4.Definición de esquemas y vocabularios en XML

OBJETIVOS

* Describir la estructura de un documento XML con DTD.
* Conocer los elementos de los que se compone una DTD.
* Definición de entidades en una DTD.
* Creación y asignación de atributos a un elemento desde una DTD.
* Asignación de una DTD a un documento XML.
* Describir la estructura de un documento XML con un esquema.
* Conocer los elementos de los que se compone un esquema.
* Creación y asignación de atributos a un elemento desde un esquema.
* Asignación de un esquema a un documento XML.
* Tipos básicos en los elementos de un esquema.
* Asignación de elementos hijos con modificación de ocurrencias.
* Distinguir entre documento bien formado y documento válido
* Conocer que herramientas de validación existen vía web o aplicación local.

XML ha sido propuesto como un estándar en el intercambio de información, independientemente de la plataforma en la que se genere o se utilice. No solo es imprescindible que el documento esté bien formado (con las etiquetas de apertura y cierre bien ubicadas, que la codificación sea correcta, que los elementos cumplan la sintaxis de XML, etc.), sino que ambos actores (emisor y receptor) se ciñan a una estructura de los datos definida previamente. Para evitar estos casos, se ha de definir una estructura fija del documento que conozcan las partes que intercambian la información.

A continuación, se mostrarán dos maneras de especificar la estructura de los datos que se van a enviar en XML, las DTD y los XML Schema.

4.1 DTD

DTD *(Document Type Definition – Definición de Tipo de Documento)* es la definición de como se construye un documento XML para que se ajuste a las necesidades previamente analizadas. Es decir, establece que elementos son aceptados (léxico) y en que posiciones deben estar dentro de un documento XML (sintaxis).

Cuando se define una DTD y se referencia dentro de un documento XML, se establece una relación de:

- Que léxico es el que se espera (sobre todo **qué** nombres de elementos y atributos).
- Que reglas sintácticas debe cumplir nuestro documento XML (el orden y el número de veces que pueden aparecer los elementos que constituyen el léxico).

El significado (semántica) se proporciona en el uso de la DTD y del documento XML al generar las aplicaciones que lo utilizan.

Por tanto, antes de crear un documento XML se deberá analizar que elementos y etiquetas se van a necesitar para la aplicación que se quiere crear. Para esto sirve una DTD.

¿Por qué resulta importante la creación de DTD? Principalmente, cuando se analiza un almacenamiento de información en XML es porque esa información es importante compartirla. Pero aunque la compartición de la información pueda resultar interesante, que la información esté sujeta a una serie de reglas para que todos los que utilicen el documento XML (en consultas, inserciones, modificaciones, borrados, etc.); se aseguren que esté bien formado y resulte válido para el uso por terceras aplicaciones. Si en el proceso de acceso y modificación del documento XML se incumple su estructura, nos encontraremos con un documento no válido y con un futuro problema en su futura utilización. Por tanto una DTD permitirá asegurar que la información que contiene es válida y cumple los requisitos de intercambio de información entre terceras (personas con la misma DTD).

La creación de una DTD resulta muy simple y se verá más adelante pero resulta interesante como ubicar estas declaraciones:

1. La DTD se puede ubicar dentro del propio documento XML.
2. La DTD se ubica fuera del documento XML (en un fichero externo).

Ambas realizan la misma función pero resulta más cómodo utilizar un fichero externo para almacenar una DTD y realizar un enlace en el documento XML para que la use. Además de la comodidad, existe otra ventaja fundamental:

no se almacena en cada documento XML la DTD. Si tuviéramos muchos ficheros XML que se basan en la misma especificación DTD, y quisiéramos realizar cambios, se tendría que cambiar en todos los ficheros la declaración de la DTD; en cambio, si estuviera enlazada en un fichero externo, se cambiaría una única vez y todos los documentos XML lo aprovecharían. Además se estaría almacenando la misma información en todos los documentos, lo que aumentaría el tamaño del documento XML.

Para continuar, establezcamos un ejemplo en el que utilizar una DTD. Se quiere almacenar los mensajes de móviles que se envían a un servidor. Los datos que se guardaran son los siguientes:

Número de teléfono del usuario.
Fecha de envío
Hora de envío
Contenido del mensaje.

Un ejemplo de documento XML que guarde estos mensajes SMS podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BDsms SYSTEM "BDsms.dtd">
<BDsms>
   <sms>
      <teléfono>955 55 66 55</teléfono>
      <fecha>1/7/2011</fecha>
      <hora>23:55</hora>
      <mensaje>Juego: Tetris</mensaje>
   </sms>
   <sms>
      <teléfono>745 15 56 11</teléfono>
      <fecha>22/9/2011</fecha>
      <hora>15:05</hora>
      <mensaje>Juego2: Arkanoid</mensaje>
   </sms>
   <sms>
      <teléfono>842 35 22 00</teléfono>
      <fecha>10/11/2011</fecha>
      <hora>09:22</hora>
      <mensaje>Juego3: Comecocos</mensaje>
   </sms>
</BDsms>
```

La DTD que permite establecer el formato de intercambio de estos mensajes SMS en el documento XML, podría ser la siguiente (fichero "BDsms.dtd"):

```
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (teléfono, fecha, hora, mensaje)>
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```

Notad que el fichero *BDsms.dtd* esta referenciado en el documento XML. Si se quisiera incluir todo en el mismo documento XML, entonces quedaría:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BDsms
[
<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (teléfono, fecha, hora, mensaje)>
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
]>

