Tema 2: Definición de esquemas y vocabularios en XML (DTD)

Introducción

DTD (Document Type Definitions) es un lenguaje de definición de esquemas. Su objetivo principal es proveer un mecanismo para validar las estructuras de los documentos XML y determinar si un documento es válido o no.

Consideramos que un documento XML es válido si verifica las reglas escritas en el DTD asociado.

Además, con los DTDs podemos compartir información entre organizaciones, ya que nos indican con qué formato espera una organización que le llegue la información escrita en XML.

Asociar un DTD a un documento XML

Existen múltiples formas de asociar un documento DTD a un documento XML. Nosotros trabajaremos con DTDs privadas externas. Esto significa que escribiremos nuestras reglas DTD en un archivo que estará junto al documento XML.

Para indicar que un documento XML está asociado a un documento DTD, añadiremos el siguiente encabezado, justo después de la línea <?xml ... ?>:

```
<!DOCTYPE etiqueta_raíz SYSTEM "nombre_archivo.dtd">
```

Por ejemplo, si tenemos un archivo *alumno.xml* como el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<alumno>Federico López Roma</alumno>
```

Le asociaremos el archivo *alumno.dtd* con la declaración:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE alumno SYSTEM "alumno.dtd">
<alumno>Federico López Roma</alumno>
```

El archivo *alumno.dtd* debe estar en la misma carpeta que *alumno.xml*, o si no, habrá que indicar la ruta para llegar a él.

Elementos en DTD

En un archivo DTD debe haber una declaración para cada elemento y cada atributo de su correspondiente XML. En general la sintaxis es:

```
<!ELEMENT etiqueta (contenido)>
```

Iremos viendo cómo se codifica cada tipo de elemento.

Etiqueta con contenido ANY

Si no queremos especificar el contenido de un elemento, podemos escribir ANY, que significa que puede ser cualquiera. Por ejemplo:

```
<!ELEMENT alumno ANY>
```

En la línea anterior estamos diciendo que la etiqueta alumno puede contener cualquier tipo de contenido.

La palabra ANY habría que evitar usarla, salvo que no haya más remedio, ya que no aporta ninguna información sobre la etiqueta.

Etiqueta con contenido #PCDATA

Una etiqueta tiene contenido #PCDATA cuando sólo contiene texto sin etiquetas. En el ejemplo del documento *alumno.xml*, el elemento alumno sólo contiene texto sin etiquetas, luego su DTD quedaría así:

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

Comprobación usando xmllint

Además de verificar el DTD con Eclipse, podemos usar *xmllint*. Para ello nos situamos en la carpeta donde están el archivo XML y el DTD. Abrimos una terminal y ejecutamos el comando:

```
xmllint archivo.xml --noout --valid
```

En el caso de que el documento xml no cumpliera el DTD, nos aparecería un mensaje de error. De otro modo, no saldrá nada.

Etiqueta que contiene una secuencia de etiquetas

A menudo, un elemento contiene una secuencia de otros elementos. Por ejemplo, para especificar mejor la información del alumno, podemos querer separar el nombre de pila y los apellidos, quedando el documento así:

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

<!DOCTYPE alumno SYSTEM "alumno.dtd">

<alumno>
	<nombre>Federico</nombre>
	<apellidos>López Roma</apellidos>
</alumno>
```

Ahora la etiqueta alumno tiene otras etiquetas, *nombre* y *apellidos*. La forma de indicarlo en el DTD es ponerlas entre paréntesis separadas por comas:

```
<!ELEMENT alumno (nombre, apellidos)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
```

Por supuesto, también hemos añadido una línea para cada una de las etiquetas nombre y apellidos.

Observación: en una secuencia el orden importa. Si cambiáramos el orden de las etiquetas nombre y apellidos, el DTD sería incorrecto:

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

<!DOCTYPE alumno SYSTEM "alumno.dtd">

<alumno>
      <apellidos>López Roma</apellidos>          ERROR
      <nombre>Federico</nombre>
</alumno>
```

