IES Puerto de la Cruz Telesforo Bravo

CICLO FORMATIVO DE GRADO SUPERIOR:

"DESARROLLO DE APLICACIONES WEB"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)

Índice

TEMA 7.- XML SCHEMA. 3

1.- INTRODUCCIÓN A XML SCHEMA. 3
2.- NAMESPACES 5
2.1.- Enlazar XSD con el XML. 5
2.2.- Declaración del esquema. 6
3.- DEFINIR ELEMENTOS SIMPLES. 8
3.1.- Valor por defecto y valor fijo. 10
3.2.- Restricciones sobre el rango de valores. 10
3.2.- Restricciones sobre un conjunto de valores. 10
3.3.- Listado de restricciones soportadas. 11
3.4.- Restricciones por patrones: expresiones regulares. 12
3.5.- Restricciones sobre los espacios en blanco. 13
3.6.- Restricciones sobre la longitud del valor. 13
4.- ELEMENTOS COMPLEJOS. 14
4.1.- Definición de tipos de datos propios. Herencia. 15
4.2.- Definición de atributos. 16
4.3.- Elementos complejos vacíos. 17
4.4.- Elementos complejos que contienen sólo texto. 18
4.5.- Elementos complejos mixtos. 18
5.- LOS INDICADORES. 19
5.1.- Indicador All. 20
5.2.- Indicador Choice. 20
5.3.- Indicador Sequence. 20
5.3.- Indicadores de ocurrencia. 21
5.4.- Indicadores de agrupamiento. 21

TEMA 7.- XML Schema.

1.- Introducción a XML Schema.

Como definición formal, un esquema XML (XML Schema) es un conjunto de reglas que describe una **clase** de documentos XML. Indica qué elementos pueden aparecer, qué atributos puede tener cada elemento, etc. En general describe la estructura de los documentos XML indicando las subetiquetas de cada elemento, la secuencia de las mismas, los tipos de datos soportados, etc.

Ya hemos aprendido a crear DTDs en el tema anterior. Los DTD son, en realidad, un tipo concreto de esquema XML.

Como puntos débiles los DTD presentan los siguientes:

- No podemos definir tipos de datos. Es imposible determinar que una etiqueta <edad> solo puede contener datos numéricos, por ejemplo.
- Las DTDs requieren una sintaxis especial a la hora de la escritura, y esta forma de escribir código está muy alejada del propio XML.

Los esquemas XML tienen dos cometidos fundamentales:

- Publicar cómo se han de construir documentos XML correctos (bien formados).
- Permitir la validación de un documento conforme a un esquema particular (para, por ejemplo, comprobar que los datos que nos llegan están en el formato correcto). Se habla entonces de que el documento XML es **válido** respecto a un esquema, lo que no debe confundirse con estar bien formado.

Aunque existen diversos estándares para la escritura de los esquemas, los más populares han sido dos:

- DTD (Document Type Definition).
- XSD (Xml Schema Definition).

Si bien al comienzo se popularizó el uso de los DTD, en la actualidad éstos están perdiendo terreno frente a los XSD. Y es que XSD tiene una ventaja evidente:

Una ventaja de XSD frente a DTD:
Se escribe en formato XML. No hay que aprender un nuevo lenguaje para usarlo.

Los archivos XML Schema:

- Define los elementos que pueden aparecer en un documento.
- Define los atributos que pueden aparecer en un documento.
- Define qué elementos son elementos hijos en la jerarquía.
- Define el orden de los elementos hijo.
- Define la cantidad de elementos hijos (no soportado en DTD).
- Define si un elemento es vacío (empty) o puede incluir texto.
- Define los tipos de datos para los elementos y los atributos (no soportado en DTD).
- Define los valores por default y fijos para los elementos y los atributos (no soportado en DTD para los elementos).
- Define los espacios de nombres que se pueden utilizar (**namespaces**) (no soportado en DTD).
- Permite asociar restricciones a los elementos (**facetas**) (no soportado en DTD).
- Permite asociar patrones de construcción en los datos (**expresiones regulares**) (no soportado en DTD).

El estándar está definido en **http://www.w3.org/XML/Schema**, cuya primera versión se aprobó el 2 de mayo de 2001 y la segunda versión fue publicada el 28 de octubre de 2004.

Ejemplo:
<date type=”date”>2010-11-03</date>
Existe el tipo de datos “date” en XML cuya sintaxis por defecto es AAAA-MM-DD, con lo que podremos leer estos valores de otras aplicaciones, como pueden ser las bases de datos.