<BDsms>
  <sms>
    <teléfono>955 55 66 55</teléfono> <fecha>1/7/2011</fecha>
    <hora>23:55</hora>
    <mensaje>Juego: Tetris</mensaje>
  </sms>
  <sms>
    <teléfono>745 15 56 11</teléfono>
    <fecha>22/9/2011</fecha>
    <hora>15:05</hora>
    <mensaje>Juego2: Arkanoid</mensaje>
  </sms>
  <sms>
    <teléfono>842 35 22 00</teléfono>
    <fecha>10/11/2011</fecha>
    <hora>09:22</hora>
    <mensaje>Juego3: Comecocos</mensaje>
  </sms>
</BDsms>
```

4.1.1 BLOQUES PARA CONSTRUIR UNA DTD

Viendo el ejemplo de la base de datos de SMS, se puede observar una serie de elementos definidos en una DTD:

Elemento: Es el bloque principal con el que se construye los documentos XML.

Atributo: Es una manera de añadir nodos de información a un elemento.

Entidad: En XML existen algunos caracteres que tienen un significado especial. La aparici6n de estos caracteres como datos almacenables hace que se puedan confundir con entidades propias de un documento XML. Son entidades las siguientes:

Entidad	Símbolo
>	>
<	<
"	"
'	'
&	&

Se pueden usar dentro del propio documento XML y tras parsear el documento, tomarán el valor indicado a la derecha. También el usuario puede definir entidades propias para que, después de analizar el documento XML, se sustituyan por el valor indicado. Es como una especie de definición

preestablecida. Un par de ejemplos son los siguientes:

<!ENTITY pi "3.141592"> => π
<!ENTITY textFile SYSTEM "fichero.txt"> => &textFile;
<!ENTITY miURL SYSTEM “http://www.as.com/fichero.txt”> => &miURL;

- **ELEMENTOS.**

Se declaran de una de las dos siguientes maneras:

<!ELEMENT nombre_del_elemento especificación_de_contenido>

0

<!ELEMENT nombre_del_elemento (nodos_hijos)>

En el segundo caso se pueden definir el *nombre_del_elemento* como el nodo padre del que cuelgan un conjunto de *nodos_hijos* (separados por comas).

La **especificación_de_contenido** puede ser**:**

*** #PCDATA:** su significado en inglés es *Parsed Character Data.* Indica que entre la etiqueta de apertura y cierre de ese elemento, se almacenarán caracteres como texto y serán analizados por un *parser.* Como es lógico, al analizarse mediante el *parser* el contenido del texto para encontrar entidades y elementos, no podrá encontrarse ninguno de los caracteres anteriormente mencionados (< > & ” ') que deberán ser sustituidos por sus respectivas entidades, pues el *parser* podría confundirse al analizar el contenido del elemento.

Ejemplo: <!ELEMENT teléfono **(#PCDATA)>**

*** EMPTY**: indica que el elemento está vacío. Un ejemplo de uso es la etiqueta ***
*** de XHTML. Esta etiqueta permite establecer un salto de línea que debería estar indicado como ***
</br>***. Si se declara como *EMPTY,* entonces se simplifica el salto de línea de la siguiente manera: ***
***. Resulta más cómodo.

Ejemplo: <!ELEMENT saltoLinea **EMPTY> =>** <saltoLinea/>

*** ANY**: indica que el elemento puede contener cualquier cosa, **PCDATA** o incluso otros elementos anidados.

Ejemplo: <!ELEMENT observaciones **ANY**> =>

```
<observaciones>
        Los siguientes alumnos deberán volver a examinarse:
        <alumno>
                <nombre>Francisco Javier</nombre>
                <apellidos>López Pérez</apellidos>
        </alumno>
        ……...
</observaciones>
```

*** SECUENCIAS o Listas de hijos**; indica que el elemento está compuesto de uno o más hijos. Ejemplo:

<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (teléfono+, fecha, hora, mensaje**)>**
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>

Podemos ver que el elemento raíz BDsms está compuesto de sms (cero o más sms) y que el elemento sms (padre) a su vez está compuesto **uno o más elementos** teléfono, fecha, hora y mensaje (hijos).

Imaginemos que una vez definida esta DTD, nos damos cuenta que por cada SMS recibido se pueden almacenar varios mensajes. En un primer momento se puede pensar que para salir del paso la solución sería esta:

<!ELEMENT BDsms (sms*)>
<!ELEMENT sms (teléfono, fecha, hora, mensaje,**mensaje2)>**
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
<!ELEMENT mensaje2 (#PCDATA)>

¿Por qué no es conveniente? La razón fundamental es que estamos haciendo cambios en la DTD que obligarán a algunos sms que quizás no tengan "mensaje2" a tenerlo. Además estamos particularizando un caso concreto en vez de fomentar la generalidad de la definición de la DTD. ¿Cómo resolverlo?

En este caso debería cambiar la ocurrencia de la aparición de los elementos, justo cuando los indicamos en el elemento padre. Las ocurrencias que pueden aparecer se indican con los siguientes operadores:

'(nada)': indica que aparece obligatoriamente una vez. Por ejemplo *fecha* debe aparecer una y sólo una vez dentro de *sms*.
'+': indica que debe haber una o más ocurrencias del elemento indicado.
<!ELEMENT sms (teléfono, fecha, hora, mensaje+)>
'*': indica que puede haber cero o más ocurrencias del elemento indicado.
<!ELEMENT BDsms (sms*)>
'?': indica que puede haber cero o una ocurrencia del elemento indicado (también llamado opcionalidad). <!ELEMENT sms (teléfono, fecha, hora, mensaje?)>

No sólo se pueden cambiar las ocurrencias de los hijos declarados, sino que podemos indicar la opcionalidad de aparición de hijos. En este caso podríamos definir que queremos almacenar o la hora o el mensaje pero no las dos simultáneamente y si obligatoriamente una de las dos, por lo que quedaría así:

<!ELEMENT BDsms (sms*)>
<!ELEMENT sms(teléfono, fecha, (hora | mensaje))>
<!ELEMENT teléfono (#PCDATA)>
<!ELEMENT fecha (#PCDATA)>
<!ELEMENT hora (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>

*** CHOICES (elección de uno de entre varios hijos)**: indica que un elemento estará compuesto por uno de entre varios hijos.
Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dibujo [
        <!ELEMENT dibujo (figura+)>
        <!ELEMENT figura (circulo | cuadrado)>
        <!ELEMENT circulo (centro,radio)>
        <!ELEMENT centro (#PCDATA)>
        <!ELEMENT radio (#PCDATA)>
        <!ELEMENT cuadrado (lado)>
        <!ELEMENT lado (#PCDATA)>
]>
<dibujo>
        <figura>
             <circulo>
                  <centro>4.5</centro>
                  <radio>5.5</radio>
             </circulo>
        </figura>
        <figura>
             <cuadrado>
                  <lado>2.5</lado>
             </cuadrado>
        </figura>
</dibujo>
```

Los *"choices"* pueden anidarse mediante paréntesis en cualquier combinación, p.ej.:
<!ELEMENT círculo (centro,(radio|diámetro))>
indica que un círculo está compuesto por un centro y por un radio o un diámetro.
<!ELEMENT centro ((x,y) | (r,ang))> indica coordenadas "cartesianas" o "polares".

<!ELEMENT tabla_nombres (nombre_completo*)>

```
<!ELEMENT nombre_completo
        ((ap1,ap2?) |
          (nombre,(
                    (nombre+, ap1,ap2?) | (ap1,ap2?)
                  )
          )
        )
>
```