Etiquetas vacías (EMPTY)

Supongamos que en el documento *alumno.xml* queremos añadir una etiqueta <*delegado*> que indique que el estudiante es delegado de clase. Esta etiqueta estará vacía siempre:

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

<!DOCTYPE alumno SYSTEM "alumno.dtd">

<alumno>
      <nombre>Federico</nombre>
      <apellidos>López Roma</apellidos>
      <delegado/>
</alumno>
```

La forma de indicar que el elemento *delegado* siempre va a ser vacío, es mediante la expresión EMPTY:

```
<!ELEMENT alumno (nombre, apellidos, delegado)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT delegado EMPTY>
```

Si ahora intentásemos escribir algo entre etiquetas delegado, la validación fallaría:

```
<delegado>Sí</delegado>       ERROR
```

Si queremos que el documento sea más realista, podemos indicar que la etiqueta <*delegado* /> puede no aparecer, agregando un "?" en la primera línea del DTD:

```
<!ELEMENT alumno (nombre, apellidos, delegado?)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT delegado EMPTY>
```

De este modo, si un alumno es delegado de su clase tendrá la etiqueta <*delegado* /> y si no lo es, no la tendrá.

Etiqueta con otras etiquetas: elección

A menudo nos encontraremos con elementos que pueden contener distintas etiquetas dentro, no necesariamente formando un única secuencia. Por ejemplo, nuestro alumno puede tener un elemento optativa donde se indique mediante una u otra etiqueta el nombre de la asignatura optativa que cursa.

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

<!DOCTYPE alumno SYSTEM "alumno.dtd">

<alumno>
      <apellidos>López Roma</apellidos>
      <nombre>Federico</nombre>
      <delegado />
      <optativa>
            <cultura_clasica />
      </optativa>
</alumno>
```

En lugar de <*cultura_clasica* /> podrían aparecer las optativas <*teatro* /> e <*informatica* />.

En el DTD esto se escribirá separando las opciones por símbolos "|":

```
<!ELEMENT alumno (nombre, apellidos, delegado?, optativa)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT delegado EMPTY>
<!ELEMENT optativa (teatro|informatica|cultura_clasica)>
<!ELEMENT teatro EMPTY>
<!ELEMENT informatica EMPTY>
<!ELEMENT cultura_clasica EMPTY>
```

El operador de alternativa, "|", también se puede combinar con secuencias. Por ejemplo, si queremos indicar que el nombre y los apellidos pueden aparecer en orden inverso (apellidos y luego nombre) reescribiremos la primera línea así:

```
<!ELEMENT alumno (((nombre, apellidos)|(apellidos, nombre)), delegado?,
optativa)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT delegado EMPTY>
<!ELEMENT optativa (teatro|informatica|cultura_clasica)>
<!ELEMENT teatro EMPTY>
<!ELEMENT informatica EMPTY>
<!ELEMENT cultura_clasica EMPTY>
```

Con este último DTD, los elementos nombre y apellidos pueden intercambiarse de orden.

Modificadores del número de apariciones de un elemento

Cuando describimos el contenido de un elemento, podemos añadir uno de los siguientes modificadores a las etiquetas que aparezcan:

Modificador	Significado
?	El elemento tanto puede aparecer (una vez), como no aparecer.
+	El elemento tiene que aparecer una o más veces.
*	El elemento puede aparecer 0 o más veces.

