CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

TEMA 7 (II)
XSD. ESQUEMAS Y VOCABULARIOS.

LM
CFGS DAW

Autor: Pascual Ligero

Revisado por:

Fco. Javier Valero – franciscojavier.valero@ceedcv.es

2019/2020

Versión:191128.1103

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

ÍNDICE DE CONTENIDO

UD07. XSD. ESQUEMAS Y VOCABULARIOS

1. INTRODUCCIÓN A LA VALIDACIÓN DE DOCUMENTOS MEDIANTE ESQUEMAS

En el ámbito de las tecnologías XML, un esquema nos describe la estructura que puede tener un documento XML para que pueda ser válido.

En la 1ª parte de la unidad ya hemos usado un sistema para validación de documentos, los DTD, mediante los cuales hemos podido describir la sintaxis y la estructura de los documentos XML que podían así convertirse en documentos válidos.

Pero los DTDs en algunas ocasiones pueden resultar insuficientes para describir completamente la estructura y características de los documentos XML, y además no son documentos XML en sí mismos

Con los esquemas podremos filtrar los tipos de datos que pueden utilizar los elementos y atributos de los documentos, cosa que con los DTDs no podíamos hacer.

1.1 EJEMPLO

Antes de ver teóricamente los conceptos que forman parte del esquema, vamos a ver un pequeño ejemplo para hacernos una idea general de lo que será la validación de documentos con XSD Si partimos de un ejemplo de películas con un solo elemento raíz llamado <pelicula> que a su vez contiene únicamente el nombre de la película dentro de la etiquetá <titulo> podríamos considerar el siguiente documento XML, que por variar de estilos nos describirá la película de Casablanca:

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula>
        <titulo>Casablanca</titulo>
</pelicula>
```

Un esquema básico que pudiese describir la estructura de este documento y por lo tanto validarlo sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="pelicula">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="titulo" type="xs:string"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Podemos observar que sigue las normas de cualquier documento XML, pero tendrá la extensión **.xsd** que es la correspondiente a los esquemas. Por tanto, lo guardaremos como peliculas.xsd

NOTA: esta versión de presentación del esquema, aunque es correcta, no hace uso de los espacios de nombres o namespace que posteriormente comentaremos.

Para poderlo utilizar y validar la información, tenemos que **modificar el documento XML** que contenía los datos de nuestra película, añadiendo la referencia al fichero que contiene el esquema (.xsd) e incluyéndolo como un atributo en el elemento raíz (en nuestro caso <pelicula>) y que aparece sombreado:

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

Este fichero lo guardaremos como **peliculas.xml**

A partir de estos elementos básicos, podemos validarlo como hacíamos con los DTDs.

Suponemos que ambos documentos **(.xml y .xsd) están en el mismo directorio**, ya que si no fuese así se tendría que especificar su ubicación.

Como resultado obtenemos el mensaje de: peliculas.xml es válido Este proceso de validación también puede realizarse on-line con algunas páginas, como

http://xmltools.corefiling.com/schemaValidate/

Analizamos ahora y comentamos lo más importante de este pequeño ejemplo, que luego ampliaremos:

- El esquema contiene un prólogo como cualquier documento xml y un elemento raíz que tiene que llamarse necesariamente schema acompañado del atributo xmlns (que se refiere al espacio de nombres por defecto). Por lo tanto, en todos nuestros esquemas comenzaremos con:

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

Esto indica que los elementos y tipos de datos se refieren al espacio de nombres "http://www.w3.org/2001/XMLSchema". También indica que los elementos y tipos de datos usados deben ir precedidos por **xs:**

- Si aparece **elementFormDefault="qualified"** indica que los elementos usados deben estar declarados por el espacio de nombres.

- Cada elemento de nuestro documento xml será declarado dentro del esquema como

  <xs:element name=.....>

  es decir, será el valor del atributo name. Se comenzará obligatoriamente por el elemento raíz, por lo que en nuestro ejemplo tenemos:

  <xs:element name="pelicula">

- Cuando el elemento este formado por otros elementos, lo indicaremos con la etiqueta **complexType**, indicando luego su forma de agruparse mediante otra etiqueta que más tarde se ampliará el significado.

En nuestro ejemplo, ya que <pelicula> no contiene directamente datos, sino otro elemento anidado en el llamado <titulo>, lo indicamos de la siguiente forma:

<xs:complexType>

<xs:sequence>

.....

Y llegamos así a la descripción del elemento <titulo> que es el que contiene datos, teniendo que especificar el tipo de datos que puede contener, en este caso string.

<xs:element name="titulo" type="xs:string"/>

Esta ha sido la explicación del contenido de un esquema básico, que nos podrá validar documentos XML que cumplan con la estructura descrita en él, pero para poder usarlo necesitamos invocarlo desde los documentos XML, es decir, desde cada instancia a validar:

Para ello, en el elemento raíz del documento XML que contiene nuestra película, que sería una posible instancia, hemos incluido los atributos necesarios para poder validarlo:

<pelicula xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="peliculas.xsd">

……

Este es un caso simplificado, no vamos a utilizar espacios de nombres, pero no obstante se hace una referencia a ello ya que relacionará nuestro documento con su correspondiente esquema incluido en el fichero **peliculas.xsd**.

2. INTRODUCCIÓN A LOS NAMESPACE

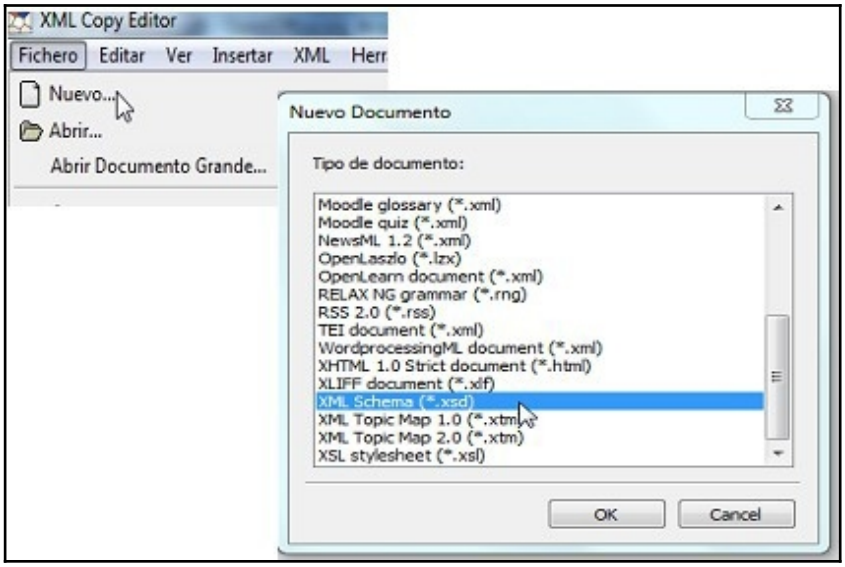
Los espacios de nombres se pueden utilizar tanto en los documentos xml como en los propios esquemas, y su función es la de evitar conflictos entre elementos y/o atributos en los que coincida el nombre, ya que pueden haber sido definidos en sitios diferentes.

Es decir, se basan en un sistema que asocia los nombres de los elementos a un nombre único para así evitar una duplicidad de identificadores.

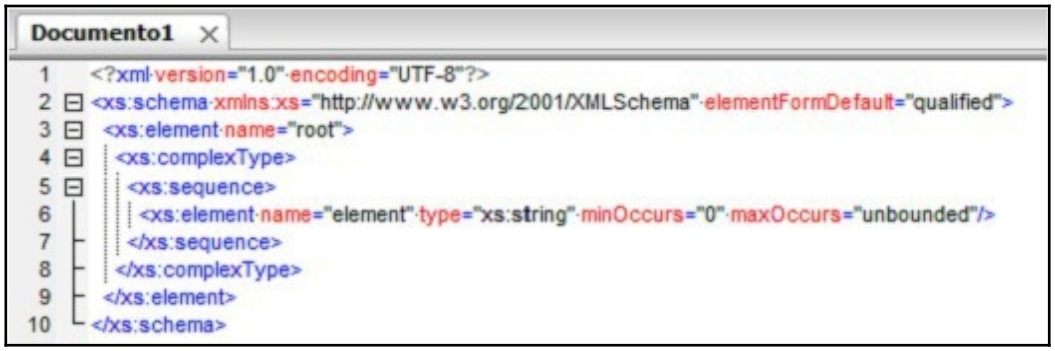
Un espacio de nombres se define a partir del atributo **xmlns:**(que quiere decir espacio de nombres XML) y a continuación el prefijo que se utilizará en el documento XML para relacionar los elementos y atributos con ese espacio de nombres en particular. Ese prefijo puede ser arbitrario, aunque en algunos casos se acostumbra utilizar algunas denominaciones concretas relacionadas con el tipo de documento que se emplea.

En la creación de esquemas se suele utilizar los prefijos **xs** y **xsd**.

Por esa razón cuando iniciamos un documento para contener un esquema en el programa editor que estamos utilizando (XMLCopyEditor), si al iniciar un documento nuevo seleccionamos la plantilla del esquema:



Vemos que nos sale una plantilla en la que todos los nombres de posibles elementos, llevan el prefijo **xs:** (otros editores insertan xsd)