*** CONTENIDO MEZCLADO**

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE diccionario
[
<!ELEMENT diccionario (definición*)>
<!ELEMENT definición (#PCDATA | termino)*>
<!ELEMENT termino (#PCDATA)>
]>
<diccionario>
        <definición>
                Una <termino>Máquina de Turing</termino> se refiere a un autómata finito
                con infinita memoria que puede probarse equivalente a cualquier otro
                autómata finito con una cantidad grande arbitraria de memoria. Así lo
                que es válido para una Máquina de Turing es válido para todas las
                Máquinas de Turing sin importar como están implementadas.
        </definición>
</diccionario>
```

- **ATRIBUTOS.**

La manera de declarar atributos que forman parte de los elementos en una DTD es de esta forma:

<!ATTLIST **nombre_elemento nombre_atributo tipo_atributo valor_por_defecto**>

nombre_elemento: es el elemento al que se le quiere añadir el atributo.

nombre_atributo: es el nombre del atributo que se quiere añadir

tipo_atributo puede ser:

CDATA: texto que podrá tener cualquier carácter.
ID: es un identificador que permite identificar al elemento de manera única en todo un documento XML (por desgracia su valor debe comenzar con _ : o una letra, no lo vamos a poder usar para identificadores que empiezan por número, como suelen ser los campos ID de la mayoría de tablas)
IDREF: es un identificador de otro elemento del propio documento XML.
IDREFS: es una lista de identificadores a otros elementos, separados por espacios.
Enumerados (valor1 | valor2 | valor3 | ...): el valor es uno de los indicados en esta lista (definida por el usuario).
NMTOKEN: es un texto que sólo podrá tener letras, dígitos, guion "-", subrayado "_", punto "." y dos puntos ":". Es un nombre válido XML, aunque menos restrictivo, los valores de este tipo de atributo pueden empezar por "." , "-" , "_", ":" o una letra.
NMTOKENS: es una lista de nombres XML validos. Es como un ***NMTOKEN*** pero se incluyen los espacios en blanco " ", tabuladores o retornos de carro.
ENTITY: el tipo del atributo es una entidad que se ha declarado anteriormente.
ENTITIES: Es una lista de entidades.

valor_por_defecto: puede indicarse lo siguiente:

#REQUIRED. El atributo es obligatorio.
#IMPLIED. El atributo es opcional (puede omitirse)
#FIXED. El atributo toma siempre un valor fijo, como en el siguiente ejemplo:
<!ATTLIST documento version CDATA #FIXED "1.0">
si se omite dicho atributo es como si estuviera presente con el citado valor.
Literal. Si en lugar de los anteriores suministramos un valor, éste será su valor por defecto, si se omite el atributo se le asignará el valor por defecto y si existe el atributo con un valor diferente prevalecerá éste último.

Como no se ha declarado ningún atributo en el ejemplo de los SMS, cambiaremos el elemento *hora* para añadir la zona o franja horaria (de manera obligatoria). Se quiere tener un documento XML de este tipo:

…
<hora zona="GMT+1">09:22</hora>

por lo que el DTD debe añadir la siguiente linea:

<!ATTLIST hora zona CDATA "GMT+1">

<hora>

10:15:25

</hora>

El atributo zona puede omitirse y en tal caso es cómo si se hubiera puesto con el valor "GMT+1" por defecto.

NOTAS

IBM, en la década de los 60, inventó un lenguaje al que denominó GML (General Markup Language) orientado a definir la estructura de un documento con determinado formato y hacer uso de él para almacenar toda la información que se manejaba en la empresa. La ISO (Internacional Oranization for Standardization), viendo el potencial de GML empezó a trabajar en una estandarización del mismo. En 1986 saca a la luz SGML (Standard Generalized Markup Language), basado en él apareció en 1989 HTML y en 1998 XML.

EJERCICIOS

Realizad un documento XML con dos registros de datos de ejemplo para cada una de las tablas y una DTD enlazada con el citado documento que refleje la estructura de la misma, para cada uno de los siguientes supuestos:

4.1 Se quiere intercambiar información en formato XML para ser introducida en una base de datos con el siguiente esquema relacional:

PIEZAS(**Código**,nombre)
PROVEEDOR(**id**, nombre)
SUMINISTRA(**Código_Pieza,idProveedor**,precio)

4.2 Ídem para el siguiente esquema:

proveedores(**nro_pro**,nom_p,categoria, ciud_p?)
items(**nro_i**,descripcion_i,ciudad_i)
pedidos(**nro_pe**,nro_c,nro_i,nro_pro,cantidad,precio)
clientes(**nro_c**,nom_c,ciudad_c)

4.3 Ídem para el siguiente esquema:

FACULTAD(**Código**, nombre)
INVESTIGADORES(**DNI**, NomApell,facultad)
EQUIPOS(**NumSerie**, Nombre, facultad)
RESERVA(**DNI,NumSerie**, comienzo, fin)

4.4 Ídem para el siguientes esquema:

Hospital (**hospital_cod,** nombre, direccion, telefono, num_camas);
Sala (**hospital_cod**, **sala_cod**, nombre, num_cama);
Plantilla (hospital_cod, sala_cod **, empleado_num**, apellido, funcion, turno, salario);
Ocupacion (**inscripcion,** hospital_cod, sala_cod, cama);
Doctor (hospital_cod, **doctor_num,** apellido, especialidad);
enfermo(inscripcion,apellido,direccion,fecha_nac,s,**nss);**

4.2 ESQUEMAS

Hasta el momento se ha hablado de como generar un lenguaje XML con la definición de las reglas que lo componen (DTD). Otra manera de formalizar esas reglas es con un lenguaje llamado **XML Schema** (también llamado **XSD o *XML** Schema Definition).* Se puede decir que un XSD (o esquema de ahora en adelante), es la evolución natural de un DTD. Permite expresar con mayor potencia, gramáticas más complejas utilizando la misma sintaxis de XML (lo que facilita enormemente el trabajo). Nació en 1998 y se recomendó el uso en el 2001 por el W3C *(World Wide Web Consortium).* Las características principales de un esquema son las siguientes:

Define que elementos pueden aparecer en un documento XML.
Define que atributos pueden aparecer en un documento XML.
Define que elementos son compuestos, indicando que elementos hijos deben aparecer y en que orden.
Define que elementos pueden ser vacíos o que pueden incluir texto asociado.
Define los tipos que pueden utilizarse en cada elemento o atributo.
Define la obligatoriedad, la optatividad de elementos y/o atributos.

¿Qué diferencia un **DTD** de un esquema XSD si sirven para lo mismo?

La principal ventaja de XSD es que al estar basado en XML, es fácilmente extensible a las futuras modificaciones o necesidades que se identifiquen. Además no es necesario aprender un nuevo lenguaje (en contraposición con los DTD). También permite definir de manera muy clara los tipos de datos y los espacios de nombres que se soportan. Esto es realmente importante puesto que si, por ejemplo, un elemento se declara que es de tipo *integer* y, finalmente, se escribe un dato que no “encaja” con un entero, no se admitirá la validez del documento XML hasta que se corrija ese error. Es más, se pueden definir los tipos exactamente igual que si fuese una base de datos, facilitando la conversión de uno a otro y las transferencias de información. Más adelante hablaremos de la validez de los documentos.

Como en la sección anterior, partiremos de un documento XML y definiremos el esquema. En la base de datos de SMS, teníamos mensajes almacenados como los siguientes:

```
<?xml version="1.0" encoding="UTF-8"?>
<BDsms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="BDsms.xsd">
<!-- <!DOCTYPE BDsms SYSTEM "BDsms.dtd">
<BDsms>-->
        <sms>
                <teléfono>955556655</teléfono>
                <fecha>2001-07-01</fecha>
                <hora>23:55:00</hora>
                <mensaje>Juego: Tetris</mensaje>
        </sms>
        <sms>
                <teléfono>745155611</teléfono>
                <fecha>2020-02-29</fecha>
                <hora>15:05:00</hora>
                <mensaje>Juego2: Arkanoid</mensaje>
        </sms>
        <sms>
                <teléfono>842352200</teléfono>
                <fecha>2011-11-10</fecha>
                <hora>13:00:00</hora>
                <hora>13:00:00</hora>
                <mensaje>Juego3: Comecocos</mensaje>
        </sms>
</BDsms>
```

Si el esquema no está almacenado en una ubicación en la red, podemos utilizar esta otra forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<BDsms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="BDsms.xsd">
```

Antes teníamos **una** DTD que permitía validarlo. Ahora definiremos el esquema en los mismos términos que el anterior (fichero *BDsms.xsd):*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1" xml:lang="es">
   <xs:element name="BDsms">
      <xs:complexType> <!-- Porque está compuesto por otros elementos -->
         <xs:sequence> <!-- Para decir que está compuesto de -->
            <xs:element name="sms" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="teléfono" type="xs:integer" />
                     <xs:element name="fecha" type="xs:date" />
                     <xs:element name="hora" type="xs:time" minOccurs="1" maxOccurs="2" />
                     <xs:element name="mensaje" type="xs:string" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

En negrita están destacados los elementos mínimos que componen un esquema (líneas 1 y 2 y última)

4.2.1 ELEMENTO RAÍZ

Un esquema se puede componer de muchos elementos enlazados. El elemento principal (***xs:schema***) es el elemento raíz, y deberá estar siempre.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="0.1" xml:lang="es">

....
</xs:schema>
```

En el documento xml enlazamos con el esquema

```
<?xml version="1.0" encoding="UTF-8"?>
<elementoRaiz  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="ficheroEsquema.xsd">
   ….....
</elementoRaiz>
```

Este elemento raíz (***xs:schema***) puede contener algunos atributos interesantes. En el ejemplo anterior se vio que se indicaba el espacio de nombres, una *version* y un lenguaje predefinido. Entre los atributos más importantes se encuentra el siguiente:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Indica cual es el espacio de nombres en el que se basa para saber que elementos y tipos de datos son los soportados en el esquema.

Dentro del elemento raíz y **antes** de empezar a definir los elementos de que consta el esquema se pueden indicar "anotaciones", dirigidas a los humanos:

```
<xs:annotation>
   <xs:documentation xml:lang="es-ES">
      Esquema de ejemplo para el Tema 4
      <a href="http://www.javier.com/xmlej">LMSGI</a>
      Documento de ejemplo para el tema 4 ...........
   </xs:documentation>
</xs:annotation>
```

Y dirigidas a las máquinas (aplicaciones):

```
<xs:annotation>
   <xs:appinfo>
      <help-text>Formato de sms transmisibles por el móvil</help-text>
      <iva>21</iva>
   </xs:appinfo>
</xs:annotation>
```

En lugar de introducir comentarios <!-- --> que al algunos "parsers" pueden quitar del esquema.

4.2.2 ELEMENTOS SIMPLES

El resto de los elementos son los que el usuario puede definir para declarar un lenguaje XML. Llamamos elemento simple a aquel que puede contener información y que no tiene otros elementos hijos asociados a él. Tampoco puede tener atributos en su interior. En el árbol que representa a la jerarquía de los elementos, son los nodos hoja en los que solo habrá datos.

Para definir un elemento nuevo, la sintaxis a seguir es la siguiente:

```
<xs:element name="nombre_elemento" type="tipo_elemento" />
```

El nombre puede ser el que desee el usuario (mientras que no esté repetido y sea un nombre válido de elemento XML). Sin embargo, el tipo tiene que ser uno de los siguientes:

xs:string

<xs:element name="nombre" type="xs:string"/> <!-- xsd -->

<nombre>Pepito Pérez</nombre> <!-- xml -->

xs:date

<xs:element name="fechaNacimiento" type="xs:date"/> <!-- xsd -->

<fechaNacimiento>1979-02-04</fechaNacimiento> <!-- xml -->
<fechaNacimiento>1979-02-04+01:00</fechaNacimiento> <!-- xml -->

xs:time
<xs:element name="hora" type="xs:time"/>

<hora>23:55:15</hora>
<hora>23:55:15+01:00</hora>
<hora>23:55:15.1</hora>

xs:dateTime
<xs:element name="fecha" type="xs:dateTime"/>

<fecha>1979-02-04T23:55:15</fecha>
<fecha>1979-02-04T23:55:15+01:00</fecha>
<fecha>1979-02-04T23:55:15.1</fecha>

xs:decimal
<xs:element name="precio" type="xs:decimal"/>

<precio>1205.74</precio>
<precio>-1205.74</precio>

xs:integer
<xs:element name="vueltas" type="xs:integer"/>

<vueltas>1205</vueltas>
<vueltas>-1205</vueltas>

xs:boolean
<xs:element name="pagado" type="xs:boolean"/>

<pagado>true</pagado>
<pagado>false</pagado>

xs:hexBinary - xs:base64Binary
<xs:element name="imagen" type="xs:hexBinary"/>
<xs:element name="foto" type="xs:base64Binary"/>
<!-- entre las etiquetas va una codificación binaria en hexadecimal o base64. Se utiliza para almacenar documentos o formatos binarios. -->

En la siguiente tabla se enumeran todos los tipos disponibles, además de los anteriormente citados:

Tipos Simples	
Tipo	Descripción
anyURI	A Uniform Resource Identifier
base64Binary	Base64-encoded binary data
hexBinary	Hexadecimal-encoded binary data
boolean	May contain either true or false, 0 or 1
byte	A signed byte quantity >= -128 and <= 127

dateTime	An absolute date and time
duration	A length of time, expressed in units of years, months, days, hours, etc.
ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS	Same values as defined in the attribute declaration section of the XML 1.0 Recommendation (los mismos que los de las DTD's)
integer	Any positive or negative integer
language	May contain same values as xml:lang attribute from the XML 1.0 Recommendation
Name	An XML name
string	Unicode string
decimal	Un número decimal

Imaginemos que queremos establecer un conjunto de elementos simples para almacenar la información de un alumno del instituto. Los datos a almacenar serian:

Código de alumno.
Nombre.
Apellido1.
Apellido2.
D.N.I.
Fecha de nacimiento.
Curso.
Cuota pagada *(booleano).*

Podríamos definir los siguientes elementos:

```
<xs:element name="código" type="xs:integer"/>
<xs:element name="nombre" type="xs:string"/>
<xs:element name="apellido1" type="xs:string"/>
<xs:element name="apellido2" type="xs:string”/>
<xs:element name="dni" type="xs:string"/>
<xs:element name="fechaNacimiento" type="xs:date"/>
<xs:element name="curso" type="xs:integer"/>
<xs:element name="cuotaPagada" type="xs:boolean"/>
```

Imaginemos que se quiere indicar, por defecto, que todo alumno tiene la cuota sin pagar. Cuando se define el elemento *cuotaPagada* podemos añadir ese valor por defecto de la siguiente forma:

esquema: `<xs:element name="cuotaPagada" type="xs:boolean" default="false"/>`

(esto en realidad no tiene mucha utilidad ya que un elemento no se puede omitir, si se omitiera cuotaPagada su valor sería vacío no el valor por defecto.)

documento xml: `<cuotaPagada>true</cuotaPagada>`

** Lo mismo con un DTD sería:

dtd: <!ELEMENT cuota EMPTY>

<!ATTLIST cuota pagada NMTOKEN “false”>

documento xml: <cuota pagada=”kdkdk”/> (nada me impide poner cualquier cosa)

En cambio, si lo que se quiere es que tenga un valor fijo siempre, el cambio seria el siguiente:

```
<xs:element name="cuotaPagada"  type="xs:boolean" fixed="false"/>
```

```
<cuotaPagada>false</cuotaPagada>
```

4.2.3 ATRIBUTOS

Los elementos declarados con un tipo simple no pueden contener atributos.

Los atributos son complementos de información que se pueden asignar a un elemento previamente declarado. La sintaxis de un atributo es la siguiente:

```
<xs:attribute name="nombre_atributo" type="tipo_atributo">
```

En los campos *name y type* se aplica exactamente lo mismo que lo visto en los elementos simples (lo cual es más coherente que en los DTDs)

Siguiendo con el ejemplo de los alumnos de una escuela, nos piden añadir un atributo al elemento curso puesto que en un colegio puede darse el caso de tener varios cursos concurrentes y no pertenecer a la misma clase (por ejemplo, 1° letra A, 1° letra B...).

```
<curso letra="A">1</curso>
```

Esto sería si no tuviese atributos:

```
<xs:element name="curso" type="xs:integer"/>
```

Ahora debemos añadirle lo siguiente:

```
<xs:element name="curso"> <!-- no se puede declarar un tipo simple si queremos atributos -->
   <xs:complexType> <!-- el elemento curso puede contener atributos -->
      <xs:simpleContent> <!-- el elemento curso es de un tipo simple de datos -->
         <xs:extension base="xs:integer"> <!-- el tipo de dato de curso es xs:integer -->
            <xs:attribute name="letra" type="xs:string"/><!--curso contiene 1 atrib. letra -->
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

```
Lo mismo con un DTD, sería:
        <!ELEMENT curso (#PCDTA)> <!-- aquí no podemos asegurar que el contenido del elemento curso sea un entero-->
                <!ATTLIST curso letra NMTOKEN #REQUIRED>
```

Los atributos pueden tomar valores *por defecto*, *fijos* u *opcionales*, a continuación se ven ejemplos de los tres casos:

```
<xs:attribute name="letra" type="xs:string" default="A"/>
```

```
<xs:attribute name="letra" type="xs:string" fixed="A"/
```

```
<xs:attribute name="letra" type="xs:string" use="optional"/>
```

Por desgracia el valor por defecto de ***use*** es ***optional***, lo cual quiere decir que a la mayoría de atributos necesitaremos ponerles el `use="required"`.

4.2.4 RESTRICCIONES

En algunas ocasiones es importante establecer un rango de valores que puede tomar un elemento. Hasta ahora no se ha podido realizar este tipo de condicionantes. Imaginemos que los alumnos anteriores deben cumplir la condición de estar entre los 16 y los 24 años para poder hacerse el carnet joven. Si la edad no está en ese rango no pueden solicitarlo. Si el elemento ***edad*** fuese de tipo simple lo definiríamos así:

```
<xs:element name="edad" type="xs:integer"/>
```

Para añadir la restricción indicada de que la edad esté en [16,24]:

```
<xs:element name="edad">
   <xs:simpleType> <!-- indica que el contenido del elemento edad es de tipo simple -->
      <xs:restriction base="xs:integer"> <!-- edad es de tipo entero -->
```

```
            <xs:minInclusive value="16" />
            <xs:maxInclusive value="24" />
            <!-- lo mismo con una E.R.: <xs:pattern value="(1[6-9]|2[01234])" /> -->
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Las restricciones no son sólo útiles para los elementos sino que se pueden aplicar a los atributos también. En este caso vamos a declarar una restricción sobre el atributo "letra", para que tome uno de los valores determinados en una lista ("A", "B", "C", "D").

También sería una solución válida la siguiente:

Ejemplo: `<curso letra=”A”>1</curso>`

```
<xs:element name="curso">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:integer">
            <xs:attribute name="letra">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:enumeration value="A"/>
                     <xs:enumeration value="B"/>
                     <xs:enumeration value="C"/>
                     <xs:enumeration value="D"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:attribute>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

```
<xs:element name="curso">
  <xs:complexType>
     <xs:simpleContent>
        <xs:extension base="xs:integer">
           <xs:attribute name="letra">
              <xs:simpleType>
                 <xs:restriction base="xs:string">
                    <xs:pattern value="[A-D]" />
                 </xs:restriction>
              </xs:simpleType>
           </xs:attribute>
        </xs:extension>
     </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

En la parte de la izquierda de la figura anterior se ha hecho una enumeración, mientras que en la parte de la derecha se han definido un patrón equivalente. El patrón se ha definido usando una ***expresión regular***, lo que permite tener mayor potencia para declarar valores predeterminados. Como alternativas podrían haberse usado las expresiones regulares equivalentes:

`<xs:pattern value="[ABCD]" />`

`<xs:pattern value="(A|B|C|D)" />`

También hay posibilidad de establecer restricciones sobre la longitud de lo contenido por un elemento. El siguiente ejemplo clarifica bastante esta restricción:

```
<xs:element name="dni">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:length value="9" /> <!-- O este o el maxlength -->
         <xs:minLength value=”2” />
         <xs:maxLength value=”9” />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

xs:lenght: establece una longitud fija. En este caso es redundante, no hace falta ponerlo *(length junto a minLength equivale a maxLength)*

xs:minLength: establece un mínimo en la longitud.

xs:maxLength: establece un máximo en la longitud.