Como ejemplo, partiremos de este documento XML:

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

<!DOCTYPE persona SYSTEM "persona.dtd">

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

En el correspondiente DTD indicamos que una persona tiene un nombre y un apellido:

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

Si quisiéramos que una persona haya de tener un apellido al menos, y dos como máximo, en el DTD podemos usar el símbolo "?":

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

Podríamos querer que una persona pueda llevar uno o más apellidos (sin límite). Por ejemplo:

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

<!DOCTYPE persona SYSTEM "persona.dtd">

<persona>
      <nombre>Amadeo</nombre>

      <!-- Puede haber más de un apellido -->
      <apellido>Martínez</apellido>
      <apellido>Gómez</apellido>
      <apellido>Rojales</apellido>
</persona>
```

Esto lo indicaremos añadiendo un símbolo + al apellido en la primera línea del DTD:

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

Aunque, por otra parte, podríamos querer admitir, además, que una persona no tenga ningún apellido, entonces cambiaremos el símbolo + por *:

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

Según este último DTD una persona ha de tener un nombre y cualquier número de apellidos, incluído no tener apellido.

Contenido mixto

El contenido mixto consiste en texto normal con etiquetas intercaladas. En el siguiente ejemplo tenemos un documento XML para almacenar el contenido de una carta que envía una empresa a otra. Ciertas partes de la información han sido etiquetadas, supuestamente para poder extraer más fácilmente la información más adelante:

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

<!DOCTYPE carta SYSTEM "carta.dtd">

<carta>
      Apreciado <empresa>Ferros Puig</empresa>:

      Sr. <director>Manel Garcia</director>, le envío esta carta para
      comunicarle que le hemos enviado su pedido a la dirección que nos
      proporcionó.

      Atentamente, Ferretería, SA.
</carta>
```

La forma de codificar una etiqueta con contenido mixto es como #PCDATA seguido de los elementos que aparecen dentro, separados por el operador "|", y añadiendo al final de todo el grupo el modificador "*":

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

Actividad: Queremos crear una aplicación que permita al usuario llevar un diario de las cosas que le han ocurrido durante el día y de los libros que va leyendo. Necesitamos inventar un vocabulario XML para guardar toda esta información. Las características serán:

- La información que guardará este diario será el nombre del usuario, y los acontecimientos que escriba se estructurarán en entradas. Habrá al menos una entrada obligatoriamente.
- Cada entrada contendrá en primer lugar una fecha. Después se indicará si es día festivo, y aparecerá una única etiqueta <*contenido*> con lo acaecido ese día.
- A continuación, si el usuario ha leído algo ese día, aparecerá una entrada <*lectura*> para cada libro que haya estado leyendo.

Un ejemplo de XML para esto es:

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

<diario>

	<nombre>Luis Pérez</nombre>

	<entrada>
		<fecha>2023-10-17</fecha>
		<contenido>Querido diario...</contenido>
	</entrada>

	<entrada>
		<fecha>2023-10-16</fecha>
		<festivo />
		<contenido>Querido diario...</contenido>
		<lectura>El camino (Miguel Delibes)</lectura>
		<lectura>El nombre del viento (Patrick Rothfuss)</lectura>
	</entrada>

	<entrada>
		<fecha>2023-10-15</fecha>
		<festivo />
		<contenido>Querido diario...</contenido>
		<lectura>El nombre del viento (Patrick Rothfuss)</lectura>
	</entrada>

</diario>
```

Escribe el DTD correspondiente a este XML.

Atributos en DTD

En un archivo DTD debe haber una declaración para cada atributo de su correspondiente XML. En general la sintaxis es:

```
<!ATTLIST etiqueta nombre_atributo tipo_dato valor_por_defecto>
```

Por ejemplo, en el siguiente XML aparece una etiqueta *persona* con un atributo *nombre* y otro atributo apellidos:

```
<persona nombre="Carlos" apellidos="Sánchez">
      <edad>47</edad>
</persona>
```

La forma de codificar estos atributos en el DTD podría ser:

```
<!ELEMENT persona (edad)>
<!ATTLIST persona nombre CDATA #REQUIRED>
<!ATTLIST persona apellidos CDATA #REQUIRED>

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

Aquí estamos diciendo que los atributos nombre y apellidos pueden contener cualquier cadena de caracteres (CDATA) y son obligatorios (#REQUIRED).

