2.2-Esquemas XML

Sitio:	Aula Virtual do IES de Teis	Impreso por:	Deivid Durán Durán
Curso:	Linguaxes de Marcas e Sistemas de Xestión de Información 2021-22 (DAM-A)	Data:	Venres, 24 de Xuño de 2022, 00:01
Libro:	2.2-Esquemas XML		

Descrición

Táboa de contidos

1. Esquemas XML versus DTDs

2. Ejemplo de esquema XML

3. Vinculación de esquemas XML con documentos XML que no tienen su propio espacio de nombres
3.1. Vinculación con instrucción de procesamiento xml-model
3.2. Vinculación de esquemas XML con documentos XML que tienen su propio espacio de nombres

4. Construcción de un esquema XML
4.1. Declaración del elementos y atributos.
4.2. Multiplicidades de elementos
4.3. Tipos simples predefinidos
4.4. Tipos simples con restricciones
4.5. Modo de declarar los elementos
4.6. Tipos complejos
4.7. Tipos complejos: compositores de orden
4.8. Grupos
4.9. Extensiones
4.10. Documentación: Annotation

Esquemas XML vs DTDs

- Ambas opciones permiten establecer los elementos, atributos, número de ocurrencias y el tipo de información que pueden formar parte de un documento XML para su posterior **validación**. Sin embargo, un esquema XML es, a su vez, **un documento XML** y, por tanto, tiene un único elemento raíz llamado ***schema*.**
- Los esquemas XML se encuentran en un fichero externo con la **extensión .xsd.**
- Los esquemas XML son mucho más potentes, pues permiten:
  - **utilizar tipos de datos** para **elementos**, además de para **atributos**: cadenas de texto, enteros, decimales, fechas, etc. **e incluso definir tipos propios.**
  - establecer con **mayor exactitud** el **número de ocurrencias** de un elemento
  - **utilizar espacios de nombres** en un documento XML
  - utilizar **el mismo nombre para dos elementos en distinto nivel** de la jerarquía del documento

Ejemplo de esquema XML

Veamos un ejemplo para el documento XML **top-billboard.xml**:

```xml
<top_billboard_espanhol>
    <cancion top="7">
        <titulo>La Macarena</titulo>
        <interprete>Los del Río</interprete>
        <anho>1995</anho>

    </cancion>
    <cancion top="9">
        <titulo>Eres tú</titulo>
        <interprete>Mocedades</interprete>
        <anho>1973</anho>
    </cancion>
    <cancion top="33">
        <titulo>Corazón Partío</titulo>
        <interprete>Alejandro Sanz</interprete>
        <anho>1995</anho>
    </cancion>
    <cancion top="32">
        <titulo>Mediterráneo </titulo>
        <interprete>Joan Manuel Serrat</interprete>
        <anho>1971</anho>
    </cancion>
    <cancion top="21">
        <titulo>Y cómo es él </titulo>
        <interprete>José Luis Perales
        </interprete>
        <anho>1982</anho>
    </cancion>
    <cancion top="7">
        <titulo>Bailando</titulo>
        <interprete>Enrique Iglesias</interprete>
        <interprete>Gente de Zona</interprete>
        <anho>2014</anho>
    </cancion>
</top_billboard_espanhol>
```

Un ejemplo de esquema XML podría ser el documento **top-billboard.xsd**:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="titulo" type="xs:string" />
                            <xs:element name="interprete" minOccurs="0" maxOccurs="unbounded" type="xs:strin
                            <xs:element name="anho" type="xs:gYear" default="2000" />
                        </xs:sequence>
                        <xs:attribute name="top" type="xs:positiveInteger" use="required" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- Tiene un único elemento raíz: **schema**
- Usa el espacio de nombres: "**http://www.w3.org/2001/XMLSchema**". Se indica de este modo que los elementos y atributos que vamos a utilizar pertenecen a ese espacio de nombres.