Otra gran ventaja de XSD frente a DTD:
permite asociar tipos de datos a los elementos.

Comparemos un ejemplo entre DTD y XML-Schema:

Fichero mensaje.xml:
```
<?xml version=”1.0”?>
<mensaje>
    <remitente>Juan</remitente>
   <destinatario>María</destinatario>
   <asunto>Reunión:</asunto>
   <cuerpo>Mañana a las 11:00 horas</cuerpo>
</mensaje>
```

Fichero mensaje.dtd:
```
<!ELEMENT mensaje (remitente, destinatario, asunto, cuerpo)>
<!ELEMENT remitente (#PCDATA)>
<!ELEMENT destinatario (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
```

Fichero mensaje.xsd:
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace=”http://www.misitio.com”     xmlns="http://www.misitio.com"
elementFormDefault="qualified">

<xs:element name="mensaje">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="remitente" type="xs:string"/>
      <xs:element name="destinatario" type="xs:string"/>
      <xs:element name="asunto" type="xs:string"/>
      <xs:element name="cuerpo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

2.- Namespaces .

2.1.- Enlazar XSD con el XML.

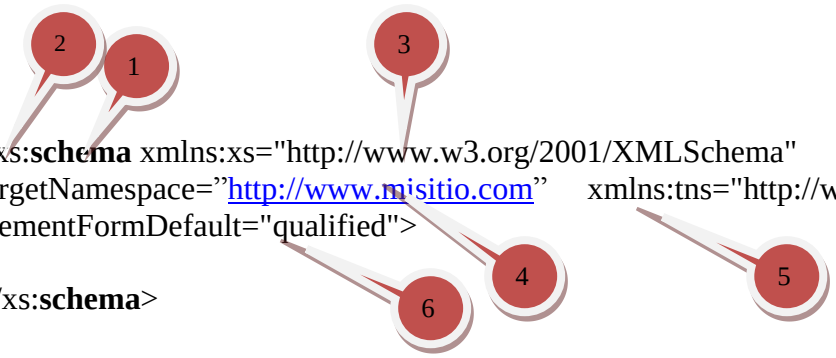
En el ejemplo anterior si queremos enlazar el esquema mensaje.xsd con un documento XML tendremos que hacer lo siguiente:

Fichero mensaje.xml:
```
<?xml version=”1.0”?>
<mensaje xmlns="http://www.misitio.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.misitio.com mensaje.xsd">
    <remitente>Juan</remitente>
   <destinatario>María</destinatario>
   <asunto>Reunión:</asunto>
   <cuerpo>Mañana a las 11:00 horas</cuerpo>
</mensaje>
```

Como ves en el cuadro anterior se enlaza el archivo .xsd en el elemento raíz de nuestro XML, que en nuestro ejemplo se llama **mensaje**. En el siguiente apartado veremos cada uno de los apartados.

2.2.- Declaración del esquema.

Comenzaremos analizando el código inicial del fichero .xsd:




```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.misitio.com"      xmlns:tns="http://www.misitio.com"
elementFormDefault="qualified">

</xs:schema>
```

Punto 1: siempre comenzamos definiendo el elemento llamado **schema**. Todo nuestro fichero estará dentro de este elemento. Antes de él estamos usando un **prefijo** (es lo que está antes de los dos puntos – xs).

Punto 3: definimos el atributo **xmlns** que significa XML-Namespace, o espacio de nombres XML y a este espacio de nombres le asociamos el identificador xs. Esto significa que todas las etiquetas que empiecen por el prefijo "xs:" están referidas al espacio de nombres del w3c.

Nota: hay muchos programadores que usan como buena práctica el prefijo "xsd", pero en sí mismo es indiferente.

Punto 2: En este ejemplo el **prefijo** que hemos utilizado lo hemos llamado "xs".

Punto 4: Al usar targetNamespace estamos diciendo que en este documento de esquema se está utilizando el espacio de nombres llamado "http:ww.misitio.com". En esta dirección web no tiene porqué haber ningún recurso, es más, no tiene porqué existir. Es sólo una manera de definir un espacio de nombres (usando URIs que no colisionen entre sí). Al final el significado de esta parte indica que todas las etiquetas que creemos en este fichero (como son mensaje, destinatario, remitente, cuerpo y asunto) pertenecerán a este espacio de nombres.