Cuando un elemento tiene dos o más atributos, podemos expresarlo en un sólo ATTLIST:

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

El tipo de dato puede ser uno de los que se ven en la siguiente tabla:

Tipo de dato	Descripción
CDATA	(Character DATA) El valor son datos de tipo carácter, es decir, texto.
Enumerado	El valor puede ser uno de los pertenecientes a una lista de valores escritos entre paréntesis "()" y separados por el carácter "|".
NMTOKEN	El valor es una cadena de caracteres, pudiendo contener letras minúsculas, letras mayúsculas, números, puntos ".", guiones medios "-", guiones bajos "_" o el carácter dos puntos ":".
NMTOKENS	El valor puede contener uno o varios valores de tipo NMTOKEN separados por espacios en blanco.
ID	El valor es un identificador único.
IDREF	El valor es un identificador que tiene que existir en otro atributo ID del documento XML.
IDREFS	El valor es una lista de valores que existan en otros atributos ID del documento XML, separados por espacios en blanco.

El campo "valor por defecto" del atributo puede tomar uno de los valores de la siguiente tabla:

Valor por defecto	Descripción
valor	Valor por defecto que tomará el atributo, entre comillas.
#FIXED valor	El valor indicado es el que tendrá el atributo obligatoriamente.
#IMPLIED	No hay un valor por defecto, y el atributo es opcional.
#REQUIRED	No hay un valor por defecto, y el atributo es obligatorio.

Tipo de atributo CDATA

Cuando usamos CDATA como tipo de un atributo, éste podrá ser cualquier cadena de caracteres.

Ejemplo

El elemento empresa tiene un atributo obligatorio llamado *nombre*:

```
<empresa nombre="Microsoft Corporation">
```

Su DTD será:

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

Ejemplo

El elemento alumno tiene un atributo opcional llamado *clase*:

```
<alumno clase="2º Bachillerato A">
```

Su DTD será:

```
<!ATTLIST alumno clase CDATA #IMPLIED>
```

Tipo de atributo enumerado

Con un tipo enumerado podemos restringir los posibles valores del atributo a una lista. Para ello escribiremos los valores entre paréntesis separados por "|".

Ejemplo

El elemento *alumno* tiene dos atributos obligatorios, *nivel* y *grupo*. Se nos indica que el grupo sólo puede tomar como valor una letra mayúscula entre la 'A' y la 'H':

```
<alumno nivel="2º Bachillerato" grupo="A">
      <nombre>Marta</nombre>
      <apellidos>Lagos Andrés</apellidos>
</alumno>
```

Su DTD será:

```
<!ATTLIST alumno nivel CDATA #REQUIRED>
<!ATTLIST alumno grupo (A|B|C|D|E|F|G|H) #REQUIRED>
```

Ejemplo

Una variación del ejemplo anterior podría ser que quisiéramos dar un valor por defecto al atributo grupo. Entonces quedaría así:

```
<!ATTLIST alumno nivel CDATA #REQUIRED>
<!ATTLIST alumno grupo (A|B|C|D|E|F|G|H) "A">
```

Ahora, si un alumno no tiene grupo asignado, por defecto será del grupo A.

Ejemplo

Otra variación del ejemplo anterior podría ser que quisiéramos dar un valor fijo al atributo grupo. Entonces quedaría así:

```
<!ATTLIST alumno nivel CDATA #REQUIRED>
<!ATTLIST alumno grupo (A|B|C|D|E|F|G|H) #FIXED "A">
```

Ahora si un alumno tiene un grupo asignado distinto de "A", la validación dará error.

Tipo de atributo NMTOKEN

El valor de un atributo NMTOKEN puede ser cualquier cadena de caracteres formada por letras minúsculas, mayúsculas, números, puntos ".", guiones medios "-", guiones bajos "_" o el carácter dos puntos ":".

Ejemplo

El siguiente XML forma parte de un documento XML donde aparecen todos los coches de clientes de un taller:

```
<coche fecha_entrada="2023-10-01">
      <marca>Kia</marca>
      <modelo>Stonic</modelo>
</coche>
```

La fecha de entrada cumple los criterios para ser de tipo NMTOKEN:

```
<!ATTLIST coche fecha_entrada NMTOKEN #REQUIRED>
```

Tipo de atributo NMTOKENS

El valor de un atributo NMTOKENS puede ser una o más cadenas de tipo NMTOKEN.

