Tema 4. XML.

Tema 4. XML.

1. El lenguaje XML.

Como ya comentamos en el primer capítulo, XML (*eXtensible Markup Language* o Lenguaje de Etiquetado Extensible) es un lenguaje de marcado que posee una Recomendación del World Wide Web Consortium (http://www.w3.org/TR/REC-xml/). Recordemos que:

- XML es un metalenguaje de etiquetas extensible que permite **definir la gramática de lenguajes específicos**. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.
- XML no se creó sólo para su aplicación en Internet, sino que se propone como un **estándar para el intercambio de información estructurada** entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y en casi cualquier cosa imaginable.

Con la tecnología XML se pretende dar una solución al problema de cómo expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen, a su vez, de otras partes. De esta forma, se consigue un árbol estructurando con pedazos de información. Estas partes se denominan *elementos*, y se las señala o referencia mediante *etiquetas*.

Una **etiqueta** consiste en una marca hecha en el documento, que señala una porción de éste como un elemento o pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma <nombre>, donde *nombre* es el nombre del elemento que se está señalando.

1.1. Objetivos de diseño de XML.

Antes de continuar, es recomendable comentar los objetivos que el equipo de expertos que desarrolló XML se puso como meta. Esto nos permitirá entender mejor las características y funcionalidades de XML.

1. XML debe ser directamente utilizable sobre Internet de forma que los documentos XML puedan ser transferidos entre diferentes sistemas a través de este medio. Al utilizar formato texto, esta transmisión por red es sencilla.
2. XML debe soportar una amplia variedad de aplicaciones de forma que la norma no esté vinculada a ninguna tecnología o fabricante en concreto.
3. XML debe ser compatible con SGML. Esto se logró haciendo que XML fuese un subconjunto de SGML.
4. Debe ser fácil la escritura de programas que procesen documentos XML. Si se quería que XML fuera compatible con otras aplicaciones, era necesario que el análisis de los documentos en formato XML fuera sencillo.
5. El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero. De esta forma se consiguen analizadores más simples y estandarizados.
6. Los documentos XML deben ser legibles por los usuarios y razonablemente claros. De esta forma, se combina la eficacia de los procesos automáticos con la posibilidad de que las personas podamos obtener información de un documento XML.
7. El diseño de XML debe ser formal, conciso y preparado rápidamente. Dicho de otra forma, la representación de los datos se debe poder hacer en poco tiempo.
8. XML debe ser simple pero perfectamente formalizado.
9. Los documentos XML deben ser fáciles de crear. Con un simple editor plano se pueden elaborar documentos XML, aunque existen aplicaciones muy útiles y eficaces que ayudan.
10. La brevedad en las marcas XML es de mínima importancia. Se prefiere que cada etiqueta usada tenga su correspondiente etiqueta de cierre, aunque haya que escribir más. Por ejemplo, HTML no es tan estricto en este sentido, pero ello suele generar problemas en los exploradores ya que analizan de forma distinta lo que no está perfectamente cualificado.

1.2. Estructura básica de un documento XML.

Empecemos con un ejemplo para indicar la sintaxis y la estructura de un documento XML. Supongamos que queremos registrar en un documento XML los datos de una película, por ejemplo: "*Mar adentro*" dirigida por "*Alejandro Amenábar*" y protagonizada por "*Javier Bardem*". Simplemente abrimos un nuevo documento con un editor de texto "plano" o "ASCII" y escribimos:

```
<?xml version="1.0"?>
<pelicula>
    Mar adentro
    <director>Alejandro Amenabar</director>
    <actores>
      <actor>Javier Bardem</actor>
    </actores>
</pelicula>
```

El archivo lo guardaremos con el nombre *mar.xml*. En realidad, la extensión del archivo puede ser cualquiera, pero es recomendable usar *xml* para identificar más fácilmente su tipo, tanto por nosotros como por las aplicaciones que lo usan.

El documento se compone de dos partes. La primera es el prólogo en donde, entre otras cosas, se indica que es un documento XML que se ajusta a una versión específica de la norma. La segunda parte es el cuerpo del documento. El inicio del cuerpo lo determina una etiqueta, en este caso <pelicula>, y el final del cuerpo la etiqueta de cierre correspondiente, es decir, </pelicula>.

Dentro del cuerpo se pueden colocar tantas etiquetas de apertura y cierre como se necesiten, siempre que estén correctamente anidadas. Por ejemplo, el siguiente fragmento sería incorrecto:

```
<actores>
   <actor>Javier Bardem</actores>
</actor>
```

Sin esta simple regla un programa no podría leer de forma automática el documento y extraer la información que contiene.

