XML XSD

1. Introducción

Una cita para empezar:

> *"the syntax of XML Schema was obviously produced by someone who grew up at the bottom of a deep well in the middle of a dark, wasteful moor where he was tortured daily by abusive giant squirrels and wishes to share his pain with the world"*.
>
> **Robin Berjon (co-author SVG 1.1 spec)**

Ideas básicas:

- XSD: *XML Schema Definition.*
- Alternativa a DTD basada en XML.
- Ventajas respecto a DTDs:
  - Están escritos en XML
    - No hay que aprender una nueva sintaxis, Lenguaje DTD no intuitivo
    - Puedes usar el editor XML y el parser XML
  - Soportan tipos de datos
    - Más fácil validar distintos tipos de datos: fechas, números
    - Se pueden definir “restricciones”
    - límites dentro de los anteriores o patrones de datos,
    - o extensiones sobre los datos, tipos derivados de otros.
  - Mejor soporte espacios de nombres.

2. Primer esquema

Dado el fichero `nota.xml` se crea un esquema inicial:

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

Puede observar:

- Se define un elemento `nota` de tipo complejo
- que será una una secuencia de elementos `xs:sequence`.
- Se definen el nombre y tipo de cada elemento.
- En este caso todos los elementos son del tipo predefinido `xs:string`.
- No se especifica el número de ocurrencias de los elementos (por defecto 1).

Es preferible usar una definiciones de tipo con asignación de nombre, y usar posteriormente ese nombre en al definición del elemento. Observe el mismo esquema donde ahora:

- primero se define el nombre del elemento y se le asigna un nombre de tipo (`type="Tnota"`).
- luego se define el tipo complejo y se le da el nombre usado en el elemento (`name="Tnota"`)

```
    <xs:element name="nota" type="Tnota" />

    <xs:complexType name="Tnota">
        <xs:sequence>
            <xs:element name="de" type="xs:string" />
            <xs:element name="para" type="xs:string" />
            <xs:element name="tema" type="xs:string" />
            <xs:element name="texto" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
```

Ya por último asocie el fichero XML con los datos al esquema creado usando la instrucción de proceso `xml-model`:

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

```
<?xml-model href="nota-02.xsd" type="application/xml"
  schematypens="http://www.w3.org/2001/XMLSchema"?>

<nota>
  <de>Ana</de>
```

> En VScode la instrucción `xml-model` se genera tecleando un < y seleccionando la opción “Insert XML Schema Association”.

Una vez asociado el esquema puede comprobar si el fichero es válido:

- En el editor VSCode , con el plugin Xml de RedHat.
- En línea de comandos con `xmlstarlet`.

```
$ xmlstarlet val -e -s nota-01.xsd nota.xml
nota.xml - valid
$ xmlstarlet val -e -s nota-01.xsd nota_errores.xml
nota_errores.xml:7.8: Element 'para': This element is not expected. Expected
is ( de ).
nota_errores.xml - invalid
$
```

3. Elementos simples

Elementos simples son aquellos que no tienen atributos ni tienen hijos:

```
<xs:element name="xxx" type="yyy"/>
```

Los tipos de datos predefinidos más usados son:

- `xs:date` (en formato `yy-mm-dd`)
- `xs:string`
- `xs:decimal` o `xs:integer`
- `xs:boolean`

Se le añade al ejemplo anterior una fecha y una prioridad con tipos `xs:date` y `xs:integer`:

```
  <xs:complexType name="Tnota">
    <xs:sequence>
      <xs:element name="fecha" type="xs:date" />
      <xs:element name="prioridad" type="xs:integer" />
      <xs:element name="de" type="xs:string" />
```

Puede ver y comprobar este código en los ficheros `nota-03.xml` y `nota-03.xsd`.

4. Restricciones

Se pueden aplicar restricciones sobre los tipos un elemento o atributo. Estas restricciones limitan los valores posibles que puede tomar.

Para crear un restricción:

1. Se define un tipo simple (`xs:simpleType`) y se le da nombre.
2. Dentro se define una restricción (`xs:restriction`) con un atributo `base` que indica el tipo a restringir.
3. Dentro de la restricción se aplica la sintaxis que corresponda a los distintos tipos de restricciones que existen.