    - El prefijo **xs** del espacio de nombres no tiene por qué ser siempre **xs**. También es posible ver prefijos **xsd** o podríamos escoger cualquier otro con tal de que luego seamos consistentes y lo utilicemos en el resto del documento. (Está presente en todo el documento)
- El **elemento raíz**, *top_billboard-espanhol*, se declara en un elemento llamado ***element*** con el atributo ***name***

Veremos en sucesivas secciones la sintaxis concreta, pero si leemos el resto del documento, intuitivamente, **vemos una estructura asimilable** con la estructura del documento XML que conocemos.

Vinculación de esquemas XML con documentos XML que no tienen su propio espacio de nombres

Podemos vincular el documento XML que deberá validar contra un esquema XML usando 2 opciones con Visual Studio Code y la extensión de XML que hemos utilizado hasta ahora :

OPCIÓN 1 (la que utilizaremos habitualmente):

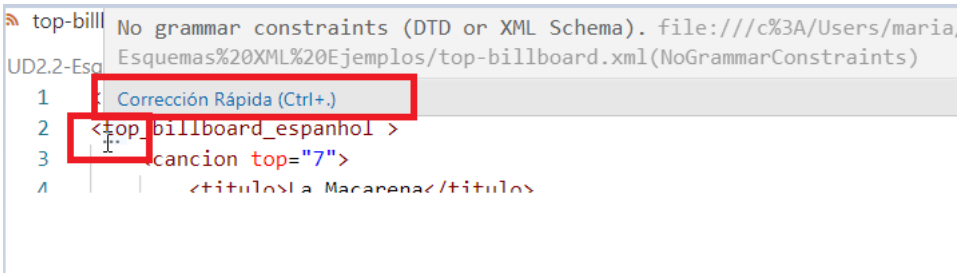
Añadimos **al elemento raíz** el siguiente espacio de nombres: **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

Si el documento XML que deberá validar contra un esquema XML **no tiene su propio espacio de nombres**, añadiremos también al elemento raíz:

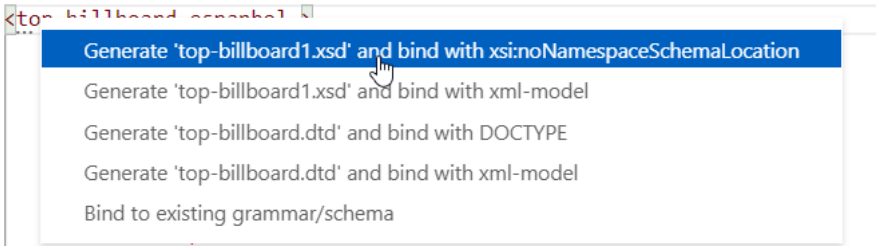
- **xsi:noNamespaceSchemaLocation="fichero.xsd"**, sustituyendo **fichero.xsd** por la ruta del fichero donde se encuentre el esquema XML.

Para realizar este proceso, podemos ayudarnos de VS Code:

Bajo el elemento raíz del documento XML, vemos 3 puntos. Si posamos el ratón durante unos segundos, aparecerá un mensaje: *No grammar constraints*



No grammar constraints > *Corrección rápida* > *Generate 'top-billboard.xsd" and bind with xsi:noNamespaceSchemaLocation:*



Se crearán automáticamente el fichero .xsd y se modificará el documento XML.

Así quedaría el documento top-billboard.xml que veíamos antes:

```xml
<?xml version="1.0" encoding="UTF-8" standalone='no'?>
<top_billboard_espanhol xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="top-billboard.xsd">
    <cancion top="7">
        <titulo>La Macarena</titulo>
        <interprete>Los del Río</interprete>
        <anho>1995</anho>

    </cancion>
    <cancion top="9">
        <titulo>Eres tú</titulo>
        <interprete>Mocedades</interprete>
        <anho>1973</anho>
    </cancion>
    <cancion top="33">
        <titulo>Corazón Partío</titulo>
        <interprete>Alejandro Sanz</interprete>
        <anho>1995</anho>
    </cancion>
    <cancion top="32">
        <titulo>Mediterráneo </titulo>
        <interprete>Joan Manuel Serrat</interprete>
        <anho>1971</anho>
    </cancion>
    <cancion top="21">
        <titulo>Y cómo es él </titulo>
        <interprete>José Luis Perales
        </interprete>
        <anho>1982</anho>
    </cancion>
    <cancion top="7">
        <titulo>Bailando</titulo>
        <interprete>Enrique Iglesias</interprete>
        <interprete>Gente de Zona</interprete>
        <anho>2014</anho>
    </cancion>
</top_billboard_espanhol>
```