Podríamos incluir elementos vacíos o, dicho de otra forma, podría ser que entre una etiqueta de apertura y su correspondiente de cierre no hubiera nada. Imaginemos que no conocemos ningún actor de la película, en este caso podríamos colocar la etiqueta actores de la forma:

```
<actores></actores>
```

Los navegadores exploradores o clientes web actuales, además de leer documentos de hipertexto, pueden leer documentos XML. La visualización de una página XML en un cliente web tiene muchas ventajas frente a la que proporciona un editor de texto plano:

1. Se resalta la sintaxis. El prólogo se ve de un color, la etiquetas de otro y el contenido de éstas de otro. Esto permite diferenciar unas partes del documento de otras facilmente.
2. Se permite la expansión y contracción de los elementos. Si se hace clic con el ratón sobre un elemento que presente un guión a su izquierda, el elemento se contraerá ocultando los elementos que aparezcan a continuación de él y el guión cambia al símbolo (+). Si se hace clic sobre él se volverá a expandir.
3. Se realiza una comprobación del documento. Si el documento está mal construido, el navegador nos avisará presentando un mensaje de error, además de cómo solucionar el problema. Errores típicos son los que se producen cuando las etiquetas no están correctamente anidadas, o el identificador de la etiqueta de apertura es dintinto a la correspondiente de cierre.

1.2.1. Analizadores XML (*Parsers*).

Si seguimos las reglas especificadas por XML podremos de una forma muy simple acceder a la información. Existen programas denominados analizadores que son capaces de **leer la sintaxis de XML y obtener la información almacenada en el documento**. Incluso podemos usar dichos analizadores dentro de nuestros propios programas, de forma que nuestra aplicación no necesita trabajar directamente con el documento XML, y que sea el analizador el que nos la proporcione.

Los *parsers* trabajan independientemente del documento XML del que obtienen los datos: no importa que etiquetas tenga, o el idioma usado. Lo importantes es que el documento cumpla las reglas XML.

1.3. Ejemplo básico.

Para comprender mejor las posibilidades de un documento XML, vamos a completar el ejemplo anterior añadiendo la posibilidad de cambiar el aspecto que presentarían los datos cuando lo abrimos con un explorador web.

Vamos a crear un archivo que contenga la definición de los estilos a aplicar al documento. Este archivo será una hoja de estilos en cascada o CSS y se llamará *estilo_xml.css*. Su contenido es:

```
pelicula
{
	display:block;
	font-size:2em;
	margin-top:25px;
	margin-left:15px;
}
director
{
	font-size:0.7em;
	display:block;
	top:15px;
	position:relative;
}
director:before
{
	content:"Director: ";
}
actor
{
	font-size:0.7em;
	display:block;
	top:15px;
	position:relative;
}
actor:before
{
	content:"Actor: ";

}
```

A continuación indicaremos en el documento XML que se debe aplicar el estilo anterior cuando un cliente web lo presente en pantalla. Para ello, se añade como segunda línea del prólogo lo siguiente:

```
<?xml-stylesheet type="text/css" href="estilo_xml.css" media="screen" ?>
```

Si ahora visualizamos el documento XML con un cliente web, se presentará de una forma completamente distinta a como se veía anteriormente. El formato de presentación se ha conseguido gracias a la hoja de estilos.

Mar adentro

Director: Alejandro Abenamar
Actor: Javier Bardem

Si quisiéramos una presentación distinta de los datos, o bien quisiéramos presentarlos en un dispositivo diferente (pantalla de un móvil, salida impresa, etc.) sólo bastaría definir una hoja de estilos adecuada al formato de presentación que se deseemos.

La ventaja de esta forma de actuar es que si una aplicación debe obtener de forma automática los datos sobre la película, lo hará como siempre, sin embargo, un cliente Web es capaz de interpretar la línea de prólogo que indica el aspecto a presentar de forma que a las personas nos sea más fácil interpretar los datos.

No obstante, persiste un problema sobre los archivos CSS: cada explorador interpreta a su manera algunas de las reglas de las hojas de estlilos. Basta probar sobre nuestro documento XML cómo lo hacen Internet Explorer y FireFox…

2. Documentos bien formados.

En este apartado trataremos las reglas sintácticas por las que se rige XML. Estas reglas son importantes y deben ser seguidas por los que quieran usar este formato. La finalidad es conseguir un documento *bien formado*.

Veremos las partes de que se compone un documento XML: prólogo, cuerpo y opcionalmente epílogo. Cada una de estas partes se constituye de otras más pequeñas. Dentro del cuerpo, las partes constituyentes se denominan elementos.

2.1. Partes de un documento XML.

Es importante recordar que XML es un lenguaje de marcas, y que éstas están delimitadas por los símbolos "<" y ">". Lo que se encuentre entre estos dos símbolos es una marca o etiqueta. Utilizaremos el ejemplo del principio del tema para que nos sirva de ejemplo.

```
<?xml version="1.0"?>
<pelicula>
    Mar adentro
    <director>Alejandro Amenabar</director>
    <actores>
        <actor>Javier Bardem</actor>
    </actores>
</pelicula>
```

2.1.1. Prólogo.

Todo documento XML debe comenzar con un prólogo en donde se indique el tipo de documento, la norma que va a cumplir y otras informaciones. En el ejemplo, el prólogo es:

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

La etiqueta es diferente a las del resto del documento pues tiene la forma `<?...?>`. Este formato indica que la etiqueta es una **instrucción de proceso**. Dicha instrucción no forma parte del contenido del documento y se utiliza para dar información a las aplicaciones que procesan el documento.