Vea un ejemplo de restricción de enteros siguiendo los tres pasos:

```
<xs:simpleType name="Tprioridad">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="5"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="prioridad" type="Tprioridad" />
```

O restricciones sobre cadenas de caracteres, en este caso un enumerado:

```
<xs:element name="car" type="carType"/>
<xs:simpleType name="carType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
</xs:simpleType>
```

O limitando la longitud de la cadena:

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

Incluso se pueden usar patrones:

```
<xs:element name="NIF" type="TNIF" />
<xs:simpleType name="TNIF">
```

```
  <xs:restriction base="xs:string">
     <xs:pattern value="[0-9]{8}[A-Z]"/>
   </xs:restriction>
 </xs:simpleType>
```

5. Atributos

Un atributo se define de forma similar a un tipo simple y los tipos que se utilizan son los mismos que para elementos simples. Vea un ejemplo

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

Pero **esta definición no indica a que elemento pertenece el atributo**, eso dependerá de dónde se incluye la definición del atributo. Y hay tres situaciones diferentes, atributo de un

1. elemento contenedor
2. elemento simple
3. elemento simple con tipo restringido

5.1. Caso 1: atributo en elemento contenedor

Este caso es el más sencillo, se coloca la definición del atributo dentro de la definición del elemento, **al final de la misma**.

Si por ejemplo se añade los atributos `idioma` y `creado` a la nota:

```
  <xs:complexType name="Tnota">
    <xs:sequence>
      ...
      <xs:element name="texto" type="xs:string" />
    </xs:sequence>

    <!-- AQUÍ al final se añaden los atributos -->
    <xs:attribute name="idioma" type="xs:string"/>
    <xs:attribute name="creado" type="xs:date"/>

  </xs:complexType>
```

Puede ver y comprobar este código en los ficheros `nota-05.xml` y `nota-05.xsd`.

5.2. Caso 2: atributo en elemento simple

Al aplicar un atributo a un elemento de tipo simple la cosa se complica. Los tipos simples por definición no admiten atributos (un elemento con atributos se considera complejo). Por tanto para definir un elemento simple con atributo hay que "convertir" el elemento a tipo complejo.

Además esto implica sintaxis adicional:

- Definir un tipo complejo cuyo contenido es simple con `xs:simpleContent`
- y dentro de ese contenido simple utilizar una `xs:extension` (o una restricción).

En el siguiente ejemplo se añade el atributo `tag` al elemento simple `prioridad`.

1. Definimos un nuevo tipo con `xs:complexType` de nombre `Tprioridad`.
2. Dentro del ese tipo complejo el contenido es simple y se indica con `xs:simpleContent`.
3. Dentro del contenido simple definimos una extensión `xs:extension` especificando en el atributo `base` el tipo inicial del elemento simple.
4. Y finalmente dentro de la extensión se incluye la definición del atributo.

El código queda:

```
<xs:complexType name="Tprioridad">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="tag" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Se ha definido el tipo. No olvide ahora modificar el elemento `prioridad` para indicar que ya no es del tipo `xs:integer` sino `Tprioridad`, sin prefijo `xs:`

```
<xs:complexType name="Tnota">
  <xs:sequence>
    ...
    <!-- <xs:element name="prioridad" type="xs:integer"  /> -->
    <xs:element name="prioridad" type="Tprioridad"   />
    ...
```

Puede ver y comprobar este código en los ficheros `nota-06.xml` y `nota-06.xsd`.

5.3. Caso 3: atributo en elemento simple con restricciones

Es similar al caso 2, pero el tipo del elemento tiene una restricción aplicada. Y eso vuelve a complicar la situación porque de acuerdo con la norma

> *"Each complex type definition is either*
>
> - *a restriction of a complex base type definition or*
> - *an extension of a simple or complex …*

Si el elemento ya tiene un tipo con una restricción no lo podemos extender. Esto obliga a definir el tipo final en dos pasos:

1. Se define un tipo simple con la restricción. Este tipo hará las funciones de tipo "intermedio".
2. Se define un tipo complejo que sea una extensión del tipo intermedio anterior (se indica en elatributo `base`) y se le añade el atributo.

3. Y se define el elemento usando el tipo del paso 2.

Vea un ejemplo con el elemento `prioridad` de tipo entero con el mínimo y el máximo restringidos. A ese elemento se le añade el atributo `tag`:

```
<!-- Paso 1 -->
  <xs:simpleType name="Tintermedio">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5"/>
 </xs:restriction>
  </xs:simpleType>

<!-- Paso 2 -->
  <xs:complexType name="Tprioridad">
    <xs:simpleContent>
      <xs:extension base="Tintermedio">
        <xs:attribute name="tag" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<!-- Definición del elemento -->
  <xs:element name="prioridad" type="Tprioridad" minOccurs="0"  />
```

Puede ver y comprobar este código en los ficheros `nota-07.xml` y `nota-07.xsd`.

5.4. Atributo: obligatorio u opcional

Los atributos son opcionales por defecto. Si desea hacer que un atributo sea obligatorio debe incluir el atributo `use` con el valor `required`:

```
  <xs:attribute name="valor" type="xs:decimal" use="required" />
```

Otras opciones posibles son que el atributo tenga un valor

- por defecto: `<xs:attribute name="lang" type="xs:string" default="EN"/>`
- fijo: `<xs:attribute name="lang" type="xs:string" fixed="EN"/>`

6. Elementos complejos

Son aquellos que no son simples, es decir, que contienen otros elementos y/o atributos.

Podemos clasificarlos en

1. Elementos contenedores que sólo contienen otros elementos.
2. Elementos vacíos
3. Elementos contenedores que contiene otros elementos y texto (elementos mixtos).
4. Elementos que sólo contienen texto y atributos

6.1. Elementos contenedores

Como elementos contenedores se presentan:

- `xs:sequence`: deben aparecer en el orden indicado.
- `xs:all`: los elementos pueden aparecer en cualquier orden, y cada uno aparece una única vez.
- `xs:choice`: sólo aparece un elemento de los que se enumeran.

Ya se ha mostrado en ejemplos previos el uso de una secuencia `xs:sequence`. Vea otro ejemplo del uso de `xs:sequence`:

```
<!-- Definición del tipo -->
<xs:complexType name="Tpersoninfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<!-- Uso del tipo en los elementos-->
<xs:element name="employee" type="Tpersoninfo"/>
<xs:element name="student" type="Tpersoninfo"/>
```

Ahora un ejemplo de `xs:all`

```
<xs:complexType name="Tpersona">
  <xs:all>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apell" type="xs:string"/>
   <xs:element name="dni" type="Tdni" minOccurs="0"/>
  </xs:all>
</xs:complexType>
```

Y un ejemplo de `xs:choice` (combinado con `xs:sequence`)

```
<xs:complexType name="anuncio">
  <xs:sequence>
     <xs:element type="xs:date" name="fecha"/>
     <xs:choice>
        <xs:element type="xs:string" name="asunto"/>
        <xs:element type="xs:string" name="texto"/>
     </xs:choice>
     <xs:element type="xs:float" name="precio"/>
    ...
    </xs:sequence>
</xs:complexType>
```

6.2. Elemento vacío

La definición de un elemento vacío es sencilla. Se define un tipo complejo con el interior vacío:

```
<xs:complexType name="Tbr">  <!-- elemento vacío -->

</xs:complexType>

<xs:element name="br" type="Tbr"/>
```

Si el elemento vacío tiene atributos se indican en su interior:

```
<xs:complexType name="Tpedido"> <!-- elemento vacío con un atributo -->
  <xs:attribute name="codigo" type="xs:integer"/>
</xs:complexType>

<xs:element name="pedido" type="Tpedido"/>
```

6.3. Elementos mixtos

En general se deben evitar los elementos mixtos, aquellos que contienen texto y al mismo tiempo nodos. Para definirlos el único cambio respecto a un elemento contenedor es que se añade el atributo `mixed="true"` en la definición del tipo:

```
<xs:element name="letter" type="lettertype"/>
  <!-- mixed hace que tengamos elemento mixto -->
    <xs:complexType name="lettertype" mixed="true">
     <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="oid" type="xs:integer"/>
     </xs:sequence>
   </xs:complexType>
```

Y un ejemplo de datos válidos según ese esquema sería:

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <oid>1032</oid> will be shipped on 2001-07-13.
</letter>
```

6.4. Elemento simple con atributos

Se ha presentado esta opción al estudiar como añadir un atributo a un elemento simple, en el apartado 5.2. Caso 2: atributo en elemento simple.