El fichero generado automáticamente **top-billboard.xsd** será el siguiente:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="titulo" type="xs:string" />
                            <xs:element name="interprete" maxOccurs="unbounded" type="xs:string" />
                            <xs:element name="anho" type="xs:string" />
                        </xs:sequence>
                        <xs:attribute name="top" type="xs:integer" use="required" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Como acabamos de ver, el proceso de creación de un esquema XML se puede facilitar enormemente gracias a las herramientas de desarrollo integrado como Visual Studio Code con la extensión de XML o editores especializados. Bien es cierto que **aún serán necesarios algunos ajustes y refinamientos sobre los tipos de datos y restricciones sobre los mismos.**

Vinculación de esquemas XML con documentos XML que no tienen su propio espacio de nombres

OPCIÓN 2 (instrucción de procesamiento xml-model)

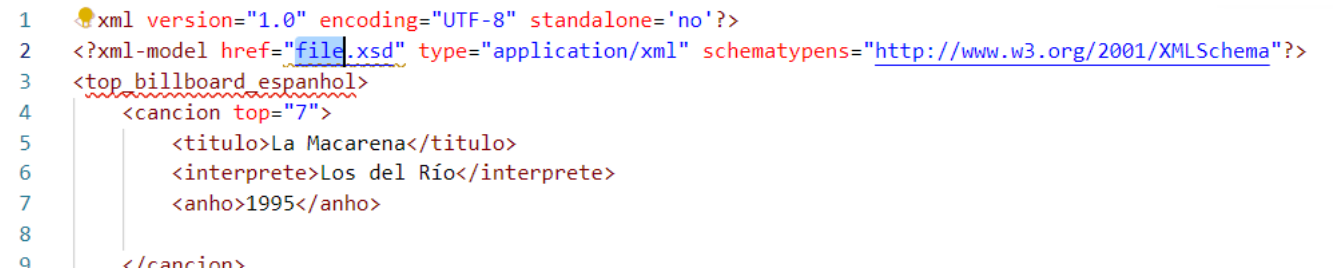
Esta opción solo está disponible en VS Code con el plugin de XML de Red Hat y no funcionará en Notepad++ con XML Tools, por lo que no la utilizaremos en clase. Sin embargo, podemos ver cómo añadirla con VS Code:

Si antes del elemento raíz (y después del prólogo si lo huebiera), escribimos un <, seleccionamos la sugerencia:
Insert XML Schema Association

Acto seguido, se añadirá la instrucción de procesamiento

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

donde reemplazaremos file.xsd por la ruta al fichero que contiene el esquema XML.



En la documentación de xml-model se puede obtener más información.

Si el documento XML tiene su propio espacio de nombres, como es el caso de alumnos_ns_base.xml,

```xml
<?xml version="1.0" encoding="utf-8"?>
<alumnos xmlns="http://lmsgi.edu/ud02">
    <alumno>
        <nombre>Jose Ramón</nombre>
        <apellidos>García González</apellidos>
        <direccion>
            <domicilio>El Pez, 12</domicilio>
            <codigo_postal cp="85620"/>
            <localidad>Suances</localidad>
            <provincia>Cantabria</provincia>
        </direccion>
        <contactar>
            <telf_casa>985623165</telf_casa>
            <telf_movil>611233544</telf_movil>
            <telf_trabajo>965847536</telf_trabajo>
            <email href="pepito@educadistancia.com"/>
        </contactar>
    </alumno>
    <alumno>
        <nombre>Carlos</nombre>
        <apellidos>López Pérez</apellidos>
        <direccion>
            <domicilio>El Cangrejo, 25</domicilio>
            <codigo_postal cp="86290"/>
            <localidad>Santillana</localidad>
            <provincia>Cantabria</provincia>
        </direccion>
        <contactar>
            <telf_casa>931132565</telf_casa>
            <telf_movil>623863544</telf_movil>
            <telf_trabajo>984657536</telf_trabajo>
            <email href="carlos@educadistancia.com"/>
        </contactar>
    </alumno>
</alumnos>
```