En este caso, se indica que el documento se ciñe totalmente a la norma establecida en la versión 1.0. El atributo *version* es el que permite indicarlo. La última versión es la 1.1. Básicamente, es como la 1.0 con mejoras a la hora de interpretar los caracteres Unicode usados para nombrar los elementos.

Si no se especifica un juego de caracteres para el documento, se supone que se usará UTF-8. Para indicar una codificación concreta se escribe el atributo *encoding*, seguido del signo (=) y, entre comillas, el nombre del código a usar: (UTF-8, UTF-16, ISO-8859-1, etc.). Por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Hay un tercer atributo opcional, *standalone*, cuyos posibles valores son *yes* o *no*. Con "*yes*" se indica que el documento es autosuficiente y que no depende de ningún otro archivo. El valor "*no*" indica que puede depender de una DTD externa o de un esquema que permita comprobar la validez de los datos de un documento XML. En los siguientes temas estudiaremos en profundidad las DTD y los esquemas. Este atributo no tiene valor por defecto, por lo que si no se especifica, el documento XML no puede validarse.

```
<?xml version="1.0" encoding="ISO-8859-1" estandalone="yes"?>
```

En el ejemplo básico que vimos anteriormente, usábamos una hoja de estilo en cascada y que contenía la siguiente instrucción de proceso:

```
<?xml-stylesheet type="text/css" href="estilo_xml.css"? media="screen" >
```

Esta instrucción indica que existe una hoja de estilo que se puede aplicar al documento. Con el atributo *type* se especifica el tipo de hoja de estilo, en este caso, una hoja de estilo en cascada CSS. También se puede usar hojas de estilo tipo XSLT para realizar transformación de los datos y presentarlos de diferentes formas. El atributo *href* señala la ubicación y archivo en que se encuentra la hoja de estilo. El atributo *media* permite indicar en qué tipo de dispositivo de salida se aplicará la hoja de estilos

Estas dos instrucciones de proceso aparecen siempre en el prólogo, pero existen otras instrucciones de proceso que pueden aparecer en el cuerpo del documento, pero su uso es muy poco habitual. Se usan normalmente para incluir instrucciones para una aplicación específica indicando que se realice algún proceso. El prólogo puede contener otras informaciones, pero las veremos cuando sea oportuno.

2.1.2. Cuerpo.

Los datos que almacena un documento XML se organizan mediante elementos y están ubicados en lo que se conoce como cuerpo. En el caso del ejemplo, se trata de todo lo que se encuentra entre <pelicula> y </pelicula> incluidas las dos etiquetas. Este elemento se conoce como el elemento raíz del documento y debe ser único.

2.1.3. Epílogo.

El epílogo es opcional y se sitúa a continuación del cuerpo del documento. Puede contener instrucciones de procesamiento, pero no como las de tipo de documento, o declaraciones XML. Realmente su uso es muy poco habitual.

2.2. Elementos.

Un elemento se identifica por medio de una etiqueta y puede ir acompañado de uno o varios atributos. Además, cada elemento normalmente tiene un contenido.

La estructura de un elemento es:

```
<etiqueta atributo1="valor_atributo1" atributo2="valor_atributo2" ... >contenido</etiqueta>
```

Para comprender bien XML es necesario conocer las reglas para los elementos:

- Cada etiqueta de inicio debe tener su etiqueta de cierre correspondiente.
- Las etiquetas no de deben superponer.
- Los documentos XML sólo pueden tener un único elemento raíz que no puede duplicarse y que es el primero que se abre y el último que se cierra.
- Los nombres de los elementos deben cumplir ciertas convenciones de nomenclatura.
  - Pueden contener letras, números y tres signos de puntuación: guión (-), guión bajo (_) y punto (.).
  - No pueden empezar por un número o un signo de puntuación (excepto el guión bajo).
  - No pueden empezar por las letras «xml» (ni cualquier combinación de caso, como `XML`, `Xml`,etc.).
  - No pueden contener espacios.
  - Los atributos siempre deben ir entrecomillados.
- XML establece diferencias entre mayúsculas y minúsculas.
- XML mantiene los espacios en blanco del texto.

El orden de los elementos es muy importante, en el sentido de que deben estar estructurados jerárquicamente en forma de árbol, correctamente anidados y sin superponerse o solaparse entre ellos. Hay que tener en cuenta, también, que sólo puede haber un elemento raíz al que estén contenidos todos los demás; por lo tanto, habrá una etiqueta contenedora que englobe a todo el contenido y que sea única.

Puede haber etiquetas vacías, es decir sin contenido. En este caso, como no engloban nada en vez de la forma:

```
<etiqueta></etiqueta>
```

se puede utilizar la notación:

```
<etiqueta/>
```

A lo mejor nos preguntamos para qué sirve un elemento vacío. Algunas veces, la sola presencia del elemento ya aporta información al documento. Otras veces puede que la inclusión de un elemento con ese nombre sea obligatoria, pero no sea obligatorio el que tenga contenido.

