Apuntes básicos del Tema 7

ESQUEMAS

INTRODUCCION A LA VALIDACIÓN DE DOCUMENTOS MEDIANTE ESQUEMAS

Contenidos

7.0- Introducción y definición

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

Siguiendo la definición de la Wikipedia:

XML Schema es un *lenguaje de esquema* utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

Los documentos que contendrán los esquemas serán documentos XSD (XML Schema Definition) y al igual que los DTD forman parte de los XSDL (*XML Schema Definition Language*) que es el nombre técnico de los lenguajes de esquema.

En el tema 3 ya hemos utilizado 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 (recordemos que cuando intentábamos validarlo nos daba error).

Además, 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.

Sin embargo, las especificaciones tecnológicas de los esquemas son de una extensión y complejidad **demasiado amplias para poder abarcarlas en su totalidad en el ámbito de este curso**, por lo que nos limitaremos a hacer una pequeña introducción del tema que nos sirva para tener una visión práctica de esta tecnología.

7.1 Comenzando con un ejemplo

Antes de ver teóricamente los conceptos que forman parte del esquema, vamos a ver un pequeño ejemplo de su mínima expresión para hacernos una idea general de lo que será la validación de documentos con XSD

Si partimos de nuestro conocido ejemplo de las películas, pero en la versión más simplificada posible 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"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="pelicula">
    <complexType>
      <sequence>
        <element name="titulo" type="string"/>
      </sequence>
    </complexType>
  </element>
</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**

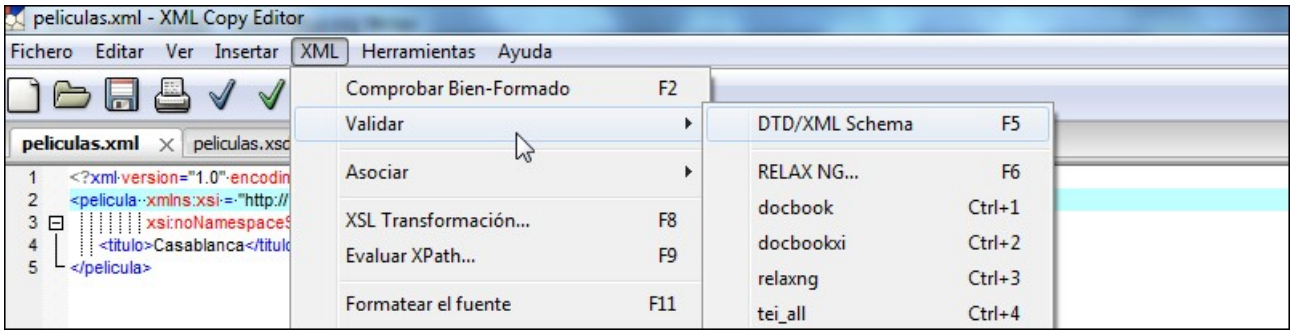
NOTA: esta versión de presentación del esquema, aunque es correcta, no hace uso de los *espacios de nombres* o *namespace* que posteriormente comentaremos.

Para poderlo utilizar y validar la información, tenemos que modificar el documento XML (al que se suele hacer referencia como *documento instancia*, ya que forma una instancia de datos posibles para validar) 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. En este caso también podríamos usar varios programas existentes o páginas de validación on-line, pero seguiremos con el Software de XMLCopyEditor, que ya conocemos.



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

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

<element name=.....>

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

```
<element name="pelicula">
```

- Cuando el elemento esté 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:

```
<complexType>
    <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*.

```
<element name="titulo" type="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 documentos XML que contiene nuestra película, que sería una posible instancia, hemos incluido los atributos necesarios para poderlo validar:

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

Este es un caso simplificado, ya que no vamos a utilizar espacios de nombres (en el apartado posterior se explicara un poco el significado de *namespace*), pero no obstante se hace una referencia a ello ya que relacionara nuestro documento con su correspondiente esquema incluido en el fichero **peliculas.xsd** .

7.2 Introducción a los *namespace*

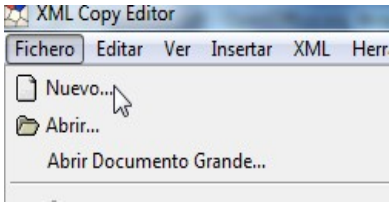
Los espacios de nombres se pueden utilizar tanto en los documentos xml como en los propios esquemas, y su función es la de evitar conflictos entre elementos y/o atributos en los que coincida el nombre, ya que pueden haber sido definidos en sitios diferentes.