Punto 5: Aquí estamos diciendo que http://misitio.com es el espacio de nombres que se aplicará por defecto. Es decir, cuando no exista prefijo en una etiqueta nos estaremos refiriendo a nuestro espacio de nombres. Nosotros podemos asociarle un prefijo también. Por ejemplo: xmlns:tns="http://misitio.com". El prefijo ahora será tns.

Punto 6: los espacios de nombres por defecto pueden ser "qualified" o "unqualified". Si ponemos qualified estaremos obligados a poner prefijo en todas las etiquetas. Si ponemos unqualified las etiquetas que no tengan prefijo se entienden dentro del espacio de nombres por defecto.

Para que entiendas mejor el uso de **namespaces** observa el siguiente texto:

"Estamos escribiendo una página en Xhtml y dentro de ella queremos editar ecuaciones matemáticas. Para esto usaremos otro lenguaje de marcas: el MathML, que también es un lenguaje creado en XML y por tanto tiene la misma sintaxis. ¿Qué ocurriría si en ambos lenguajes hay una etiqueta con el mismo nombre? ¿Cómo sabe el navegador cuando encuentra una etiqueta a qué estándar pertenece?"

Los espacios de nombres resuelven esta circunstancia. Consisten en identificar **mediante un prefijo** el ámbito de las etiquetas. Al comienzo del documento, en el elemento raíz, identificaremos qué espacios de nombres estamos usando. Observa el siguiente código (utiliza Xhtml, MathML y SVG).

```
<html xmlns:xhtml="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg"
      xmlns:math="http://www.w3.org/1998/Math/MathML"
      xml:lang="es">
   <head>… </head>
    <body>
        <h1>Veamos un ejemplo escrito en SVG</h1>
        <svg:svg width="600px" height="600px">
        <svg:desc>Un cuadrado y un círculo</svg:desc>
        <svg:defs>
           <svg:rect id="rectangulo" width="200" height="200"
fill="red" x="10" y="10" stroke="black" />
           <svg:circle id="circulo" r="100" cx="200" cy="200"
fill="white" stroke="black" />
        </svg:defs>
        </svg:svg>
        <h1>Veamos un ejemplo escrito en MathML</h1>
        <math:mrow>
          <math:apply>
            <math:eq/>
            <math:ci>A</math:ci>
            <math:matrix>
                <math:matrixrow>
                    <math:ci>x</math:ci>
                    <math:ci>y</math:ci>
                </math:matrixrow>
                <math:matrixrow>
                    <math:ci>z</math:ci>
                    <math:ci>w</math:ci>
                </math:matrixrow>
            </math:matrix>
          </math:apply>
        </math:mrow>
        </body>
    </html>
```

Como ves al comienzo del documento definimos los espacios de nombres que vamos a usar con **xmlns** seguido de dos puntos y el **prefijo asignado** y después un URL que actúa como identificador del espacio de nombres.

Declaración del espacio de nombres:

xmlns:prefijo="nombre"

El prefijo lo utilizaremos luego en las etiquetas XML. El nombre es una URI. Veamos un ejemplo:

Declaramos el espacio de nombres:
xmlns:milenguaje=”http://www.miservidor/milenguaje”
xmlns:otrolenguaje=”http://miservidor/milenguaje”
En el fichero XML podremos hacer:
<milenguaje:h3> Hola </milenguaje:h3>
<otrolenguaje:h3> Hola </otrolenguaje:h3>

Para que este código funcione se debe haber definido el elemento h3 en los dos lenguajes.

Debes saber:

- La URI de un namespace no tiene que existir. En realidad, aunque se trate de un URL solo actúa como un identificador dentro de la página. Y como identificador debe ser único.
- Las aplicaciones que procesan XML se fijan en estas URI y no en los prefijos

PRACTICA 01.- Prueba en Editix el fichero mensajes.xsd. Chequéalo hasta que sea válido. Cuidado al copiar las comillas dobles de este PDF al Editix.

Después prueba mensajes.xml. Enlázale el esquema y valida el resultado.

3.- Definir elementos simples.

Los elementos simples sólo pueden contener texto. No pueden contener otros elementos ni atributos. Pueden llevar un tipo asociado, que puede ser un tipo básico definido en XML Schema o bien un tipo de datos creado por nosotros (ya veremos cómo hacerlo). La sintaxis que siguen es la siguiente:

```
<xs:element name=”nombre” type=”tipo”>
```

Los tipos básicos más utilizados son los siguientes:

- xs:string → una cadena de texto
- xs:decimal → un número real, puede tener parte entera y parte fraccional.
- xs:integer →un número entero.
- xs:boolean → un valor “true” o “false”.
- xs:date → una fecha
- xs:time → una hora.