En XML, no es lo que mismo <actor> que <Actor>. Es decir, se distingue entre mayúsculas y minúsculas. Por ello, se dice que XML es *sensible a mayúsculas.* Para evitar errores lo mejor es seguir una norma; por ejemplo, escribir la etiquetas siempre en minúsculas o mayúsculas.

En XML se pueden usar los espacios en blanco. Se considera espacio en blanco no sólo el carácter generado con la barra espaciadora sino, también, el tabulador, el retorno de carro y el salto de línea. Los espacios en blanco nos permiten a las personas leer y estructurar de forma más fácil el documento. Para este fin, es conveniente seguir estas reglas:

- Colocar cada elemento en una línea del documento.
- Usar indentación mediante tabuladores para situar unos elementos dentro de otros.
- Usar espacios para conseguir una buena lectura.

2.2.1. Atributos.

Las etiquetas pueden aprovecharse para incluir otros datos usando atributos. Por así decirlo, los atributos permiten añadir propiedades a un elemento, algo así como adjetivarlo.

Un atributo está formado por el nombre del mismo y el valor que toma entrecomillado (comillas simples o dobles), separados por el signo de igualdad (=). Si un elemento tiene un atributo, es obligatorio asignarle un valor. Los atributos sólo se pueden escribir en las etiquetas de apertura.

Normalmente, el uso de atributos permite diferenciar elementos del mismo tipo. Volviendo al ejemplo de la película, supongamos que hemos encontrado los nombres de otros actores que aparecen en la película y queremos añadirlos al documento XML. Con atributos podríamos por ejemplo diferenciar entre los actores que son protagonistas de los secundarios:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="estilillo.css"?>
<pelicula>
    Mar adentro
   <director>Alejandro Amenábar</director>
   <actores>
      <actor papel="protagonista">Javier Bardem</actor>
      <actor papel="secundario">Celso Bugallo</actor>
    </actores>
</pelicula>
```

Es recomendable que un atributo de una etiqueta no tenga la misma categoría que las etiquetas contenidas, pues si fuera de la misma categoría sería conveniente crear otra etiqueta contenida. Así, por ejemplo, tendría más sentido añadir el atributo *idioma* a la etiqueta película, pero no tanto un atributo *vestuario*. Este atributo sería mejor sustituirlo por otra etiqueta que quedara contenida en <pelicula>.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="estilillo.css"?>
<pelicula idioma="español">
    Mar adentro
    <director>Alejandro Amenábar</director>
    <actores>
       <actor papel="protagonista">Javier Bardem</actor>
       <actor papel="secundario">Celso Bugallo</actor>
    </actores>
    <vestuario>Sonia Grande</vestuario>
</pelicula>
```

Si una etiqueta dispone de varios atributos, el orden en que aparecen es irrelevante ya que los atributos no se organizan de forma jerárquica.

La experiencia dice que no hay que abusar del uso de atributos, ya que a la hora de procesar y obtener los datos es más eficaz si éstos se presentan como contenidos de elementos que como valores de atributos. Es habitual usar un atributo cuando se necesita identificar un elemento mediante un valor clave. Por ejemplo, si disponemos un documento XML con información de varias películas, podríamos añadir un atributo *código* al elemento <pelicula> de forma de que cada película tuviera un código distinto:

```
<videoteca>
    <pelicula codigo="1">
    . . .
    </pelicula>
    <pelicula codigo="2">
    . . .
    </pelicula>
</videoteca>
```

2.2.2. Entidades predefinidas.

Ya sabemos que como contenido de un elemento, podemos incluir otros elementos o texto independiente. Dicho texto no tiene que limitarse a una línea. Por ejemplo:

```
<critica>
   Un lento, plácido y doloroso viaje a la muerte a través de un hermoso, cálido y poético canto a la vida, que plasma
   el final del periplo de este marinero gallego que amaba tanto la vida que deseaba la muerte
   por no prolongarla indignamente.
</critica>
```

Existen algunas limitaciones sobre los caracteres que pueden ser escritos como contenido de un elemento. Los caracteres &, < , >, ', y " tienen un significado especial, por lo que, para poder escribirlos directamente se deben utilizar las llamadas *entidades* o *referencias de entidad*. Las entidades comienzan con "&" y acaban con ";". El listado siguiente muestra la entidad que sustituye al símbolo que le precede.

```
&     &amp;
<     &lt;
>     &gt;
'     &apos;
"     &quot;
```

Por ejemplo, si una película se titulara *Black & White* deberíamos escribirlo como:

```
<pelicula idioma="inglés">
   Black &amp; White
   . . .
</pelicula>
```

Por otro lado, cualquier carácter se puede codificar usando *referencia de caracteres*. Se expresan como cadenas de la forma **&#nnn;** ó **&#xnnn**, donde nnn es el valor Unicode expresado en decimal o en hexadecimal, según se use un formato u otro. Así, para el símbolo de *copyright* © su referencias serían &#169 ó &#xA9.

2.2.3. Secciones CDATA.

Una vez escrito un documento XML, lo normal es que se procese con alguna herramienta software para extraer los datos. Antes de la extracción de datos suele usarse un analizador para comprobar que el documento está bien construido. Además, el analizador permite acceder a los diferentes elementos que componen el documento XML.