Para el dni vendría como anillo al dedo:

```
<xs:pattern value="[0-9]{1,8}[A-HJ-NP-TV-Z]" />
```

Ej. Definir un elemento dentro del esquema para representar una dirección IP
Solución:

```
<xs:element name="dirIP">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:pattern
            value="(([1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}([1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Ej. Definir un elemento dentro del esquema para representar una dirección de email válida

```
<xs:element name="email">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:pattern value="[a-zA-Z0-9\.\-_]+@(hotmail|gmail|yahoo)\.(com|es)" />
         <xs:pattern value="[a-zA-Z0-9\.\-_]+@([a-zA-Z0-9\-_]+\.){1,126}[a-z]{2,4}” />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Existe otro tipo de restricciones que van asociadas al contenido que almacena el elemento. En concreto son importantes aquellas que permiten definir el comportamiento que debe tener el procesador XML cuando se encuentra un espacio en blanco. Imaginemos que se quiere almacenar la dirección de vivienda habitual del alumno. Es claro que deberíamos crear un nuevo elemento:

```
<xs:element name="dirección" type="xs:string"/> <!-- xsd -->

<dirección>C/ Molino, 13</dirección> <!-- xml -->
```

```
<xs:element name="dirección">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:whiteSpace value="replace" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

En el caso de los espacios en blanco nos encontramos con las siguientes opciones:

- **preserve:** mantiene los espacios en blanco que almacene entre sus etiquetas de apertura y cierre (realmente mantiene los espacios, tabuladores, saltos de linea y retornos de carro).
- **collapse:** borra todos los espacios en blanco que encuentre entre sus etiquetas de apertura y cierre (espacios, tabuladores, saltos de linea y retornos de carro).
- **replace:** sustituye todos los espacios en blanco, tabuladores, saltos de linea y retornos de carro por un único espacio en blanco (lo mismo que hacen los navegadores en el código *HTML*).

Hemos visto las más representativas a la hora de definir un esquema XML, pero existen más, cuyo nombre define su funcionalidad perfectamente:

- **xs:maxExclusive:** es el máximo valor. Serían válidos todos aquellos que sean menores que el indicado.
- **xs:minExclusive:** es el mínimo valor. Serían válidos todos aquellos que sean mayores que el indicado.
- **xs:totalDigits:** indica el número exacto de cifras permitidas en el elemento o atributo. Debe ser mayor que cero.
- **xs:fractionDigits:** indica el número máximo de decimales que se permiten. Debe ser mayor o igual a cero.

4.2.5 ELEMENTOS COMPLEJOS

En contraposici6n a los elementos simples, **los elementos complejos son todos aquellos que se componen de otros elementos y/o que tinen atributos propios**. Podemos identificar como elementos complejos los siguientes:

- Elementos que contienen otros elementos en su interior.
- Elementos que contienen atributos en su interior.
- Elementos que contienen otros elementos y atributos en su interior.

Un ejemplo de elemento complejo es el siguiente (por tener un atributo):

```
<curso letra="K">1</curso> <!- xml -->
```

Cuya estructura quedaría reflejada en el esquema de la siguiente manera:

```
<xs:element name="curso">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:int">
            <xs:attribute name="letra" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

Siguiendo con el ejemplo de los mensajes a móviles:

```
<xs:element name="BDsms">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="sms" minOccurs="2" maxOccurs="10">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="teléfono">
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:int">
                              <xs:attribute name="movil" type="xs:boolean" />
                           </xs:extension>
                        </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="fecha" type="xs:string" />
                  <xs:element name="hora" type="xs:string" />
                  <xs:element name="mensaje" type="xs:string" />
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

```
<teléfono movil="true">1234</teléfono><!-- xml -->
```

Podemos ver que el elemento *BDsms* se compone de elementos *sms*. Y un *sms* se compone de

elementos de tipo *teléfono, fecha, hora y mensaje.* A la hora de definir un elemento complejo en el esquema, hay que indicarlo de alguna manera. Esta manera es añadiendo, a continuación, un elemento llamado ***<xs:complexType>***. Justo después de añadirlo, si el elemento tiene hijos, se inserta otro elemento llamado ***<xs:sequence>*** que permite indicar que los siguientes elementos a definir se declaran como hijos del elemento inicial (en ese determinado orden (secuencia)). Así se define un elemento con hijos como subelementos.

Se ha cambiado el elemento *teléfono* para que admita un atributo ***movil*** que antes no tenía y que por lo tanto lo convierte en un ***complexType***.

4.2.6 SECUENCIA DE ELEMENTOS

En el punto anterior, se hablaba del elemento ***<xs:sequence>***, que permite indicar una determinada secuencia de hijos en un elemento complejo. Pero no se ha hablado de los indicadores que permiten, por ejemplo, marcar cuantas veces deben aparecer determinados elementos. Es lo que se llama **indicadores.**

Un indicador permite determinar los siguientes patrones:

- El orden en el que se establecen los elementos hijos.
- La ocurrencia (número de veces) con la que aparecen los elementos hijos.
- El grupo al que pertenecen los elementos hijos.

Los **indicadores de orden** permiten saber como se pueden ubicar los elementos hijos. Dentro de este tipo de indicador podemos encontrarnos tres:

- **<xs:all>**: especifica que todos los elementos hijos pueden aparecer en cualquier orden determinado, siempre que solo aparezcan una única vez.
- **<xs:choice>:** especifica que de entre los elementos hijos sólo pueden aparecer o uno u otro.
- **<xs:sequence>:** ya lo conocemos y permite indicar el orden especifico en el que deben aparecer los elementos hijos.

Los **indicadores de ocurrencia** permiten conocer cuantas veces pueden repetirse cada elemento hijo. Podemos distinguir los siguientes:

- **minOccurs:** indica el mínimo numero de veces que un elemento hijo puede aparecer.
- **maxOccurs:** indica el máximo numero de veces que un elemento hijo puede aparecer.

En estos dos indicadores, **los valores por defecto son "1"**. Si pudiera concurrir un numero ilimitado de veces un elemento hijo, se indicaría de la siguiente forma:

```
<xs:element name="sms" maxOccurs="unbounded">
```

Los indicadores de grupo facilitan la manera de establecer un conjunto de elementos asociados entre si. La mejor manera de ver su funcionalidad es con un ejemplo. Imaginemos que queremos representar las características de un coche. Nos gustaría almacenar lo siguiente:

Marca.

Modelo.

Caballos.

Como cualquier automóvil tiene esas características, la definición del esquema podría quedar así:

```
…….....
<xs:element name="coche" type="TipoCoche" />
<xs:element name="autobus" type="TipoAutobus" />
…….....
```

Lo que va a continuación (definición de grupos y tipos) debe ir fuera del elemento raíz, se suele acostumbrar a poner las definiciones de grupos y tipos tras el cierre (***</xs:element>***) del elemento raíz y antes del cierre del documento (***</xs:schema>***):

```
<xs:group name="grupoCoche">
   <xs:sequence>
      <xs:element name="marca" type="xs:string" />
      <xs:element name="modelo" type="xs:string" />
      <xs:element name="caballos" type="xs:integer" />
   </xs:sequence>
</xs:group>
<xs:complexType name="TipoCoche">
   <xs:sequence>
      <xs:element name="código" type="xs:integer" />
      <xs:group ref="grupoCoche" />
      <xs:element name="combustible" type="xs:string" />
   </xs:sequence>
</xs:complexType>
<xs:complexType name=”TipoAutobus”>
   <xs:sequence>
      <xs:element name="código" type="xs:integer" />
      <xs:group ref="grupoCoche" />
      <xs:element name="numPlazas" type="xs:integer" />
      <xs:element name="combustible" type="xs:string" />
   <xs:sequence>
</xs:complexType>
```

En el documento xml los elementos coche y autobús aparecerían:

```
<coche>
   <código>1</código>
   <marca>Ford</marca>
   <modelo>Focus</modelo>
   <caballos>110</caballos>
   <combustible>Gasolina</comnbustible>
</coche>
<autobus>
   <código>1</código>
   <marca>Volvo</marca>
   <modelo>X123</modelo>
   <caballos>1100</caballos>
   <numPlazas>65</numPlazas>
   <combustible>Gasoil</combustible>
</autobus>
```

El mismo concepto se puede seguir con grupos de atributos:

```
<xs:attributeGroup name="grupoCoche2">
   <xs:attribute name="marca" type="xs:string" />
   <xs:attribute name="modelo" type="xs:string" />
   <xs:attribute name="caballos" type="xs:integer" />
</xs:attributeGroup>
<xs:complexType name="TipoCoche">
   <xs:sequence>
      <xs:element name="código" type="xs:integer" />
      <xs:element name="modeloCoche">
         <xs:complexType>
            <xs:attributeGroup ref="grupoCoche2" />
         </xs:complexType>
      </xs:element>
      <xs:element name="combustible" type="xs:string" />
   </xs:sequence>
</xs:complexType>
```

Un ejemplo en el documento xml de tipo TipoCoche ahora sería:

```
<coche>
   <código>2</código>
   <modeloCoche marca=”Ford” modelo=”Focus” caballos=”110” />
   <combustible>Gasolina</combustible>
</coche>
```

Los grupos permiten modularizar los elementos en pequeños trozos y reutilizarlos si fuera necesario en otras partes del esquema que se define.

Para finalizar, existe un elemento que permiten extender la funcionalidad de un documento XML sin tener que especificar lo que debe venir en el esquema. Es como la utilización de tipos de objetos *void* en *C++*, es decir, podemos esperarnos cualquier tipo de objeto (en nuestro caso, elemento). En este caso el elemento se llama *<xs:any>* (*https://faculty.kfupm.edu.sa/ics/sukairi/swe423(021)/W3Schools/schema/schema_complex_any.asp.htm*).

Con los indicadores de ocurrencia podemos indicar cuantos elementos genéricos podemos encontrarnos en ese punto en el esquema. Suele dar flexibilidad a los documentos pero desvirtúa ligeramente el cometido principal de los esquemas (que es tener bien definido todo lo que te puedes encontrar en un documento XML). A efectos, el elemento *<xs:anyAttribute>* (*https://faculty.kfupm.edu.sa/ics/sukairi/swe423(021)/W3Schools/schema/schema_complex_anyattribute.asp.htm*) permite realizar lo mismo pero a nivel de atributos.

NOTAS

*** RELAX NG** *(REgular LAnguage for XML Next Generation)* es una nueva manera de especificar un patrón para la estructura y el contenido de un documento XML. A efectos es como declarar una DTD o un XML Schema. La principal ventaja es que es más sencillo de utilizar y bastante más intuitivo.

*** *Schematron*** es un lenguaje de validación basado en reglas (en vez de gramáticas). Se suele utilizar para extender la funcionalidad de DTD, XML Schema o RELAX NG.

EJERCICIOS.

4.5 Repetir los ejercicios 4.1 a 4.4 utilizando esquemas

4.3 VALIDACIÓN DE DOCUMENTOS XML

La validación de los documentos XML es una serie de comprobaciones que permiten saber si el documento XML este bien formado y si se ajusta a una estructura previamente definida (ya sea DTD o esquemas). Cuando se habla de documento bien formado se quiere decir que se siguen las normas y reglas básicas que se establecen en todos los documentos XML (vistas en el tema anterior). Cuando se habla de un documento válido, quiere decir que además de estar bien formado, su estructura es acorde con las normas que se dictan en la definición del tipo de documento o esquema asociado.

Resumiendo, el proceso de validación es necesario:

- Porque nos permite asegurar que en una transferencia de información XML entre un emisor y un receptor, ambos interpretarán su contenido de igual manera.
- Porque nos permite asegurar que el documento contiene toda la información declarada como obligatoria. Porque los datos vendrán en un formato conocido y correcto.

Se hablará a continuación de ambos casos.

DOCUMENTOS BIEN FORMADOS

Todo documento XML que quiera serlo, debe cumplir una serie de normas básicas:

- Al principio del documento se debe declarar una cabecera (en formato etiqueta) que indique lo siguiente:

  - Versión de XML. Actualmente se puede indicar la version 1.0 ó 1.1
  - Tipo de codificación utilizada. De esta manera se sabe que juego de caracteres se están utilizando. Lo normal es utilizar una codificación UTF-8, ISO-8859-1 o ISO-8859-15.
  - Si está basado en un documento de estructura externo (DTD o esquema). Si es así, se indicaría en el atributo *standalone="no"*. Por defecto es que no, con lo cual se suele omitir. Si el DTD estuviera embebido sería *yes*

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

- Todas las entidades, elementos y atributos deben tener una sintaxis correcta:

  - XML distingue entre mayúsculas y minúsculas
  - Los nombres de los elementos pueden ser alfanuméricos pero empezando obligatoriamente por letra.

    Elemento que se abra *<etiq>, deberá* cerrarse *</etiq>*.
  - Los valores que se indiquen en los atributos deberán ir entre comillas dobles o simples.
  - *<etiq atributo="1">*.
  - Si hay un elemento vacío *EMPTY, deberá* autocerrarse. *<etiq />*.

- En la estructura del documento XML, se encadenarán los elementos en un formato jerárquico, en el que solamente habrá un elemento raíz

- Si se usan entidades definidas por el usuario dentro del documento XML, en la DTD deberán estar declaradas.

DOCUMENTOS VALIDOS

Un documento XML válido es aquel que, además de estar bien formado, está acorde con la especificación de un DTD o esquema. Cuando se valida un documento XML se comprueban las siguientes cuestiones:

- Sólo se pueden utilizar elementos o atributos definidos en el DTD o esquema. Cualquier elemento no declarado hará que el documento no pase el proceso de validación (análisis léxico).

- Que los elementos y atributos estén en el orden definido en el DTD o esquema. Además se comprueba que los atributos o elementos obligatorios estén presentes dentro del documento (análisis sintáctico).

- Si se declaran valores que deben ser únicos, como por ejemplo, atributos ID, debe asegurarse que en todo el documento XML se cumple esta condición.

- Si se define un tipo concreto para los atributos o elementos, los valores que tomen deberán pertenecer al tipo declarado. No es lo mismo declarar un tipo entero que un tipo *string*.

HERRAMIENTAS PARA VALIDAR

Aunque se puede validar echando un simple vistazo en el documento XML, cuando los tamaños a manejar son grandes no es la mejor manera. En la actualidad existen muchas herramientas que nos permiten comprobar que se tienen documentos bien formados y que cumplen los criterios de validez. A continuación enumeraremos algunas herramientas bastante conocidas que permitirán realizar esta tarea de manera fácil y sencilla:

Vía web (la mayoría permite insertar en un formulario el documento XML con las DTD o esquemas):