Es importante resaltar que **el orden sí importa**. Es decir, el orden en que se escriben los elementos en el esquema será el orden en que deben aparecer en el documento XML válido.

Lista de tipos de datos:

Tipo en XSD	¿Qué significa?	Tipo equivalente en programación en la plataforma .NET de Microsoft
anyType	Cualquier tipo es admitido. Es un objeto genérico.	object
Boolean	Valor del tipo verdadero/falso	bool
Byte	Un byte de memoria (de -127 a 127). El bit más significativo es para el signo.	sbyte
date | dateTime | time	Una fecha.	DateTime
decimal	Un número real, con parte fraccional.	decimal
Doublé	Un número real con mayor precisión (utiliza más bytes que decimal).	double
duration	Una duración (intervalo de tiempo)	Timespan
Float	Un tipo real con un poco menos de precisión que double. Está representado en formato de coma flotante.	single
ID | Name	Una cadena de texto.	string
Int	Un número entero (positivo o negativo).	Int32
integer | long	Un número entero largo (8 bytes).	Int64
Short	Un número entero corto (2 bytes).	Int16
String	Una cadena de texto.	string
unsignedByte	Un byte sin signo (De 0 a 255)	Byte
unsignedInt	Un entro sin signo (4 bytes). Entero positivos.	UInt32
unsignedLong	Un entero sin signo largo (8 bytes).	UInt64
unsignedShort	Un entero sin signo corto (2 bytes).	UInt16

Ejemplo: definimos la etiqueta <edad>, que sólo podrá contener un número entero:

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

PRACTICA 02.- Basándote en el fichero de la práctica 1, crea un esquema donde se defina el elemento "persona", compuesto por una secuencia de los siguientes elementos simples: nombre (cadena), apellidos (cadena), edad (entero), fechanacimiento (fecha).

Después crea un fichero .xml que utilice este esquema. Rellénalo con datos y valida el resultado.

Finalmente prueba a poner "Maria" en la etiqueta edad y valida el documento. ¿Qué error se obtiene".

¿Y si ponemos "Arturo" en la etiqueta fechanacimiento?

3.1.- Valor por defecto y valor fijo.

Podemos usar los atributos **default** y **fixed** dentro de un elemento para indicar su valor por defecto o que el elemento siempre tiene un valor fijo. Ejemplo:

```
<xs:element name="color" type="xs:string" default="red"/>
        ó
<xs:element name="color" type="xs:string" fixed="red"/>
```

PRACTICA 03.- En el código de la práctica anterior agrega un valor por defecto al campo edad estableciéndolo en 18 años.

Después de probar la validación indica que los 18 años es un valor fijo y no por defecto. ¿Qué ocurre si colocas otro dato en edad?

3.2.- Restricciones sobre el rango de valores.

Dentro de los elementos simples sí que podemos indicar **restricciones,** también llamadas en el mundo de XML como **facetas**. Veremos varios tipos de restricciones. El primer tipo es una restricción de los valores posibles: es una restricción soportada por algunos tipos de datos, como pueden ser los números. Observa el ejemplo: queremos restringir la edad a los valores 0-100:

```
<xs:element name="edad">
<xs:simpleType>  ← Observa que dentro del elemento indicamos que es un tipo simple
<xs:restriction base="xs:integer">  ← Aquí definimos la restricción en el tipo de datos
<xs:minInclusive value="0"/>
<xs:maxInclusive value="100"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Usamos minInclusive (valor mínimo) y maxInclusive (valor máximo).

PRACTICA 04.- Continuando con la gramática de la persona define un rango de edad entre 18 y 50 años. Prueba el resultado indicando en el XML que la persona tiene 63 años. ¿Qué error se obtiene?

3.2.- Restricciones sobre un conjunto de valores.

En el siguiente ejemplo veremos que podemos restringir un elemento para que acepte únicamente una lista de valores. Definamos el elemento coche:

```
<xs:element name="coche">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Toyota"/>
      <xs:enumeration value="Mercedes"/>
      <xs:enumeration value="Peugeot"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Aquí el elemento coche sólo podrá ser Toyota, Mercedes o Peugeot. No se admitirán otros valores dentro de la etiqueta.