Puede ocurrir que sea necesario incluir en el documento partes que no deban ser procesadas por el analizador. Con este fin se utilizan las secciones CDATA (*Character DATA*) que permiten especificar datos utilizando cualquier carácter, especial o no, sin que se interprete como marcado.

Por ejemplo, a veces, hay que incluir trozos de código en un lenguaje de programación o partes que tienen muchos caracteres especiales y que habría que sustituir por entidades quedando el documento muy poco claro. En estos casos, lo mejor es usar una sección CDATA e incluir dichos trozos.

El elemento CDATA comienza con <![CDATA[y termina con]]> . Todo lo incluido en su interior no se interpretará por el analizador. (Todo no; lógicamente la cadena de cierre]]> no puede ser incluida).

Por ejemplo, supongamos un documento XML con una etiqueta <codigo> que debe tener como contenido la definición de una función de un lenguaje de programación, como por ejemplo:

```
function saludo(int a, int b)
{
      if(0 < a && a < b)
          alert("Hola");
}
```

Si usamos CDATA nos quedaría de la forma:

```
<codigo>
   <![CDATA[
      function saludo(int a, int b)
      {
           if(0 < a && a < b)
               alert("Hola");
      }
   ]]>
</codigo>
```

Si no, tendríamos que usar entidades, quedando el documento mucho menos legible:

```
<codigo>
    function saludo(int a, int b)
    {
        if(0 &lt; a &amp;&amp; a &lt; b)
          alert(&quot;Hola&quot;);
    }
</codigo>
```

2.2.4. Comentarios.

A veces es necesario incluso conveniente incluir datos dentro de un documento XML que nos sirvan como referencia, pero que no formen parte de los datos del documento XML. Por ejemplo: quién es el autor del documento, la fecha de creación, o lo que consideremos conveniente.

Estas informaciones que sirve de documentación para nosotros o para otras personas, llamadas comentarios, se puede incluir usando un tipo especial de etiquetas. Para abrir un comentario se usa la secuencia de caracteres <!--, y para cerrarlo -->. Lo que quede dentro quedará oculto para el analizador.

Se pueden usar tantos comentarios como se desee, pero no se pueden anidar unos dentro de otros.

Otro uso de los comentarios es ocultar parte del documento para el analizador. Por ejemplo:

```
<!-- Autor: Pepe Cohete
      Fecha: Iaunarius MMXVII -->
<pelicula idioma="español">
     Mar adentro
    <director>Alejandro Amenábar</director>
    <actores>
        <actor papel="protagonista">Javier Bardem</actor>
        <!-- No listar más de cinco actores secundarios -->
        <actor papel="secundario">Celso Bugallo</actor>
    </actores>
</pelicula>
```

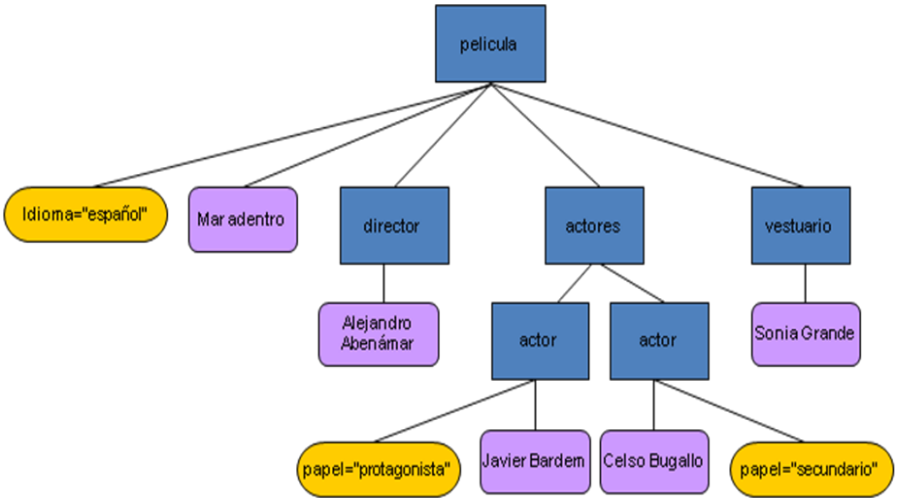
2.3. Nodos.

Técnicamente, un documento XML se estructura en forma de árbol. En informática, un árbol es una estructura de datos compuesta de nodos conectados entre sí, donde en su nivel superior existe un nodo llamado raíz que conecta con sus nodos hijos o descendientes. Éstos, a su vez, se pueden conectar con sus propios nodos hijos y, así, sucesivamente.

Una propiedad importante de un árbol es que cada nodo y sus hijos forman también un árbol. Así, un árbol es una estructura jerárquica de árboles en la cual cada árbol está constituido por pequeños árboles.

En un documento XML toda información constituye un nodo; así hay nodos elemento, nodos para los atributos, nodos para el contenido textual, etc.