```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="pelicula">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="titulo" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

Es importante saber que para poder aplicar algunas características avanzadas de los esquemas, como por ejemplo la creación de nuevos tipos, es necesario usar estos espacios de nombres.

En la llamada a la validación en la que se suele usar el prefijo xsi:

En nuestro ejemplo anterior hemos utilizado:

<pelicula xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="peliculas.xsd">

Finalmente, es importante entender que estas dos etiquetas son equivalentes:

<xs:element name="pelicula"> y <element name="pelicula">

Las dos definen como debe ser la etiqueta pelicula dentro de un xml, la única diferencia es que la primera indica que las definiciones están dentro de un espacio de nombres declarado previamente y no tienen que ver con las definiciones normales. Es decir, que ante dos palabras iguales YO dicto cual es su definición. Por ejemplo, ante la palabra once, si previamente no defino donde se está utilizando puedo inducir a error, ¿Estoy hablando de un número, de la o.n.c.e o de la palabra en inglés que se traduce como érase una vez?.

Con el espacio de nombres acoto posibles errores, así que vamos a aprender las definiciones de los esquemas usando siempre el espacio de nombres.

Otro ejemplo podría ser el elemento table, ¿ a que me refiero con él?. ¿A un elemento html o a un mueble en inglés?. En el siguiente ejemplo vemos el uso de los espacios de nombres para diferenciar los dos elementos:

```
 <root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="https://www.w3schools.com/furniture">

<h:table>
        <h:tr>
                <h:td>Apples</h:td>
                <h:td>Bananas</h:td>
        </h:tr>
</h:table>

<f:table>
        <f:name>African Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
</f:table>

</root>
```

3. DEFINICIONES DE ESQUEMAS.

3.1 Declaración

Ya hemos visto y comentado en el ejemplo de las películas que la raíz del documento esquema siempre debe ser el elemento schema, que será una etiqueta que contará con su correspondiente cierre, comprendiendo entre ellas todas las etiquetas que definen las normas que seguirán los documentos xml que se validen utilizando este esquema. El atributo de esta etiqueta será xmlns cuyo valor indicará la norma que seguirá el esquema:

<schema xmlns="http://www.w3.org/2001/XMLSchema">

……

</schema>

Pero recordemos que esta es la forma más simple que podemos encontrar, existiendo muchas otras opciones de utilización entre las que se encuentra el uso de espacios de nombres comentado en el apartado anterior.

3.2 Elementos

Los elementos que forman los esquemas se concretan en la declaración de las etiquetas que podrán formar parte de los documentos XML que serán la instancia a validar.

El concepto es muy similar al que usábamos en los DTD, cuando por ejemplo declarábamos:

<!ELEMENT pelicula (titulo)>

Mediante ese sistema estábamos indicando que el elemento <pelicula> contenía otro elemento llamado <titulo> que a su vez debía ser definido posteriormente como contenedor de datos de la siguiente forma:

<!ELEMENT titulo (#PCDATA) >

De forma similar, en nuestros esquemas vamos a tener distintos casos para declarar los elementos según contengan otros elementos o directamente datos.