entonces la vinculación entre el documento XML y el esquema ha de realizarse de la siguiente forma:

- En el **documento XML**, el **elemento raíz** irá acompañado de:
  - El **espacio de nombres del propio documento XML**, por ejemplo: **xmlns="http://lmsgi.edu/ud02"**
  - **El atributo xsi:schemaLocation="http://lmsgi.edu/ud02 alumnos_ns_por_defecto.xsd":** Fijémonos en que, en primer lugar se usa el espacio de nombres del documento XML **seguido de un espacio** seguido de la ruta al esquema. En este caso, se supone que el documento y el esquema están en la misma ubicación. Se indica así que para validar los elementos calificados por ese espacio de nombres se deben validar con el esquema que se encontrará en el archivo indicado.

- En el esquema XML, **al elemento raíz *schema* se le añade**:
  - **El atributo targetNamespace="http://lmsgi.edu/ud02":** Indica el espacio de nombres **del documento XML que deberá validar contra el presente schema**.
  - **Añadir un targetNamespace no es suficiente**. Los nuevos tipos declarados en el esquema han de pertenecer al mismo espacio de nombres que indica targetNamespace y esto, lo conseguiremos de forma sencilla **declarando un espacio de nombres por defecto en el propio esquema** puesto que no deja de ser un documento XML. Habrá que añadir un espacio de nombres por defecto coincidente con el targetNamespace: **xmlns="http://lmsgi.edu/ud02"**
  - El atributo **elementFormDefault="qualified"** para indicar que los elementos a los que se le aplica el esquema han de estar calificados por el targetNamespace para ser validados. Esto es así en todos los elementos de alumnos_ns_por_defecto.xml, puesto que el espacio de nombres por defecto afecta a todos los elementos. Esta es una forma abreviada de indicar que todos los elementos tienen que estar calificados. También sería posible indicarlo de forma individualizada para cada elemento con el atributo *form="qualified"* o *form="unqualified"*.

Tenéis un ejemplo de este tipo de vinculación en los siguientes ficheros:

- alumnos_ns.xml
- alumnos_ns_por_defecto.xsd

Construcción de un esquema XML

Un esquema XML es un documento XML con:

- un elemento raíz *schema* y
- el espacio de nombres **xmlns:xs="http://www.w3.org/2001/XMLSchema" (recordemos que el prefijo puede variar)**
- declaraciones de elementos ***element*** (**al menos uno**, pues modela un documento XML que siempre ha de tener elemento raíz)
- opcionalmente: declaraciones de atributos ***attribute***

Recordemos el ejemplo que veíamos inicialmente:

Además, existen dos tipos de construcciones principales que veremos a continuación:

- Tipos simples o ***simpleType:*** Se aplican **a todos los atributos y a aquellos elementos que**:
  - no contienen subelementos, no tienen atributos y su contenido es de tipo simple: cadena de texto (string), números, fechas, etc. En definitiva, tipos predefinidos en el esquema XML (que veremos a continuación).

- Tipos complejos o ***complexType:*** Se aplican a cualquiera de los siguientes:
  - Los elementos que tienen subelementos
  - Los elementos que tienen atributos
  - Los elementos vacíos
  - Los elementos que tienen contenido mixto (texto y subelementos)

Declaración del elementos y atributos.