En esta estructura de nodos, siempre debe haber un único nodo raíz del cual descenderán los posibles nodos hijos. Con nuestro documento XML de ejemplo podemos hacer un grafo con la jerarquía de nodos:



2.4. Espacios de nombres.

Como XML es extensible, nos permite usar las etiquetas que queramos, y puede ocurrir que utilicemos una misma marca para dos elementos distintos. Por ejemplo, supongamos un documento XML donde se registran datos de una serie de libros conteniendo un elemento *precio* para indicar el precio del libro. Por otro lado tenemos otro documento XML donde registramos los datos de una serie de CDs de música y que también dispone de un elemento llamado *precio* con el mismo fin. Si una aplicación tiene que usar conjuntamente los datos de los dos documentos, a priori, no podrá distinguir cuando el elemento *precio* se refiere al de un libro o al de un CD musical.

El uso de espacios de nombres (*namespaces*) permite diferenciar entre etiquetas con el mismo nombre utilizando prefijos. De esta forma se pueden evitar conflictos de nombres.

Podemos pensar en un espacio de nombres como una forma de agrupar elementos que permite distinguirlos de otros elementos que pertenezcan a otro espacio de nombres.

La forma de definir un espacio de nombres es mediante el atributo xmlns de la forma:

xmlns:prefijo = nombre_namespace

Donde nombre_namespace es el nombre del espacio de nombres, y prefijo es el identificador por el que nos referimos en el documento actual, usándolo como prefijo, a los elementos del espacio de nombres correspondiente.

El nombre o identificador del espacio de nombres puede ser cualquier cadena de caracteres, por ejemplo:

xmlns:book="misLibros"

Para usar un espacio de nombres se añade el atributo xmlns a los elementos que queremos incluir en dicho espacio de nombres. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<libro xmlns:book="misLibros">
   <book:titulo>Aprenda XML en 10 minutos</book:titulo>
   <book:autor>Pepe Cohete</book:autor>
   <book:precio>19.99</book:precio>
</libro>
```

En el elemento libro, se define un espacio de nombres llamado **misLibros** al cual se hará referencia mediante el prefijo book:. Todas las etiquetas, pertenecen al espacio de nombres **misLibros** ya que se declaran con el prefijo book:.

Como ya hemos comentado, a la hora de combinar datos XML de distintas fuentes pueden producirse conflictos en los nombres de los elementos y/o atributos. Supongamos que disponemos del siguiente fichero XML con nuestra colección de libros:

```
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
   <item>
      <titulo>Cocinar sin cocina</titulo>
      <autor>Rosa Moreno</autor>
      <precio>9.49</precio>
   </item>
   <item>
      <titulo>Construya su casa en 10 minutos</titulo>
      <autor>Luis Chapucero</autor>
      <precio>27.75</precio>
   </item>
</coleccion>
```

y del siguiente fichero con nuestra colección de discos:

```
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
   <item>
      <titulo>Violin Concerto in D</titulo>
      <compositor>Beethoven</compositor>
      <precio>14.95</precio>
   </item>
```

```
        <item>
            <titulo>Violin Concertos Numbers 1, 2, and 3</titulo>
            <compositor>Mozart</compositor>
            <precio>16.49</precio>
        </item>
    </coleccion>
```

Supongamos que queremos combinar estos documentos en uno solo, y además, gestionar el resultante con una sola aplicación. El problema surge al haber elementos repetidos (con el mismo nombre). Por ejemplo, ¿cómo hacer una lista de todos los libros?, ¿cómo calcular el precio medio de los CDs?.

El mecanismo de los espacios de nombres facilita esta tarea: basta con definir dos espacios de nombres en el mismo documento para diferenciar cada elemento. En el documento combinado, cada elemento de un libro (item, titulo…) se asigna al espacio de nombres *book* y cada uno de los elementos de un CD (item, titulo…) se asigna al espacio de nombres *cd*. Así el documento combinado sería:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Fichero: Coleccion.xml -->
<coleccion  xmlns:book="misLibros"  xmlns:cd="miMusica">
    <book:item>
            <book:titulo>Cocinar sin cocina</book:titulo>
            <book:autor>Rosa Moreno</book:autor>
            <book:precio>9.49</book:precio>
    </book:item>
    <cd:item>
            <cd:titulo>Violin Concerto in D</cd:titulo>
            <cd:compositor>Beethoven</cd:compositor>
            <cd:precio>14.95</cd:precio>
    </cd:item>
    <book:item>
            <book:titulo>Construya su casa en 10 minutos</book:titulo>
            <book:autor>Luis Chapucero</book:autor>
            <book:precio>27.75</book:precio>
    </book:item>
    <cd:item>
            <cd:titulo>Violin Concertos Numbers 1, 2, and 3</cd:titulo>
            <cd:compositor>Mozart</cd:compositor>
            <cd:precio>16.49</cd:precio>
    </cd:item>
</coleccion>
```

Los espacios de nombres se usan cuando no hay otro remedio, o cuando hay que combinar conjuntos de etiquetas XML de diferentes procedencias. En todo caso, conviene especificar un espacio de nombres para que las etiquetas se puedan identificar fácilmente dentro de un documento.