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.

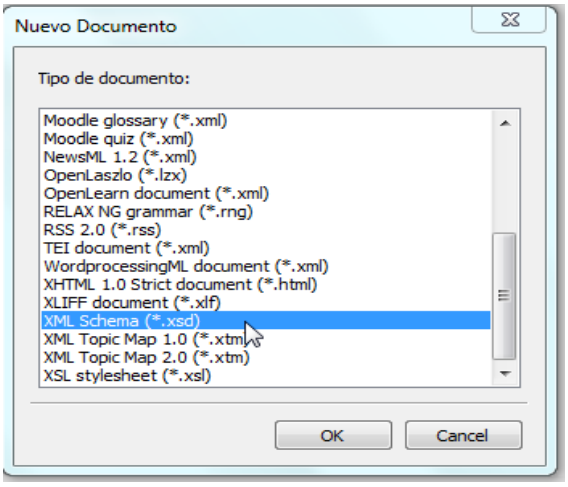
Un espacio de nombres se define a partir del atributo ***xmlns:*** (que quiere decir ***espacio de nombres XML***) y a continuación el prefijo que se utilizará en el documento XML para relacionar los elementos y atributos con ese espacio de nombres en particular. Ese prefijo puede ser arbitrario, aunque 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 utilizandom (XMLCopyEditor), si al iniciar un documento nuevo seleccionamos la plantilla del esquema:

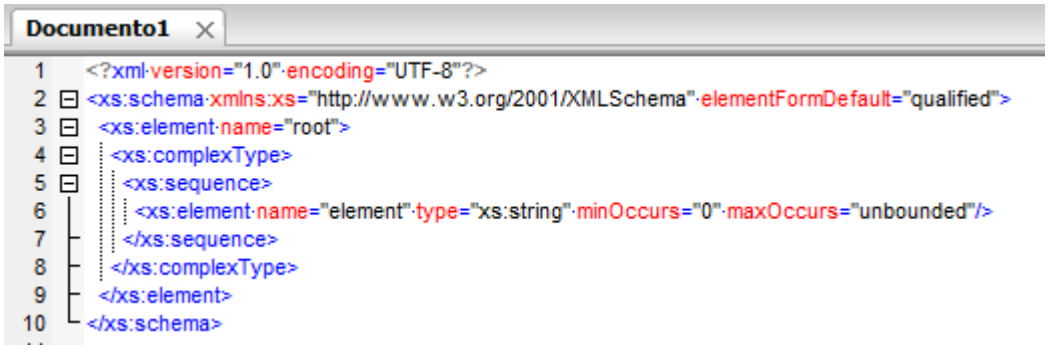


Vemos que nos sale una plantilla en la que todos los nombres de posibles elementos, llevan el prefijo ***xs:*** (otros editores insertan ***xsd***)



Siguiendo esta plantilla, podríamos haber creado un esquema para nuestro ejemplo básico, de forma totalmente equivalente a como lo hemos hecho en el apartado anterior, pero anteponiendo el prefijo ***xs:*** a cada uno de los elementos y a los tipos básicos (en este caso string):

```
<?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>
```



Pero, teniendo claro la gran utilidad y versatilidad que puede tener el uso de los espacios de nombres en aplicaciones complejas en las que intervengan gran diversidad de documentos base , y sabiendo ya a que se refiere este concepto para que no nos confunda cuando lo encontremos en textos alternativos que tratan el tema, en nuestro caso prescindiremos de momento de su uso para simplificar los documentos y ganar en claridad y comprensión de los conceptos básicos.

También 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, por lo que en los anexos finales se retoma estos conceptos para utilizarlos en expresiones más avanzadas.