- **http://xmlvalidation.com:** permite insertar en un formulario un fichero XML con las DTD incluidas o con un esquema externo y verifica la validez del documento.
- **https://www.validome.org/xml-rpc-validator**

Aplicaciones:

xmllint herramienta de línea de comandos que nos permite validar documentos xml contra dtds y xsds:

- ***xmllint --dtdvalid fichero.dtd fichero.xml***
- ***xmllint --schema fichero.xsd fichero.xml***

EJERCICIO 4.5

Verificar que los ejercicios 4.1-4.4 utilizando DTD's y esquemas son válidos

RESUMEN

En este capitulo se han prestado mucha atención a las ventajas de tener documentos bien formados y validos. Se ha clarificado la diferencia entre ellos (un documento válido es, de por si, un documento bien formado).

Se han comentado las dos formas más comunes de definir la estructura de un documento XML para poder validarlos:

Un DTD *(Document Type Definition).*
XSD *(XML Schema Definition)* o esquemas.

En cada un de ellos se ha explicado como declarar los elementos más destacados, incluyendo atributos y restricciones que permiten cambiar su ocurrencia, su secuencia o incluso su tipo de datos.

Finalmente se nombraron una serie de herramientas web y/o aplicaciones locales que aseguran que un documento XML esta bien formado y es válido con respecto a una gramática determinada. Estas herramientas permitirán comprobar que el trabajo realizado en un documento XML y asociado a una DTD o esquema es correcto o, por el contrario, indicará donde se han cometido los errores para solucionarlos.

EJERCICIOS PROPUESTOS

1. Crear un documento XML que permita definir el almacenamiento de coches de un concesionario de coches de segunda mano. Para ello, los campos a almacenar serían los siguientes:

Código de coche.
Marca. Modelo. Matricula.
Potencia (caballos).
Plazas.
Número de puertas.
- Crear un DTD y un XSD para que validen el documento

2. Una comunidad de propietarios, en su Junta General Ordinaria, decide crear un sistema de almacenamiento de la información para almacenar datos de la Comunidad. De esta manera podrán realizar las gestiones de manera más ágil y con un estándar de comunicación con el resto de acreedores y deudores de la Comunidad. En principio los campos a almacenar son los siguientes:

- Código de vecino.
- Nombre.
- Apellidos.
- Portal.
- Piso y letra.
- Código de Cuenta Corriente (CCC).
- Cargo (Presidente, Vicepresidente, Secretario, Vocal, Ninguno).
- Se pide lo siguiente:
    * Generar una DTD que defina esta estructura de información.
    * Generar un esquema que defina esta estructura de información.
    * Generar un documento XML, con al menos 10 vecinos. Dos de ellos tendrán el cargo de Presidente y Vicepresidente (al menos).

TEST DE CONOCIMIENTOS

1. ¿Qué permite definir una DTD?
   a) Define como se construye un documento XML válido. Para ello únicamente es necesario establecer una serie de reglas léxicas
   b) Define como se construye un documento XML válido. Para ello únicamente es necesario establecer una serie de reglas sintácticas
   c) Define como se construye un documento XML valido. Para ello únicamente es necesario establecer una serie de reglas léxicas y sintácticas
   d) Ninguna de las anteriores.
2. El elemento *<!ELEMENT BDsms (sms*)>:*
   a) El numero de ocurrencias del elemento *sms* varía entre 1 e infinito.
   b) El numero de ocurrencias del elemento *sms* varía entre cero e infinito.
   c) El numero de ocurrencias del elemento *sms* puede ser 0 ó 1.
   d) Ninguna de las anteriores.
3. El elemento *<!ELEMENT BDsms (sms+)>:*
   a) El numero de ocurrencias del elemento *sms* varía entre 1 e infinito.
   b) El numero de ocurrencias del elemento *sms* varía entre cero e infinito.
   c) El numero de ocurrencias del elemento *sms* puede ser 0 ó 1.
   d) *Ninguna de las anteriores.*
4. El elemento *<!ELEMENT BDsms (sms?)>:*
   a) El numero de ocurrencias del elemento *sms* varía entre 1 e infinito.
   b) El numero de ocurrencias del elemento *sms* varía entre cero e infinito.
   c) El numero de ocurrencias del elemento *sms* puede ser 0 ó 1.
   d) *Ninguna de las anteriores.*
5. En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado *<xs:all>*, significa:
   a) Que el orden de todos los hijos es indiferente siempre que sólo aparezcan una vez (en el caso de no cambiar la ocurrencia).
   b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
   c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
   d) Ninguna de las anteriores.
6. En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado *<xs:Choice>*, significa:
   a) Que el orden de todos los hijos es indiferente siempre que sólo aparezcan una vez (en el caso de no cambiar la ocurrencia).
   b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
   c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
   d) Ninguna de las anteriores.
7. En un esquema, si queremos establecer una secuencia de elementos hijos y se añade un indicador llamado *<xs:sequence>*, significa:
   a) Que el orden de todos los hijos es indiferente siempre que solo aparezcan una vez (en el caso de no cambiar la ocurrencia).
   b) Que el orden de todos los hijos es importante y debe aparecer en el documento XML en el indicado en la secuencia.
   c) Que el orden de todos los hijos es indiferente puesto que sirve únicamente para elegir uno de los hijos indicados.
   d) Ninguna de las anteriores.
8. En un esquema, si nos encontramos *<xs:whiteSpace value="preserve" />*, significa:

a) Que en el texto que se almacene podrá sustituir todos los espacios en blanco, tabuladores, saltos de linea y retornos de carro; por un único espacio en blanco.

b) Que en el texto que se almacene se mantendrá inalterado, aún cuando se tengan multitud de espacios en blanco, tabuladores, saltos de linea y retornos de carro.

c) Que en el texto que se almacene se eliminaran todos los espacios en blanco, tabuladores, saltos de linea y retornos de carro.

d) Ninguna de las anteriores.

9. Un documento está bien formado:

a) Si tiene una cabecera que indique que el documento es XML.

b) Si tiene una sintaxis correcta todas las entidades, elementos y atributos.

c) Si se tiene una estructura jerárquica en la que solo puede haber un nodo raíz

d) Si se han definido correctamente todas las entidades, elementos y atributos en una DTD o esquema.

e) Todas las anteriores.

f) Ninguna de las anteriores.

10. Un documento es valido:

a) Si esta bien formado.

b) Si en el análisis de validez del documento con respecto a una DTD o esquema se cumplen todas las reglas sintácticas que en él se definen.

c) A y B, en su conjunto definen un documento válido.

d) A y si se tiene una estructura jerárquica con multinodo raíz

e) Todas las anteriores.

f) Ninguna de las anteriores.

APÉNDICE

Añadir más funciones xml a eclipse
En Help->Install new software->"Available Software Sites"
Marcar: http://download.eclipse.org/webtools/...
En Web, XML... marcar:

- Eclipse web developer tools
- Eclipse xml editor and tools
- Eclipse xsl developer tools