2.4.1. Uso de URI en los espacios de nombres.

Vimos que la forma de definir un espacio de nombres es:

xmlns:prefijo = nombre namespace

Donde podemos elegir libremente el nombre del espacio de nombres. Pero ¿qué ocurre cuando dos documentos XML diferentes usan, casualmente, el mismo espacio de nombres y contienen elementos con el mismo nombre que deben procesarse conjuntamente? Aunque la solución no es definitiva, se recomienda utilizar como identificador para un espacio de nombres un **URI**.

Un **URI** (*Uniform Resource Identifier* o Identificador Uniforme de Recursos) es una cadena de caracteres que identifica inequívocamente un recurso (puede ser un servicio, una página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente, estos recursos son accesibles en una red o sistema. Los URI pueden ser de dos tipos **URL** o **URN**.

Un **URL (***Uniform Resource Locator* o Localizador Uniforme de Recursos), es una secuencia de caracteres de acuerdo a un formato estándar, que se usa para nombrar recursos en Internet con objeto de localizarlos o identificarlos, como documentos textuales, imágenes, videos, presentaciones digitales, etc. Por ejemplo: "http://www.example.com/bookinfo/".

Un **URN (***Uniform Resource Name* o Nombre Uniforme de Recursos), es una secuencia de caracteres que permite identificar un recurso pero sin expresar su ubicación o como acceder a él. Por ejemplo: "urn:mybookstuff.org:bookinfo".

Lo que se pretende realmente usando un URI como nombre para un espacio de nombres es un identificador único para dicho espacio de nombres ya que un URI expresa un identificador único para un recurso, aunque podría usarse cualquier cadena de caracteres. Por cierto, cuando se usa un URL no tiene porqué hacer referencia a un servidor activo (Ningún analizador se va a entretener en comprobarlo…).

2.4.2. Namespaces por defecto.

Un espacio de nombres XML declarado sin prefijo se convierte en el espacio de nombres por defecto para dicho elemento y para todos los subelementos del elemento en el que aparece la declaración. En esta situación, todos los subelementos que aparezcan sin prefijo harán referencia al espacio de nombres por defecto. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<libro xmlns ="www.PepeCohete.com/libros">
    <titulo>XML en 10 minutos</titulo>
    <autor>Pepe Cohete</autor>
    <precio>19.99<precio>
</libro>
```

En el siguiente ejemplo el espacio de nombres por defecto es www.atrapiedra.com/MisLibros ya que no se indica ningún prefijo en su declaración. Por lo tanto, los elementos no cualificados (esto es, sin prefijo relativo a un *namespace*) se consideran pertenecientes al *namespace* por defecto:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Fichero: Coleccion2.xml -->
<coleccion xmlns="www.atrapiedra.com/MisLibros"  xmlns:cd="www.atrapiedra.com/MiMusica">
    <item>
            <titulo>Cocinar sin cocina</titulo>
            <autor>Rosa Moreno</autor>
            <precio>$9.49</precio>
    </item>
    <cd:item>
            <cd:titulo>Violin Concerto in D</cd:titulo>
            <cd:compositor>Beethoven</cd:compositor>
            <cd:precio>$14.95</cd:precio>
    </cd:item>
    <item>
            <titulo>Construya su casa en 10 minutos</titulo>
            <autor>Luis Chapucero</autor>
            <precio>$27.75</precio>
    </item>
    <cd:item>
            <cd:titulo>Violin Concertos Numbers 1, 2, and 3</cd:titulo>
            <cd:compositor>Mozart</cd:compositor>
            <cd:precio>$16.49</cd:precio>
    </cd:item>
</coleccion>
```

2.4.3. Ámbito de los namespaces.

Los elementos no cualificados (esto es, sin prefijo relativo a un *namespace*) se consideran pertenecientes al *namespace* por defecto más interno. En el siguiente ejemplo, <libro>, <titulo> y <autor> corresponden al espacio de nombres por defecto (el definido en <libro>), mientras que <editorial> y <nombre> pertenecen al espacio de nombres más interno:

```
<?xml version="1.0" encoding="utf-8"?>
<libro xmlns="www.atrapiedra.com/MisLibros">
    <titulo>Domine a su vecino</titulo>
    <autor>Indalecio Incordio</autor>
    <editorial xmlns="urn:publicistas:publicos">
        <nombre>Ediciones Presiona</nombre>
    </editorial>
</libro>
```

3. Herramientas.

Existen múltiples herramientas para analizar o procesar documentos XML (*parser*) y comprobar si están o no bien formados. Otras herramientas además comprueban que el documento XML sea válido.

La validación no es lo mismo que comprobar si un documento está bien o no formado. Un documento es válido si está bien formado y, además, lo es conforme a las reglas contenidas en un *DTD* o *Schema*.

(La validación se verá en otro tema, pero podemos adelantar que la validación lo que pretende es comprobar si se cumplen un conjunto de reglas sintácticas para definir etiquetas. Por ejemplo qué etiquetas se pueden usar en un documento, en qué orden deben aparecer, cuáles pueden aparecer dentro de otras, cuáles tienen atributos, etc.).