7. Elementos y ocurrencias

Se puede modificar el número de ocurrencias de un elemento usando los atributos `minOccurs` y `maxOccurs`. El valor por defecto de estos atributos es 1. También se puede usar el valor predefinido `unbounded` (ilimitado).

Modifiquemos el ejemplo de la nota para permitir que los destinatarios del mensaje sean ilimitados (`maxOccurs="unbounded"`) y que la prioridad sea opcional (`minOccurs="0"`):

```
  <xs:complexType name="Tnota">
    <xs:sequence>
      <xs:element name="fecha" type="xs:date" />
      <xs:element name="prioridad" minOccurs="0" type="xs:integer" />
      <xs:element name="de" type="xs:string" />
      <xs:element name="para" maxOccurs="unbounded" type="xs:string" />
     ...
```

Puede ver y comprobar este código en los ficheros `nota-04.xml` y `nota-04.xsd`.

8. Anexos

8.1. Más sobre tipos

Existen más tipos predefinidos. Algunos de ellos:

- `xs:boolean`: Valores false/true
- `xs:anyURI`: Para URL
- `xs:time`: "hh:mm:ss“ (todos necesarios)
- `xs:duration` (intervalo de tiempo)
  - Formato: "PnYnMnDTnHnMnS“
  - Ejemplos:

    ```
    <period>P5Y2M10D</period>
    <period>P5Y2M10DT15H</period>
    <period>PT15H</period>
    ```

    También existen tipos numéricos:
- `xs:negativeInteger`
- `xs:positiveInteger`
- ...

O tipos derivados de string

- `xs:language`
- `xs:NCName`
- `xs:ID`
- ...

8.2. Extensión de tipo

Permite definir un tipo extendiendo otro al que le añade nuevos elementos o atributos. Utiliza

- `xs:complexContent` (o `xs:simpleContent`)
- `xs:extension`

Vea un ejemplo donde se extiende el tipo `personInfo` y se crea el tipo `fullpersoninfo` que añade elementos dirección y ciudad. Produce soluciones más simples, legibles y fáciles de modificar y mantener.

```
<!-- define personinfo" -->
<xs:complexType name="personinfo">
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    <xs:element name="username" type="xs:string"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
  <!-- utiliza y extiende el tipo personinfo" -->
     <xs:extension base="personinfo">
       <xs:sequence>
          <xs:element name="address" type="xs:string"/>
          <xs:element name="city" type="xs:string"/>
       </xs:sequence>
     </xs:extension>
  </xs:complexContent>
</xs:complexType>


<xs:element name="employee" type="fullpersoninfo"/>
<xs:element name="user" type="personinfo"/>
```

8.3. TODO: validar el fichero XSD

Se puede y debe comprobar antes de usar un esquema para validar datos que

1. El esquema XSD está bien formado, no olvide que es un fichero XML. Lo puede comprobar en el editor o en la línea de comandos:

   ```
   $ xmlstarlet val -e esquema.xsd
   nota-07.xsd - valid
   $
   ```

2. Y es válido como *schema*, sigue la sintaxis XSD. Para ello puede usar el editor. Si quiere validar el esquema XSD en línea de comandso necesita un fichero adicional dónde se detalle la sintaxis de un esquema. Se le proporciona el fichero `XMLSchema.xsd` para ello:

   ```
   $ xmlstarlet val -e -s XMLSchema.xsd nota-07.xsd
   nota-07.xsd - valid
   ```

```
$ xmlstarlet val -e -s XMLSchema.xsd nota-07-errores.xsd
nota-07-errores.xsd:16.15: Element
'{http://www.w3.org/2001/XMLSchema}test': This element is not expected.
nota-07-errores.xsd - invalid
$
```