PRACTICA 05.- Agrega a la práctica anterior un elemento llamado estadocivil cuyos valores pueden ser "soltero", "casado", "separado", "divorciado" o "viudo". Prueba el resultado con diferentes valores. ¿Se obtiene error cuando el valor escrito en el XML no está en la lista?

3.3.- Listado de restricciones soportadas.

Ya hemos visto los dos primeros tipos de restricciones. Hay muchas y nos centraremos en las más utilizadas, pero aún así mostramos en la siguiente tabla el conjunto de restricciones soportadas:

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

La restricción maxExclusive funciona igual que maxInclusive pero en ese caso nos serviría, por ejemplo, para indicar que la edad es **menor que 100**. MaxInclusive puede interpretarse como **menor o igual que 100**.

Sigamos viendo las más utilizadas.

3.4.- Restricciones por patrones: expresiones regulares.

Las **expresiones regulares** se utilizan en muchos lenguajes de programación. Son en sí mismas casi un lenguaje pues tienen un conjunto de caracteres que significan cosas concretas. Una expresión regular sirve para definir un patrón de texto. Veamos ejemplos:

[a-z] → un texto será válido si es una única letra minúscula. El guión alto es un rango.

[a-zA-Z0-9] → una letra minúscula más una mayúscula más un número.

[0-9]* → cero o más números. [0-9]+ → uno o más números.

[A|Z]? → cero o una veces la letra A mayúscula ó la B mayúscula.

[A-Z][A-Z][A-Z] → exactamente tres letras mayúsculas.

[0-9]{2} → exactamente dos números.

[(a-c)|(x-z)]{5} → cinco letras que pueden estar entre la a y la c, o bien, entre la z y la x.

Veamos ejemplos del uso de las expresiones regulares.

```
<xs:element name="codigopostal"> <!—son 5 dígitos -->
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="iniciales">  <!—son tres letras separadas por punto -->
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z].[a-zA-Z].[a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="sexo">  <!—es hombre ó mujer -->
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="hombre|mujer"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="password"> <! – ocho letras, mayúsculas o minúsculas -->
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

PRACTICA 06.- Agrega a la práctica anterior un elemento llamado dni que puede contener ocho números más una letra mayúscula. Utiliza una expresión regular. Prueba el resultado con diferentes valores.

3.5.- Restricciones sobre los espacios en blanco.

Recuerda que el XML considera un espacio en blanco no sólo el carácter de espacio en blanco, sino también los tabuladores y los saltos de línea. Con los espacios en blanco podemos hacer tres cosas: preservarlos = dejarlos como estén (**preserve**), colapsarlos (**collapse**) = contraer todos los que estén seguidos en uno solo, o por último reemplazarlos (**replace**) (lo que hace es que los tabuladores y saltos de línea se reemplazan por espacios en blanco. Veamos un ejemplo de uso (para los otros dos casos basta con cambiar la palabra reservada en negrita):

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

3.6.- Restricciones sobre la longitud del valor.

Tenemos varias opciones. Sirven para acotar el número de caracteres de texto, por ejemplo, usando **lenght** (que indica el tamaño del texto en caracteres), **minLenght y maxLength** (para indicar texto entre un número minimo y máximo de caracteres). También podemos acotar los dígitos de un número con **totalDigits.** En el caso de números reales con parte fraccional podemos acotar esta parte con **fractionDigits**.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

PRACTICA 07.- Modifica la práctica anterior para que la etiqueta apellidos sólo soporte un tamaño de 10 caracteres. Por otra parte el nombre puede contener entre 3 y 8 caracteres.

4.- Elementos complejos.

Un elemento complejo es un elemento XML que contiene otros elementos y/o atributos.

Existen cuatro tipos de elementos complejos:

- Elementos vacíos.
- Elementos que contienen solo otros elementos.
- Elementos que solo contienen texto.
- Elementos que contienen tanto texto como otros elementos.

Ejemplo: definimos el elemento complejo alumno que contiene nombre y apellidos:

Definición de un elemento complejo **que sólo contiene otros elementos**:

```
<xs:element name="alumno">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Uso en XML:

```
<alumno>
  <nombre>Juan</nombre>
  <apellido>González</apellido>
</alumno>
```

Esto mismo podríamos hacerlo creando nuestro **propio tipo de datos**, que podría reutilizarse cuando fuera necesario:

Definición:

```
<xs:element name="alumno" type="tipopersona"/>
<xs:element name="trabajador" type="tipopersona"/>

  <xs:complexType name="tipopersona">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Uso en XML:

```
<alumno>
  <nombre>Juan</nombre>
  <apellido>González</apellido>
</alumno>

<trabajador>
   <nombre>Ana</nombre>
   <apellido>Delgado</apellido>
</trabajador>
```

4.1.- Definición de tipos de datos propios. Herencia.

Para que nuestros esquemas sean más sencillos podemos usar la etiqueta xs:complexType para definir nuestros propios tipos de datos. En el ejemplo anterior hemos visto cómo crear el tipo llamado "tipopersona". Esto lo conseguimos agregando un atributo "name" a xs:complexType.

PRACTICA 08.- Crea un .xsd donde definimos el elemento coche que será de tipo "datoscoche". Debe contener los elementos marca (cadena), modelo (cadena), puertas (entero positivo) y color (enumeración de "blanco", "gris", "rojo", "verde" y "azul".

Otra de las ventajas de XML Schema frente a las DTD es que no sólo permite la creación de tipos de datos propios, sino que también permite **extender** un tipo de datos creado. Esto se conoce como **herencia**, ya que podremos crear un tipo de datos a partir de otro existente. Supón que tenemos el tipo de datos "datospersona" que contiene nombre y apellido. Y queremos crear un tipo de datos llamado "datoscontacto" que contenga nombre, apellido, direccion, ciudad y pais. Lo haríamos así:

```
<xs:element name="alumno" type="datoscontacto"/>

<xs:complexType name="persona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="datoscontacto">
<xs:complexContent>
<xs:extension base="persona">
   <xs:sequence>
    <xs:element name="direccion" type="xs:string"/>
    <xs:element name="ciudad" type="xs:string"/>
    <xs:element name="pais" type="xs:string"/>
  </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

En este ejemplo el tipo "datoscontacto" **extiende** todo el contenido del tipo "persona". Al decir en la primera línea que la etiqueta alumno es de tipo "datoscontacto" tendrá los elementos de "persona" más los elementos de "datoscontacto".

PRACTICA 09.- Utilizando el .xsd de la práctica anterior crea el tipo de datos "remesa" que extiende "datoscoche". Este tipo de datos nos servirá para indicar una remesa de coches que viaja por barco desde el país productor al país comprador. Incluirá las etiquetas numcoches (entero), nombrebarco (cadena), paisproductor (cadena) y paiscomprador (cadena). El elemento principal de este esquema será "compra" de tipo "remesa".

Crea un fichero xml con los datos de una remesa de 100 Toyota LandCruiser, de color gris, 3 puertas. Vienen desde Japón a España en el barco llamado "SeaAdventurer".

4.2.- Definición de atributos.

Los atributos no pueden ser definidos dentro de elementos simples. Se deben definir dentro de elementos complejos. La sintaxis para su definición es la misma que para los elementos simples, sustituyendo la palabra element por **attribute**.

```
<xs:attribute  name="nombre"  type="tipo">
```

A diferencia de las DTD cuando definimos un atributo no le enlazamos el elemento. Sabemos que este atributo se declara con un elemento concreto porque se coloca dentro del mismo.

Los atributos tienen que al final de una etiqueta xs:complexType. A diferencia de los elementos, los atributos pueden aparecer en cualquier orden y no pueden incluir otros elementos (al ser de tipos simples). No obstante, su característica más interesante es que pueden ser opcionales u obligatorios y se les puede asignar un valor por defecto o un valor fijo. Mira el ejemplo:

Definimos un atributo idioma:

```
<xs:attribute name="idioma" type="xs:string"/>
```

Y si estuviera dentro de un elemento llamado "texto" lo usaríamos así:

```
<texto idioma="inglés">What's your name?</texto>
```

Modificadores de los atributos:

Valor por defecto:

```
<xs:attribute name="idioma" type="xs:string" default="Spanish">
```

Valor fijo:

```
<xs:attribute name="idioma" type="xs:string" fixed="Spanish">
```

Valor obligatorio:

```
<xs:attribute name="idioma" type="xs:string" use="required">
```

Valor opcional:

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

4.3.- Elementos complejos vacíos.