Así, con todo esto, podemos dividir los *parsers* XML en dos grupos:

- Sin validación: el *parser* no valida el documento utilizando un *DTD* o *Schema*, sino que sólo verifica que el documento esté bien formado (sólo hay una etiqueta raíz, las etiquetas están bien anidadas, etc.).
- Con validación: además de comprobar que el documento está bien formado, comprueba si el documento es válido frente a un *DTD* o *Schema*.

Básicamente, disponemos de cuatro grupos de herramientas: *exploradores*, *navegadores* o *clientes web*, *servicios de comprobación vía web*, *aplicaciones de terceros* y *programas de usuario* usando una API (Interfaz de Programación de Aplicaciones). Dependiendo del trabajo que se quiera realizar usaremos una herramienta u otra.

3.1. Clientes web.

Es algo que ya hemos usado para comprobar que nuestros documentos XML están bien formados. En este caso, el cliente web funciona como interfaz entre nosotros y el analizador. Cuando el cliente web recibe la orden de cargar un documento XML, invoca al analizador XML que lleva incluido pasándole el documento, mostrándose la respuesta que el analizador genera.

El problema de los clientes web es que al tener analizadores distintos, éstos tienen comportamientos diferentes entre sí. Lo que a lo mejor está permitido en uno de ellos, no lo está en otro y viceversa. No obstante, hay que comentar que las últimas versiones de los exploradores web tienen comportamientos muy similares debido a que implementan mejor las recomendaciones del W3C.

3.2. Comprobación vía web.

Sabemos que la norma XML se desarrolló bajo el auspicio del W3C (*World Wide Web Consortium)* cuya página web es http://www.w3c.org.

W3C nos ofrece una herramienta de análisis en http://validator.w3.org. Si visitamos esta dirección se nos muestra la página del *servicio de validación de marcas*. Este servicio permite comprobar si el documento está bien formado y, si contiene una *DTD* o un *Schema* con el que validarse, además se intentará validar.

El servicio de validación permite realizar la comprobación del documento XML de tres maneras:

1. Indicando la URL donde se encuentra el archivo que contiene el documento.
2. Subiendo desde nuestro ordenador al servidor el archivo con el documento XML a comprobar.
3. Escribiendo directamente en un formulario el texto a comprobar.

El servicio nos informa de los errores y de advertencias que pueden tener el documento, así como de las causas de los errores además de consejos relativos a las advertencias.

En Internet hay otros sitios donde validar "*on line*" documentos XML, como por ejemplo:

http://www.xmlvalidation.com
http://www.validome.org/
http://www.w3schools.com/xml/xml_validator.asp
http://www.utilities-online.info/

3.3. Aplicaciones.

Existen editores de documentos XML que ayudan a escribir documentos XML bien formados y sin errores, validarlos frente a DTD o esquemas y mantener una estructura válida. En general un editor XML debería ser capaz de:

- Añadir automáticamente la etiqueta de cierre después de escribir la de apertura.
- Forzar a escribir una estructura XML que sea válida.
- Verificar el documento frente a un DTD o Schema.
- Colorear el código de forma diferente según la sintaxis.

Algunas de las aplicaciones de este tipo clasificadas como Software Open Source son: Serna Free (http://www.syntext.com/products/serna-free/), oXygen XML Editor (la version trial es para 30 días) (http://www.oxygenxml.com/), XML CopyEditor (http://xml-copy-editor.sourceforge.net/).

Como software propietario destaca XMLSpy (http://www.altova.com). Muchos entornos de desarrollo como Visual Studio, Eclipse o NetBeans, disponen de herramientas para trabajar con tecnologías XML.

3.4. Bibliotecas.

Se puede usar una biblioteca de clases para realizar la tarea de validación usando un lenguaje de programación. En muchos lenguajes actuales existe una API, una interfaz de programación, que nos permite construir una pequeña aplicación para determinar por ejemplo si un documento XML está bien formado.

Si el lenguaje está orientado a objetos, la técnica es muy similar en todos ellos: se crea un programa en el que se intenta cargar o leer el documento XML. Dependiendo de que el documento se pueda leer o no, se informará de que está bien formado o no lo está.

El siguiente código escrito en C# es una simple aplicación de consola que nos permite comprobar si un documento está bien formado. Básicamente lo que pretende es crear un objeto que permita tratar con documentos de tipo XML. Posteriormente le decimos al objeto que recupere el archivo XML del disco. Si al intentar cargar dicho archivo comprueba que está bien formado, presentará un menaje indicándolo, en caso contrario se produciría un error y se mostraría en pantalla un mensaje indicando el error.

```
using System;
using System.Xml;
namespace XML_BF
{
    class Program
    {
        static void Main(string[] args)
        {
            XmlDocument xml = new XmlDocument();
            try
            {
                xml.Load("Archivo.xml");
                Console.WriteLine("El documento está bien formado");
            }
            catch (XmlException ex)
            {
                Console.WriteLine("El documento NO está bien formado");
                Console.WriteLine(ex.Message);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.ReadKey();
        }
    }
}
```