Así mismo, clarificar el concepto, aunque no lo usemos en el esquema, nos ayuda a entender la sintaxis que de forma estándar se incluye, como por ejemplo en el documento *instancia* del xml , 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">
```

Y ahora ya sabemos algo más de su significado.

7.3 Partes importantes de los esquemas.

7.3.1 -Declaración

Ya hemos visto y comentado en el ejemplo de las peliculas 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,.

.

7.3.2 - Elementos que contienen elementos y elementos que contienen datos.

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.

Cuando un elemento contiene directamente datos tenemos que incluir dos atributos en su definición

- el atributo ***name*** que contiene el identificador del elemento

- el atributo ***type*** que 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:

<element **name="titulo" type="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....).

Cuando un elemento solo contiene otros elementos , lo especificamos mediante un nuevo elemento llamado <***complexType***> que agrupará los distintos elementos que lo forman, y dará información de como se organizan dentro del grupo que lo forman.

Esta organización responde a los tres modos siguientes, que se especifican dentro del esquema como un elemento más:

<sequence>

- Contiene un conjunto de elementos que deberán incluirse en los documentos, respetando estrictamente el orden en que están declarados.
  Ejemplo:

  Con esta porción de esquema:

```
<element name="pelicula">
  <complexType>
   <sequence>
          <element name="titulo" type="string"/>
          <element name="director" type="string"/>
   </sequence>
  </complexType>
</element>
```

Podemos validar este contenido de un documento :

```
<pelicula>
    <titulo>Casablanca</titulo>
    <director>Michael Curtiz</director>
</pelicula>
```

Pero **NO sería válido** si el documento estuviese redactado de la siguiente forma:

```
<pelicula>
    <director>Michael Curtiz</director>
     <titulo>Casablanca</titulo>
</pelicula>
```

Ya que el esquema exige que primero se especifique el título y a continuación el director.

<all>

- Contendrá todo el conjunto de elementos que se describen a continuación, pero sin importar el orden en el que se incluyen
- Ejemplo:
  Si tenemos el esquema:

```
<element name="pelicula">
  <complexType>
   <all>
         <element name="titulo" type="string"/>
        <element name="director" type="string"/>
   </all>
  </complexType>
</element>
```

Los dos contenidos de documentos serán igualmente válidos:

<pelicula> <titulo>Casablanca</titulo> <director>Michael Curtiz</director> </pelicula>	<pelicula> <director>Michael Curtiz</director> <titulo>Casablanca</titulo> </pelicula>

<choice>

- El conjunto de elementos que se describen sirve para que el documento elija solo uno de ellos.

- Ejemplo:
  Si tenemos el esquema:

```
<element name="pelicula">
 <complexType>
  <choice>
   <element name="titulo" type="string"/>
   <element name="director" type="string"/>
  </choice>
 </complexType>
</element>
```

Sería válido cualquiera de estos dos contenidos de documento:

<pelicula> <titulo>Casablanca</titulo> </pelicula>	<pelicula> <director>Michael Curtiz</director> </pelicula>

Pero no validaría un documento que contuviese ambos elementos.

7.3.3- Atributos

El elemento <complexType> utilizado anteriormente para contener la agrupación de distintos elementos se emplea también para incluir la descripción de los atributos que pertenecen a un elemento, añadiendo ahora el elemento **<attribute>** que contendrá la definición del nombre y el tipo de datos que podrá tener cada atributo, de forma similar a como declarábamos los elementos.

Podemos encontrarnos tres casos distintos al usar los atributos en los elementos:

1. Elementos que contienen grupos de elementos y atributos
2. Elementos vacíos que contienen atributos
3. Elementos que contienen directamente texto y atributros

Veremos cada caso por separado:

1. Elementos que contienen grupos de elementos y atributos

   La definición de los atributos se incluirán siempre después de la 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** ="...").

   Si modificamos nuestro documento inicial añadiendo información sobre el año del estreno y su duración podríamos tener el siguiente fragmento:

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

   Que se validaría con la parte del esquema:

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

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

2. Elementos vacíos que solo contienen atributos

   En esta ocasión se trata también de un caso complejo, por lo que tenemos que especificarlo en *complexType* como en el caso de que contenía otros elementos.

   Si queremos validar este elementos de un documento:

```
<clase  genero="drama" />
```

La estructura del esquema es similar a la declarada cuando contenía elementos pero ahora solo se especifica el atributo anidado dentro de *complexType:*

```
<element name="clase">
        <complexType>
                <attribute name="genero" type="string"
use="required"/>
        </complexType>
</element>
```

3. Elementos que contienen texto y atributos

Si quisieramos validar un contenido como el siguiente:

```
<titulo genero="drama">Casablanca</titulo>
```

en el que queremos incluir un atributo en el elemento que contiene el título de la película, tendremos que usar dos nuevas etiquetas : ***simpleContent*** y ***extensión*** con su atributo ***base*** para especificar el tipo de datos del elemento, antes de describir los atributos que contiene y se relacionan anidados:

```
<element name="titulo">
        <complexType>
                <simpleContent>
                        <extension base="string">
                                <attribute name="genero" type="string"
use="required"/>
                        </extension>
                </simpleContent>
        </complexType>
</element>
```

7.3.4 Repeticiones de elementos

Ademas de los atributos *name* y *type* comentados en el apartado 4.2, existen otros dos atributos que pueden incorporarse para acotar el número de veces que un elemento puede repetirse.

- ***minOccurs contiene el valor mínimo del número de veces que el elemento puede aparecer en el documento.***

- Si no se pone nada, su valor será 1, por lo que el elemento deberá figurar obligatoriamente una vez.

-Si contiene el valor 0 indicará que el valor es opcional.

- ***maxOccurs contiene el número de veces máximo que un elemento puede aparecer en el documento.***

-Su valor por defecto también es 1.

-El valor ***unbounded*** significa que no tiene límite, por lo que se podrá incluir el elemento un número indefinido de veces.

Ampliaremos nuestro ejemplo completo del esquema de la película incluyendo un elemento <reparto> que a su vez pueda incorporar cualquier numero de elementos <interprete > aunque también podría no tener ninguno:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="pelicula">
    <complexType>
      <sequence>
        <element name="titulo" type="string"/>
        <element name="director" type="string"/>
        <element name="reparto">
           <complexType>
                <sequence>
                   <element name="interprete" minOccurs="0"
                                              maxOccurs="unbounded"
type="string"/>
                </sequence>
           </complexType>
        </element>
      </sequence>
      <attribute name="estreno" type="string"  > </attribute>
      <attribute name="minutos" type="integer" use="optional" default ="100" >
</attribute>
    </complexType>
  </element>
</schema>
```

Por lo que el siguiente documento sería validado con dicho esquema, independientemente del número de interpretes que incorporemos.

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-
             instance"
            xsi:noNamespaceSchemaLocation
             ="peliculas_repe1.xsd"
            estreno="1942"  minutos="102 ">
    <titulo>Casablanca </titulo>
    <director>Michael Curtiz</director>
    <reparto>
       <interprete>Humphrey Bogart</interprete>
       <interprete> Ingrid Bergman</interprete>
       <interprete>Paul Henreid</interprete>
       <interprete>Claude Rains</interprete>
       <interprete>Conrad Veidt</interprete>
       <interprete>Curt Bois</interprete>
    </reparto>
</pelicula>
```

7.3.5 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.

Existe una distinción sobre los que son *primitivos* y *derivados de primitivos* (como por ejemplo el tipo *integer*, que es una derivación del *decimal*...) pero esa distinción no tiene ninguna relevancia en este contexto, por lo que en la siguiente tabla se enumeran y explican los mas comunes:

Tipos de dato	Significado	Ejemplos
string	Cualquier cadena de texto que contenga letras y/o numeros	“Casablanca” “0005RX” “964 23 25 25”
boolean	Valores lógicos correspondientes a cierto y falso	false, true
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-MM-DD)	13/02/11

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

```
<attribute name="estreno" type="integer"  > </attribute>
<attribute name="minutos" type="integer " use="optional" default ="100" >
</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.

ANEXO I . DE AMPLIACIÓN PARA DEFINIR NUEVOS TIPOS DE DATOS:

NOTA IMPORTANTE: es necesario usar el método de creación de esquemas con la inclusión de ESPACIOS DE NOMBRES para evitar problemas con la mayoría de validadadores cuando se hace uso de ***simpleType***

Notar que simplemente hemos añadido el prefijo ***sx:*** a cada uno de los elementos y tipos primitivos. Para facilitar la lectura se resalta en negrita dicho prefijo.

Definición de tipos enumerados y sus restricciones

Podemos crear tipos de datos nuevos basados en los datos simples y luego aplicar las restricciones para asegurarnos de que la instancia del documento XML contiene solo el conjunto de datos que nosotros hemos creado.