Los elementos complejos vacíos son aquellos que no pueden contener ni texto ni otros elementos. Un ejemplo sería la etiqueta
 en xhtml. Lo único que pueden contener estos elementos son atributos. Veamos un ejemplo:

Definimos el elemento "componente" que tiene dos atributos, el primero se llama "numeroserie" y es de tipo entero. El segundo se llama "modelo" y es una cadena de texto.

```
<xs:element name="componente">
     <xs:complexType>
         <xs:attribute name="numeroserie" type="xs:integer"/>
         <xs:attribute name=”modelo” type=”xs:string”/>
    </xs:complexType>
</xs:element>
```

- Un posible uso en XML sería:

```
<componente numeroserie=”382299988” modelo=”impresora HP-6215”/>
```

También podríamos haberlo hecho definiendo un tipo de datos:

```
<xs:element name="componente" type=”tipocomponente” />
 <xs:complexType name=”tipocomponente”>
    <xs:attribute name="numeroserie" type="xs:integer"/>
    <xs:attribute name=”modelo” type=”xs:string”/>
 </xs:complexType>
```

PRACTICA 10.- Crea un esquema que defina el elemento "pasaje". Se trata de un elemento vacío que guarda datos de un pasaje de avión. Sus atributos son:

1) Asiento de tipo cadena. Es un atributo obligatorio.
2) Ventanilla de tipo booleano con valor por defecto "true".
3) Precio de tipo número real. Es un atributo opcional.
4) Moneda: atributo de tipo cadena con valor por defecto "euros".

Después crea un documento xml válido con un pasaje cuyo asiento es el 18D, no está en ventanilla y costó 98,16 dólares.
Prueba a quitar el atributo asiento. Prueba a quitar el atributo precio. Prueba a quitar el atributo ventanilla. ¿En qué casos da error?

4.4.- Elementos complejos que contienen sólo texto.

Serán elementos que no pueden contener ningún otro elemento, sino texto. En principio sería elementos simples pero aquí queremos que tengan atributos. Mira el siguiente elemento xml:

<talla sistema=”europeo”>38</talla>

En este caso, al contener un atributo, no podemos declarar un elemento simple. Tendremos que usar xs:complexType. Dentro de complex type indicamos un tipo base y al final agregamos los atributos que necesitemos. Observa el esquema:

```
<xs:element name="talla">
  <xs:complexType>
  <xs:simpleContent>
    <xs:extension base="xs:integer">  → aquí indicamos de qué tipo es el texto
      <xs:attribute name="sistema" type="xs:string" /> → aquí va el/los atributo/s
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
```

PRACTICA 11.- Crea un esquema que defina el elemento “parrafo”. Se trata de un elemento que sólo contendrá un texto. Sus atributos son:

5) idioma de tipo cadena. Es un atributo obligatorio.
6) codificación de tipo cadena. Es un atributo opcional.

Después crea un documento xml válido con un párrafo escrito en español y codificación UTF-8 cuyo contenido es:

En un lugar de la mancha de cuyo nombre no quiero acordarme…

4.5.- Elementos complejos mixtos.

Este tipo de elementos puede contener tanto texto como otros elementos. Es más, puede haber incluso texto intercalado entre los diferentes elementos. Mira el ejemplo:

<boletin>
El alumno <nombre>Juan</nombre><apellidos>González Delgado</apellidos>, ha obtenido una calificación de <nota>8</nota> en la asignatura <materia>Lenguajes de Marcas</materia>.
</boletin>