Para declarar un elemento se usaría esta sintaxis:

```
<xs:element name="xxx" type="yyy" default="zzz"> o <xs:element name="xxx" type="yyy" fixed="vvv">
```

Es obligatorio **name** para dar nombre al elemento.

- **type**: tipo de dato que contiene el elemento. Aparecerá salvo que se utilice un tipo de datos anónimo (un *simpleType* sin atributo *name* o un *complexType* sin atributo *name*). En caso de que no aparezca ni type ni simpleType ni complexType, se asumirá valor por defecto xs:string.

Son opcionales y exclusivos (si aparece uno no puede aparecer el otro):

- **default**: valor por defecto
- **fixed**: valor asignado de forma automática o fija

Por ejemplo:

```
<xs:element name="anho" type="xs:gYear" default="2000">
```

Para declarar un **atributo** se usaría la sintaxis:

```
<xs:attribute name="xxx" type="yyy" fixed="vvv" use="required">
```

- Se mantienen las consideraciones anteriores y, además, el atributo ***use* es opcional**. Si no existe *use*, el atributo que se está definiendo (perteneciente al documento XML que se va a validar con el esquema XML) será opcional.

Por ejemplo, el atributo top es obligatorio y de tipo entero:

```
<xs:attribute name="top" type="xs:integer"  use="required">
```

Si fuese opcional, omitiríamos *use*:

```
<xs:attribute name="top" type="xs:integer">
```

La posición de la declaración de los atributos, **siempre va tras los subelementos, si los hubiese**:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="top_billboard_espanhol">
4          <xs:complexType>
5              <xs:sequence>
6                  <xs:element name="cancion" minOccurs="0" maxOccurs="unbounded">
7                      <xs:complexType>
8                          <xs:sequence>
9                              <xs:element name="titulo" type="xs:string" />
10                             <xs:element name="interprete" minOccurs="0" maxOccurs="unbounded" type="xs:string" />
11                             <xs:element name="anho" type="xs:gYear" default="2000" />
12                         </xs:sequence>
13                         <xs:attribute name="top" type="xs:positiveInteger" use="required" />
14                     </xs:complexType>
15                 </xs:element>
16             </xs:sequence>
17         </xs:complexType>
18     </xs:element>
19
20
21 </xs:schema>
```

Multiplicidad de elementos

Es posible indicar el número de veces que se puede repetir un elemento en el documento XML con los siguientes atributos:

- **minOccurs**: El mínimo nº de veces que puede ocurrir en el documento XML. Por defecto, si se omite, el valor es 1. El valor ilimitado se representa con *unbounded*
- **maxOccurs**: El máximo nº de veces que puede ocurrir en el documento XML. Por defecto, si se omite, el valor es 1. El valor ilimitado se representa con *unbounded*

Tipos simples predefinidos

- Se pueden consultar los tipos predefinidos para un esquema XML en la imagen del estándar. (Si hacéis clic sobre el nombre de cada tipo en la imagen del estándar, se puede consultar información detallada de cada tipo).
- En los esquemas XML, cuando se haga referencia a cada tipo a través del atributo type, cada tipo predefinido deberá **ir acompañado del prefijo del espacio de nombres utilizado para** "http://www.w3.org/2001/XMLSchema". En los ejemplos vamos a suponer que el prefijo es **xs.**
- Los tipos simples predefinidos más destacables son:
  - **string**: para cadenas de texto genéricas

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

  - numéricos:
    - **integer**: para enteros en general

```
 <xs:attribute name="top" type="xs:integer" use="required" />
```

      - **positiveInteger** (5) o **nonPositiveInteger** (-5)
    - **float** y **double**: para decimales de simple y doble precisión (32 o 64 bits)**.** Permite también notación científica.
    - **decimal** para decimales de longitud arbitraria.
  - Para fechas:
    - **date**: para fechas en formato ISO 8601 (2021-11-23)
    - **time**: formato "hh:mm:ss"(08:00:00)
    - **dateTime**: formato "YYYY-MM-DDThh:mm:ss" (2021-11-23T08:20:00)
    - **gDay**, **gMonth** para la representación en dos dígitos respectivamente (23 y 11) y **gYear** para la representación en 4 dígitos del año (2021)