Ejemplo

Supongamos que el taller del ejemplo anterior quiere guardar no sólo la fecha de compra de cada vehículo, sino también las fechas de las revisiones que ha ido pasando. El XML quedaría así:

```
<coche fecha_compra="2020-02-01"
      fechas_revisiones="2021-01-25 2022-01-28 2023-02-05">
      <marca>Kia</marca>
      <modelo>Stonic</modelo>
</coche>
```

Cada una de las fechas dentro de *fechas_revisiones* es un NMTOKEN válido. Por eso el tipo del atributo *fechas_revisiones* puede ser NMTOKENS:

```
<!ATTLIST coche fecha_compra NMTOKEN #REQUIRED>
<!ATTLIST coche fechas_revisiones NMTOKENS #REQUIRED>
```

Tipo de atributo ID

Un atributo ID debe cumplir que sea un identificador único, es decir, su valor sólo aparezca una vez en todo el documento. Además debe cumplir las mismas condiciones que un NMTOKEN y debe empezar por letra, guión bajo "_" o símbolo de dos puntos ":".

Ejemplo

Si añadimos a la información de cada coche del taller un atributo *matrícula*, éste va a ser único para cada coche, luego puede ser de tipo ID:

```
<coche matricula="MAT-7890ECC" fecha_compra="2020-02-01"
      fechas_revisiones="2021-01-25 2022-01-28 2023-02-05">
      <marca>Kia</marca>
      <modelo>Stonic</modelo>
</coche>

<coche matricula="MAT-8361DTG" fecha_compra="2019-06-10"
      fechas_revisiones="2020-06-01 2021-06-15">
      <marca>Volkswagen</marca>
      <modelo>Taigo</modelo>
</coche>
```

En tal caso definiremos el atributo así:

```
<!ATTLIST coche matricula ID #REQUIRED>
```

Tipo de atributo IDREF

El valor de un atributo IDREF debe ser un identificador que tiene que existir en otro atributo ID del documento XML.

Ejemplo

En el documento del taller también deben aparecer las repaciones que se van haciendo. En cada reparación se hará *referencia* a la matrícula del coche:

```
<taller>
	<coche matricula="MAT-7890ECC" fecha_compra="2020-02-01"
		fechas_revisiones="2021-01-25 2022-01-28 2023-02-05">
		<marca>Kia</marca>
		<modelo>Stonic</modelo>
	</coche>

	<coche matricula="MAT-8361DTG" fecha_compra="2019-06-10"
		fechas_revisiones="2020-06-01 2021-06-15">
		<marca>Volkswagen</marca>
		<modelo>Taigo</modelo>
	</coche>

	<reparacion mat_vehiculo="MAT-8361DTG">
		<concepto>Pinchazo rueda trasera izquiera</concepto>
		<precio>10</precio>
	</reparacion>
</taller>
```

Como los valores del atributo *mat_vehiculo* siempre van a aparecer como matrículas más arriba, y como *matrícula* es de tipo ID, podemos definir *mat_vehiculo* como de tipo IDREF:

```
<!ATTLIST reparacion mat_vehiculo IDREF #REQUIRED>
```

Tipo de atributo IDREFS

El valor de un atributo IDREFS debe ser uno o más valores que tienen que existir en otros atributos ID del documento XML.