Por ejemplo:

<titulo>Casablanca</titulo>

Se declarará en nuestro esquema como:

<xs:element name="titulo" type="xs:string"/>

con lo que estamos especificando que el contenido de la etiqueta <titulo> puede ser un dato formado por caracteres. Veremos posteriormente que podrá contener otros atributos que nos ayudarán en el filtraje de los datos que puede contener (como minOccurs….).

3.3 Tipos de elementos

Los elementos que podemos encontrar en un XML pueden ser uno de estos tipos:

1. Elementos Simples
2. Elementos simples con atributos
3. Elementos que contienen elementos
4. Elementos vacíos
5. Elementos con texto y elementos
6. Elementos con texto, atributos y elementos

3.3.1 Elementos Simples

<xs:element name="xxx" type="yyy" [default | fixed]/>

Como indica su nombre es la definición de el elemento mas sencillo que podemos definir en un XML.

- el atributo **name** contiene el identificador del elemento
- el atributo **type** contiene el tipo de datos que permitiremos que contengan estos elementos. Se refiere a que serán caracteres o números, o fechas, etc., (en los DTDs no podíamos hacer esta distinción). Este concepto es muy importante para los esquemas y lo trataremos en un apartado posterior.

Por ejemplo, el elemento que usábamos en nuestro ejemplo:

```
<titulo>Casablanca</titulo>
```

Se declaraba en nuestro esquema como:

```
<xs:element name="titulo" type="xs:string"/>
```

con lo que estamos especificando que el contenido de la etiqueta <titulo> puede ser un dato formado por caracteres. Veremos posteriormente que podrá contener otros atributos que nos ayudarán en el filtraje de los datos que puede contener (como minOccurs....).

Un ejemplo completo sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<nombre xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation ="1.simple.xsd">
  Jana
</nombre>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="nombre" type="xs:string" />
</xs:schema>
```

A un elemento podemos añadirle estas dos características:

https://www.w3schools.com/xml/schema_simple.asp

- default: define un valor por defecto para el elemento si no se ha rellenado. A MI, no me ha funcionado en ningún caso!.
- fixed: define que un elemento debe estar rellenado con ese valor y si no da error. Este si funciona.

3.3.2 Elementos simples con atributos

<xs:element name="xxx"> ← ¡OJO, pierdo el tipo del elemento y se indica en extensión!

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:yyy">

<xs:attribute name="xxx" type="yyy" [default="EN" | fixed="EN" | use="required"]/>

El elemento tiene atributos y un texto. Al contener un atributo el elemento pasa a ser mas complejo.

```
<?xml version="1.0" encoding="UTF-8"?>
<nombre idioma="EN" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation ="2.atributo.xsd">
  Jana
</nombre>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="nombre">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="idioma" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Un atributo admite las siguientes características:

- default: en este caso si funciona.
- fixed: obliga a un valor que no se puede alterar.
- use: indica si el atributo es o no obligatorio. Required o optional.

3.3.3 Elementos que contienen elementos

```
<xs:element name="elemento">
        <xs:complexType>
                [<xs:sequence> <xs:all> <xs:choice>]
```

El elemento contiene a otros elementos, sin texto ni atributos:

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="3.contiene.elementos.xsd">
   <nombre>Jana</nombre>
   <apellidos>Gonzalez</apellidos>
   <edad>34</edad>
</alumno>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="alumno">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="edad" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Como veremos mas adelante podemos usar sequence, all o choice para definir como serán los elementos que contiene el elemento. De momento usaremos xs:sequence indicando que todos los elementos deben ser esos y en ese orden.

3.3.4 Elementos vacíos

```
<xs:element name="elemento">
        <xs:complexType>
                <xs:attribute name="atributo" type="xs:integer"/>
```