Para hacer el esquema que defina el elemento boletín tendremos que indicarle que es un elemento **mixto**. Observa el esquema:

```
<xs:element name="boletin" type="tipoBoletin"/>
<xs:complexType name="tipoBoletin" mixed="true">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="nota" type="xs:positiveInteger"/>
    <xs:element name="materia" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Vamos a aprovechar la siguiente práctica para agregar algo de complejidad: definiendo restricciones en los atributos (XML Schema lo soporta casi todo…).

PRACTICA 12.- Crea un esquema que defina el elemento "boletin" de tipo mixto que está escrito en esta misma página. Después crea un fichero xml donde se exponga que el alumno Juan González Delgado obtuvo un 8 en Lenguajes de marcas. Valida el documento.

Ahora vamos a complicar un poco el asunto. A la etiqueta boletín agrégale un atributo llamado "evaluacion", de tipo entero. En el xml indica que el boletín corresponde a la primera evaluación.

Y ahora la complicación final. Queremos restringir el atributo evaluación de forma que sólo pueda valor 1, 2 o 3. Ningún otro valor. ¿Se te ocurre cómo hacerlo? Una buena pista es dejarse llevar por la escritura automática en Editix.

La solución la discutimos en clase.

5.- Los indicadores.

Hasta ahora hemos hecho esquemas donde se define sólo un elemento principal. Nos falta la parte de jerarquía. Los ejercicios realizados con DTD tenían más complejidad en este nivel.

Existen 7 tipos de indicadores:

Indicadores para el orden:

All Choice Sequence

Indicadores de ocurrencia:

maxOccurs minOccurs

Indicadores de grupo:

Group name attributeGroup name

5.1.- Indicador All.

El indicador **all** especifica que los elementos hijos pueden aparecer en cualquier orden y que aparecen como mucho solo una vez.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:all>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

PRACTICA 13.- Con la definición de este esquema para el elemento "persona" prueba que en xml podemos alterar el orden de nombre y apellido sin problemas.

5.2.- Indicador Choice.

El indicador **choice** especifica que de los elementos hijos sólo puede aparecer uno. Es el equivalente al | en las DTD.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:choice>
      <xs:element name="alumno" type="xs:string"/>
      <xs:element name="profesor" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

En este ejemplo dentro de <persona> sólo podremos tener <alumno> o bien <profesor>, pero no ambos.

5.3.- Indicador Sequence.

El indicador **sequence** lo hemos estado utilizando en muchos ejemplos. Define una la secuencia exacta en que deben aparecer los elementos hijo.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

En este caso debe aparecer primero el elemento <nombre> seguido del elemento <apellido>.

5.3.- Indicadores de ocurrencia.

Sirven para indicar el número mínimo (minOccurs) y máximo (maxOccurs) de veces que puede aparecer un elemento hijo. Podemos usarlos juntos o por separado. Imagina un ejemplo: una persona puede tener 0, 1 ó 2 abuelos y hasta 5 hijos:

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
      <xs:element name=”abuelo” type=”xs:string” minOccurs=”0” maxOccurs=”2”/>
      <xs:element name=”hijo” type=”xs:string” maxOccurs=”5”/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

PRACTICA 14.- Define en un esquema el elemento <clinica>. Está compuesta por un máximo de 500 <paciente>. Cada paciente tiene un nombre, un apellido y de cero a dos <visitante>. Cada visitante también tiene nombre y apellido, con lo que es recomendado hacer el tipo persona.

Crea un fichero .xml con los datos de una clínica que tiene 3 pacientes y sus visitantes.

5.4.- Indicadores de agrupamiento.

Sirven para agrupar elementos o atributos bajo una misma denominación. Así podremos utilizar este agrupamiento en cualquier otra parte de nuestro esquema.

Group permite agrupar bajo un nombre un conjunto de elementos:

```
<xs:group name="grupoPersona">
<xs:sequence>
<xs:element name="nombre" type="xs:string"/>
<xs:element name="apellido" type="xs:string"/>
<xs:element name="cumpleanos" type="xs:date"/>
</xs:sequence>
</xs:group>
```

Una vez creado el grupo se puede utilizar en otra definición. Para ello usaremos la palabra **ref**.

```
<xs:element name="persona" type="infoPersona"/>
<xs:complexType name="infoPersona">
<xs:sequence>
<xs:group ref="grupoPersona"/>
<xs:element name="pais" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

Podemos hacer algo similar pero agrupando atributos. Para esto se utiliza la palabra **attributeGroup.**

```
<xs:attributeGroup name="grupoAttrPersona">
<xs:attribute name="nombre" type="xs:string"/>
<xs:attribute name="apellidos" type="xs:string"/>
<xs:attribute name="cumpleanos" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="persona">
<xs:complexType>
<xs:attributeGroup ref="grupoAttrPersona"/>
</xs:complexType>
</xs:element>
```

PRACTICA FINAL.- Crea un .xsd para nuestra clínica veterinaria.

- clinicaveterinaria es una colección de cero o más cliente.
- clinicaveterinaria tiene un atributo llamado codigo que es de tipo cadena obligatorio.
- Un cliente tiene un nombre, un apellido opcional y uno o más animal.
- Tanto nombre como apellido son cadenas de texto.
- animal puede ser un perro, un gato o un reptil.
- los perros, gatos y reptiles tienen nombre, edad y raza.
- La edad es un entero comprendido entre 0 y 15 años.
- la raza es una cadena de texto.
- Los perros, gatos y reptiles tienen un atributo llamado sexo cuyos valores pueden ser H (hembra) o M (macho). El valor por defecto es M.

Crea un fichero .xml con los datos de una clínica que tiene 3 pacientes y sus visitantes.

IES Puerto de la Cruz
Telesforo Bravo

CICLO FORMATIVO DE GRADO SUPERIOR:

"DESARROLLO DE APLICACIONES WEB"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)