Ejemplo

En este ejemplo tenemos un documento XML que almacena la cartelera de un festival de cine de los 80:

```
<cartelera>
      <actor cod="HF">Harrison Ford</actor>
      <actor cod="SW">Sigourney Weaver</actor>
      <actor cod="BP">Bill Paxton</actor>
      <actor cod="MB">Michael Biehn</actor>
      <actor cod="AS">Arnold Schwarzenegger</actor>

      <pelicula actores="HF">En busca del arca perdida</pelicula>
      <pelicula actores="SW BP MB">Aliens, el regreso</pelicula>
      <pelicula actores="AS MB">Terminator</pelicula>
</cartelera>
```

Cada actor tiene un código distinto (del tipo ID), y cada película tiene un atributo *actores* cuyo valor son los códigos de los actores que aparecen. Como estos valores son referencias de IDs, podemos declarar *actores* como atributo IDREFS.

```
<!ATTLIST actor cod ID #IMPLIED>
<!ATTLIST pelicula actores IDREFS #IMPLIED>
```

Otras formas de asociar DTDs a un documento

DTD interna

Podemos escribir las reglas DTD directamente dentro del DOCTYPE del documento XML. En lugar de la línea:

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

Escribiremos:

```
<!DOCTYPE persona []>
```

Y dentro de los corchetes escribiremos las reglas que irían en el archivo DTD. Por ejemplo:

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

<!DOCTYPE persona [
      <!ELEMENT persona (edad)>
      <!ATTLIST persona nombre CDATA #REQUIRED>
      <!ATTLIST persona apellidos CDATA #REQUIRED>
      <!ELEMENT edad (#PCDATA)>
]>

<persona nombre="Carlos" apellidos="Sánchez">
      <edad>47</edad>
</persona>
```

DTD pública

Las DTD públicas se reservan para organismos de estandarización, oficiales o no. El esquema de definición de una DTD pública es:

```
<!DOCTYPE etiqueta_raíz PUBLIC "FPI" "dirección_archivo.dtd">
```

El FPI (Formal Public Identifier) son cuatro cadenas separadas por el símbolo "//". Éstas son:

- Símbolo: Puede aparecer "-" si se trata de un propietario no registrado, "+" si se trata de un propietario registrado, o "ISO" si se trata de un identificador asignado por la organización ISO.
- Nombre del responsable: Organización o persona responsable.
- Identificador único del documento descrito.
- Idioma.

Ejemplo

Cuando se creaba una página web con el estándar XHTML 1.1, los documentos iban precedidos con este DOCTYPE:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

El FPI de este DOCTYPE es `-//W3C//DTD XHTML 1.1//EN` y significa:

- Símbolo: (-) Propietario no registrado (curioso que el W3C no lo esté)
- Responsable: W3C
- Identificador del documento: DTD XHTML 1.1
- Idioma: EN (inglés)

La dirección donde está guardado este DTD es http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd.

Limitaciones de los DTD

DTD adolece de algunas limitaciones:

- No dispone de tipos de datos. No podemos indicar que dentro de un elemento hayamos de poner una fecha, un número decimal, una cadena de caracteres, etc.
- No soporta espacios de nombres.
- No permite expresiones no deterministas. Eclipse suele ignorar este fallo, pero si usamos *xmllint* a veces nos dirá que el DTD no es válido por este motivo.

DTD determinista

Un DTD es determinista cuando al leer un documento, el parser sólo necesita mirar al siguiente elemento para saber en qué punto se encuentra en la expresión dentro de paréntesis.

Ejemplo

El siguiente XML contiene en su DTD una regla no determinista (la de mascota):

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

<!DOCTYPE mascota [
      <!ELEMENT mascota ((propietario, perro) | (propietario, gato))>
      <!ELEMENT propietario (#PCDATA)>
      <!ELEMENT perro (#PCDATA)>
      <!ELEMENT gato (#PCDATA)>
]>

<mascota>
      <propietario>Carlos Pérez</propietario>
      <perro>Pipo</perro>
</mascota>
```

Cuando el parser llega a la etiqueta mascota, y ve que después viene una etiqueta propietario, no sabe si está en el primer caso (propietario, perro) o en el segundo (propietario, gato).

Ejemplo

Podemos corregir el ejemplo anterior cambiando la expresión de mascota así:

```
<!ELEMENT mascota (propietario, (perro | gato))>
```

Significa lo mismo pero no hay ambigüedad.