una empresa ha preparado una tienda en Internet que genera los pedidos de los clientes en un formato XML; los pedidos se envían al programa de gestión de manera automática.

Los XML que se generan contienen datos del cliente y del pedido realizados:

```
<?xml  version = "1.0"  encoding = "UTF-8"?>
<!DOCTYPE pedido SYSTEM "pedido.dtd">
<pedido numero="26" día="2011-12-01">
    <cliente código ="20">
        <nombre > Frederic </nombre>
        <apellido > García </apellido>
    </cliente>
    <artículos>
        <artículo>
            <descripción> Yoda Mimobot USB Flash Drive 8GB </descripción>
            <cantidad> 5 </cantidad>
            <precio> 38.99 </precio>
        </artículo>
        <artículo>
            <descripción> Darth Vader Half Helmet Case for iPhone </descripción>
            <cantidad> 2 </cantidad>
            <precio> 29.95 </precio>
        </artículo>
    </artículos>
    <total valor = "254.85" />
</pedido>
```

Antes de enviarlo al programa, han decidido que para tener más seguridad se hará un paso previo que consistirá en validar el documento. Para ello necesitan que se genere la DTD.

3.1. Solución

Se deberán hacer dos cosas:

1. Asociar las reglas en el archivo XML.
2. Crear un fichero con las reglas, que se llamará "pedido. dtd".

3.1.1-Asociar el archivo XML

En la segunda línea del documento se especifica la regla ***DOCTYPE*** para asociar el fichero con la DTD.

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

3.1.2-Crear las reglas

No hay una manera única de crear una DTD, pero normalmente seguir un método puede ayudar a evitar errores. El método que se utilizará consiste en empezar por la raíz para ir definiendo las hojas por niveles.

La raíz del documento es el elemento <pedido>, que tiene tres hijos:

```
<!ELEMENT pedido (cliente, artículos, total) >
```

También tiene dos atributos, numero y día, que sólo pueden tener valores sin espacios y, por tanto, serán ***NMTOKEN***. Son datos obligatorios por motivos fiscales.

```
<!ATTLIST pedido numero NMTOKEN #REQUIRED >
<!ATTLIST pedido día NMTOKEN #REQUIRED >
```

Una vez definido el primer nivel podemos pasar a definir los otros niveles. Por un lado, el elemento <cliente>, que tiene un atributo para los clientes existentes; no tendrá para los nuevos:

```
<!ELEMENT cliente (nombre, apellido) >
<! ATTLIST cliente código NMTOKEN #IMPLIED >
```

Por otra parte, el elemento <total>, que está vacío, pero tiene un atributo:

```
<!ELEMENT total EMPTY >
<!ATTLIST total valor CDATA #REQUIRED >
```

También el elemento <artículos>, que contiene una lista de los artículos que ha comprado el cliente. Como no tendría sentido hacer un pedido sin artículos el ***modificador*** que se utilizará será **+**.

```
<!ELEMENT artículos (artículo+) >
```

Definimos el elemento <artículo>:

```
<!ELEMENT artículo (descripción, cantidad, precio) >
```

Para terminar, sólo quedan los elementos que contienen datos:

```
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT apellido (#PCDATA) >
<!ELEMENT descripción (#PCDATA) >
<!ELEMENT cantidad (#PCDATA) >
<!ELEMENT precio (#PCDATA) >
```

El archivo resultante será (pedido.dtd):

```
<!ELEMENT pedido (cliente,artículos,total)>
<!ATTLIST pedido numero NMTOKEN #REQUIRED>
<!ATTLIST pedido día NMTOKEN #REQUIRED>
<!ELEMENT cliente (nombre,apellido)>
<!ATTLIST cliente código NMTOKEN #IMPLIED>
<!ELEMENT total EMPTY>
<!ATTLIST total valor CDATA #REQUIRED>
<!ELEMENT artículos (artículo+)>
<!ELEMENT artículo (descripción, cantidad, precio)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT descripción (#PCDATA)>
<!ELEMENT cantidad (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
```

4. Validar XML con una DTD

¿Cómo podemos saber si un documento XML es válido conforme a su DTD? La mayoría de los navegadores no tienen en cuenta si el documento se ajusta a su DTD sino simplemente si el documento está bien formado. Es decir, sólo tienen en cuenta que el documento se ajuste a las normas básicas de creación de archivos XML.

- Ya vimos en unidad anterior que podemos poner el atributo standalone="yes" en la primera declaración del archivo XML para comprobar si el XML se ajusta a la DTD:

```
<?xml version="1.0" encode="UTF-7" standalone="yes"?>
```

- También podemos usar el validador del W3C.

- Existen editores como ***XML Copy Editor***, también Oxygen, …