```
<xs:element name="anho" type="xs:gYear" />
```

    - **gYearMonth** y **gMonthDay**: Noviembre del 2021 => 2021-11 y el 4 de julio (--07-04)
    - **duration:** duración con el formato P*n*Y*n*M*n*DT*n*H*n*M*n*S donde
      - **P es obligatorio**.
      - Y indica años (year)
      - la 1ª M indica meses (month)
      - D indica días (day)
      - T sería obligatorio si se van a indicar horas, minutos o segundos
      - H indica horas
      - la 2ª M indica minutos
      - S indica segundos (pueden tener parte decimal)

Ejemplos válidos podrían ser: P1347Y, P1347M y P1Y2MT2H. Incluso `-P120D` para indicar que es una duración negativa de -120 días.

  - **boolean**: true o false o también 1 ó 0
  - **anyURI** para una cadena de texto que siga el formato de una URI
  - **ID, IDREF, IDREFS, NMTOKEN y NMTOKENS** para compatibilidad con DTDs (con su mismo significado)

Tipos simples con restricciones

Las restricciones o facets (facetas) nos permiten limitar los posibles valores de un tipo simple de datos.

Por ejemplo, para restringir el formato de un DNI con 8 dígitos, seguido de un guion medio y de una letra mayúscula:

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

- En este caso, se utiliza un ***simpleType*** con un nombre **tipo_dni** que escogemos nosotros. De esta forma podrá ser reutilizado por varios elementos o atributos que usen este tipo de datos.
- Se indica el tipo predefinido base sobre el que se aplicará la restricción, en este caso *string*
- A continuación, el tipo de restricción que se aplicará: En este caso, es un patrón o *pattern* que tiene una sintaxis similar a la de las expresiones regulares:
  - Con los corchetes indicamos un rango de valores posibles
  - Con las llaves indicamos el nº de veces que han de repetirse
  - Los caracteres literales se usan tal y como se desea que aparezcan

En este caso, se permiten dígitos del 0 al 9 que se deben repetir 8 veces, seguidos de un guion medio y finalmente una letra que deberá estar en el rango de letras de la A mayúscula a la Z mayúscula.

Algunos elementos para construir patrones sobre el tipo string

Patrón	Significado
`[A-Za-z]`	Letra.
`[A-Z]`	Letra mayúscula.
`[a-z]`	Letra minúscula.
`[0-9]`	Dígitos decimales.
`\D`	Cualquier carácter excepto un dígito decimal.
`(A)`	Cadena que coincide con A.
`A | B`	Cadena que es igual a la cadena A o a la B.
`AB`	Cadena que es la concatenación de las cadenas A y B.
`A?`	Cero o una vez la cadena A.
`A+`	Una o más veces la cadena A.
`A*`	Cero o más veces la cadena A.
`[abcd]`	Alguno de los caracteres que están entre corchetes.
`[^abcd]`	Cualquier carácter que no esté entre corchetes.
`\t`	Tabulación.
`\n`	Retorno de carro
`\d`	Cualquier dígito. Equivalente a [0-9]
`.`	Cualquier carácter excepto retorno de carro (\r) o nueva línea (\n)
`\\`	Barra invertida
`\?`	El interrogante de cierre
`\*`	El carácter asterisco
`\\|`	La barra vertical
`<`	Menor que (<)

Una lista más completa de ejemplos de caracteres para la formación de expresiones regulares se puede encontrar aquí

Vamos a ir realizando estos cambios en el esquema generado a partir del archivo alumno_base.xml.

Para restringir los valores de los posibles módulos de formación profesional que comienzan por MP y van seguidos de 4 dígitos, por ejemplo: MP0373