El elemento solo tiene un atributo pero no tiene ningún contenido.

```
<?xml version="1.0" encoding="UTF-8"?>
<nombre idioma="EN" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="4.elemento.vacio.atributo.xsd">
</nombre>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="nombre">
    <xs:complexType>
      <xs:attribute name="idioma" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.3.5 Elemento con texto y elementos

<xs:element name="elemento”>
<xs:complexType mixed="true">
<xs:sequence>

El elemento no tiene atributos, pero si que tiene texto y elementos dentro.

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="5.elemento.texto.elementos.xsd">
    Alumna superdotada
    <nombre>Jana</nombre>
    <apellidos>Gonzalez</apellidos>
    <edad>34</edad>
</alumno>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="alumno">
    <xs:complexType  mixed="true">
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellidos" type="xs:string"/>
        <xs:element name="edad" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.3.6 Elemento con texto, atributos y elementos

El elemento tiene todas las posibilidades, atributos, texto y elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno idioma="ES" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="6.texto.atributo.elementos.xsd">
    Alumna superdotada
    <nombre>Jana</nombre>
    <apellidos>Gonzalez</apellidos>
    <edad>34</edad>
</alumno>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="alumno">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellidos" type="xs:string"/>
        <xs:element name="edad" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="idioma" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.4 Atributos

Como hemos definido previamente la definición de los atributos se incluirán siempre después de la declaración correspondiente al grupo de los elementos que lo forman.

Además podemos incluir información sobre la obligatoriedad de incluir el atributo (**use ="optional"**) o un valor por defecto (**default** =”...”).

<xs:attribute name="xxx" type="yyy" [default="zzz" | fixed="zzz" | use="required"]/>

Por ejemplo:

```
<pelicula estreno="1942" minutos="102 ">
        <titulo>Casablanca</titulo>
        <director>Michael Curtiz</director>
</pelicula>
```

Que se validaría con la parte del esquema:

```
<xs:element name="pelicula">
<xs:complexType>
        <xs:sequence>
                        <xs:element name="titulo" type="xs:string"/>
                        <xs:element name="director" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="estreno" type="xs:string" > </xs:attribute>
        <xs:attribute name="minutos" type="xs:integer" use="optional" default ="100" >
        </xs:attribute>
</xs:complexType>
</xs:element>
```

Si no se indica nada, los atributos son todos opcionales. En caso de querer obligar su incorporación tendremos que poner use = "required”

3.5 Tipos de datos

En los ejemplos anteriores, al utilizar el atributo **type** con el que se puede definir la clase de datos a la que pertenecerá el elemento que no es un tipo complejo, se ha asignado por simplificación solamente el valor string, ya que este contiene todos los caracteres tratados como tal, es decir, todas las letras, números y signos básicos, pero podemos controlar mucho más el tipo de datos al que pertenecen los valores de los elementos que incorporemos en nuestros documentos.

- Tipos simples primitivos y/o derivados de primitivos

Los tipos de datos que no son complejos, es decir, los que normalmente asignamos en el atributo

Type cuando declaramos un elemento que contiene datos, puede pertenecer a uno de los casi cincuenta tipos de datos simples que están previamente especificados.

En la siguiente tabla se enumeran y explican los más comunes:

Tipos de dato	Significado	Ejemplos
String	Cualquier cadena de texto que contenga letras y/o números	"Casablanca" "0005RX" "964 23 25 25"
Boolean	Valores lógicos correspondientes a cierto y falso	True,false
Integer byte short long ….	Números enteros, positivos o negativos. Se diferencian solo por el rango de valores que pueden representar.	10 -88 0 125
decimal float double ….	Números con parte decimal	15.3 300.0 -80.7777
Time	Hora	12:30:00
Date	Fecha (formato AAAA-MMDD)	7/01/2013

Así, por ejemplo, podríamos modificar el tipo de los atributos estreno y minutos a integer para que solo pudiesen validar números enteros, como corresponde al dato relativo al año de estreno a los minutos de duración de la pelicula:

<xs:attribute name="estreno" type="xs:**integer**" > </xs:attribute>

<xs:attribute name="minutos" type="xs:**integer** " use="optional" default ="100" > </xs:attribute>

De esta forma, un documento instancia que contuviese la siguiente información **NO** podría validarse:

<pelicula…

estreno="pelicula antigua " minutos="muchos ">

….

</pelicula>

Nuestro esquema ahora solo admitiría números enteros. Sin embargo, con el tipo string con el que lo habíamos declarado inicialmente, sí que sería válido, ya que el tipo string admite todos los literales, sean letras o dígitos, igual que nos ocurría en los DTD con los elementos #PCDATA.