Para ello define el nuevo tipo mediante el elemento ***simpleType*** que contiene el atributo ***name*** con el que ponemos un identificador a este nuevo tipo, y que anida otro elemento ***restrictión*** que a su vez contiene los posibles valores mediante los elementos ***enumeration***

Notar que esta declaración de un nuevo tipo debe ir **al principio** del esquema, justo después del elemento schema y antes de la declaración para el elemento raíz.
A continuación lo vemos aplicado a nuestro ejemplo de las películas, en las que vamos a incluir un elemento *genero* que nos clasifique según un criterio determinado, de forma que solo puedan ser *drama, comedia, fantasía*, etc.,

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

  <xs:simpleType name="Tipo_Genero">
    <xs:restriction base="xs:string">
      <xs:enumeration value="drama"/>
      <xs:enumeration value="comedia" />
      <xs:enumeration value="fantasia" />
      <xs:enumeration
value="documental" />
      <xs:enumeration value="historia" />
      <xs:enumeration value="guerra" />
      <xs:enumeration value="western" />
    </xs:restriction>
  </xs:simpleType>
```

```
 <xs:element name="pelicula">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="director" type="xs:string"/>
    <xs:element name="genero" type="Tipo_Genero"/>
    <xs:element name="reparto">
      <xs:complexType>
          <xs:sequence>
            <xs:element name="interprete"
                 minOccurs="0" maxOccurs="unbounded" type="xs:string"/>
          </xs:sequence>
      </xs:complexType>
    </xs:element>
   </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>
</xs:schema>
```

Este esquema validaría ya nuestra instancia de documento xml en el que la película Casablanca incluyese el género como “drama”.

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation ="peliculas_rest1.xsd"

           estreno="1942"  minutos="102 ">
    <titulo>Casablanca </titulo>
    <director>Michael Curtiz</director>
    <genero>drama</genero>
    <reparto>
      <interprete>Humphrey Bogart</interprete>
      <interprete> Ingrid Bergman</interprete>
      <interprete>Paul Henreid</interprete>
      <interprete>Claude Rains</interprete>
      <interprete>Conrad Veidt</interprete>
      <interprete>Curt Bois</interprete>
    </reparto>
</pelicula>
```

Otras restricciones a los tipos de datos

Las restricciones también se pueden utilizar para hacer un control más detallado del valor de un elemento o atributo.

Por ejemplo, mediante los elementos minInclusive y maxInclusive podemos hacer un rango de valores y podemos definir un tipo de dato basado en uno numérico, pero que sólo permita valores enteros entre 0 y 10, por ejemplo, para validar un elemento que se refiera a una nota:

```
<xs:simpleType name="TipoNota">
 <xs:restriction base="xs:decimal">
     <xs:minInclusive value="0" fixed="true" />
     <xs:maxInclusive value="10" fixed="true" />
 </xs:restriction>
</xs:simpleType>
```

También podemos controlar la cadena de caracteres que forma un string.
Entro otros, podemos usar los elementos:

Elemento	Significado	Ejemplos
pattern	Patrón para los caracteres	patern value= “([0-9\s])*” Validaria todos los dígitos del 0 al 9 y espacios en blanco, sin límite de número de caracteres ---------- pattern value=”{3}-[A-Z]{2} “ Validaria un dato con tres dígitos, un guión y dos letras mayúsculas como: 732-AB ----------
length	Número de caracteres permitidos totales	length value =”9” Nos permite tener un valor de exactamente nueve caracteres
minLength	Número de caracteres mínimos permitidos	minLength = “5” Valor con más de 5 caracteres
maxLength	Número de caracteres máximo permitidos	maxLength=”10” Valor con menos de 10 caracteres

Anexo II - Elementos que contienen otros elementos y texto

NOTA IMPORTANTE: es necesario usar el método de creación de esquemas con la inclusión de ESPACIOS DE NOMBRES para evitar problemas con la mayoría de validadadores cuando se emplea el atributo **mixed**.
Notar que simplemente hemos añadido el prefijo sx: a cada uno de los elementos y tipos primitivos. Para facilitar la lectura se resalta en negrita dicho prefijo.

Aunque los elementos mixtos siempre se suelen evitar por su falta de transparencia en el estructura, supongamos que queremos incluir la siguiente información en un elemento:

```
<productora>
        La Warner  Bros. Pictures eligió como productor a
         <productor> Hal B. Wallis </productor>
</productora>
```

para poder incluirlo en el esquema necesitamos un atributo **mixed**

```
<xs:element name="productora" >
        <xs:complexType mixed="true">
            <xs:sequence>
                <xs:element  name="productor" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
</xs:element>
```