```
<xs:simpleType name="tipo_codigo_modulo">
```

```
        <xs:restriction base="xs:string">
        <xs:pattern value="MP[0-9]{4}"></xs:pattern>
        </xs:restriction>
    </xs:simpleType>
```

Para una calificación entre 0 y 10 con hasta 2 números decimales:

```
    <xs:simpleType name="tipo_calificacion">
        <xs:restriction base="xs:decimal">
            <xs:fractionDigits value="2"></xs:fractionDigits>
            <xs:totalDigits value="3"></xs:totalDigits>
            <xs:minInclusive value="0"></xs:minInclusive>
            <xs:maxInclusive value="10"></xs:maxInclusive>
        </xs:restriction>
    </xs:simpleType>
```

Para enumerar un número **limitado de valores** que puede tomar **un elemento o un atributo** podríamos usar una restricción con una enumeración de los posibles valores:

```
    <xs:simpleType name="tipo_genero">
        <xs:restriction base="xs:string">
            <xs:enumeration value="masculino"></xs:enumeration>
            <xs:enumeration value="femenino"></xs:enumeration>
        </xs:restriction>
    </xs:simpleType>
```

O también sería posible con un pattern:

```
  <xs:simpleType name="tipo_genero">
        <xs:restriction base="xs:string">
           <xs:pattern value="femenino|masculino"></xs:pattern>
        </xs:restriction>
    </xs:simpleType>
```

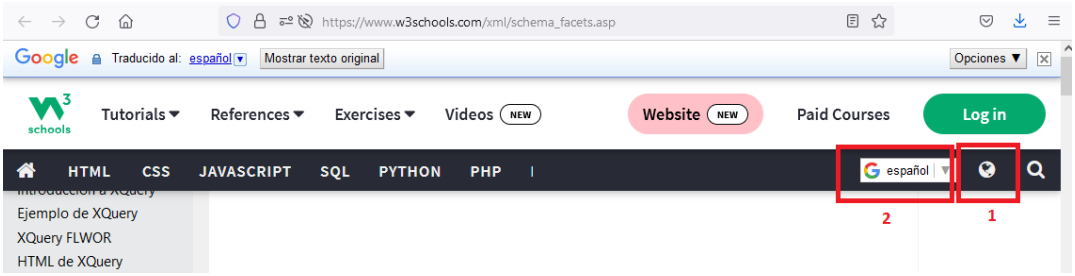
El resultado final se puede observar en **alumnado.xsd** y **alumnado.xml**.

Podéis encontrar otros tipos de restricciones y más ejemplos aquí.

Para traducirlo:

1- Clic sobre el icono de la Tierra "Trasnlate W3Schools" en la parte superior derecha de la página.

2-Elegid el idioma de destino



Las restricciones disponibles para cada tipo, se pueden consultar en el estándar, haciendo clic sobre cada tipo.

Modo de declarar los elementos

Los elementos y tipos pueden declararse de forma **global** o **local** a la raíz del documento ***schema***:

- **globalmente**: hijos directos de la raíz schema y **con el atributo name**, lo que permite su reutilización.
- **localmente**: incluida dentro de la estructura del esquema XML anidada dentro de otra declaración. Estas declaraciones generarán **tipos anónimos** (**carecen del atributo *name*).** Veremos que la herramienta de generación de esquemas automática hace este tipo de anidaciones con declaraciones locales y anónimas.

Además de la creación de tipos globales, es posible declarar **un elemento de forma global con name** y en lugar de reutilizar el tipo, hacer referencia al elemento con el atributo ***ref***. Obsérvese que en ese caso, el elemento que será sustituido por la referencia carece de atributo ***name:***

Tipos complejos

Tipos complejos o ***complexType*** se aplican a cualquiera de los siguientes casos:

1. **Los elementos que tienen subelementos:**

Por ejemplo, para el documento XML top_billboard_espanhol_subelementos.xml, se podría usar una definición del elemento *cancion* con 3 subelementos a través de un ***complexType* anónimo** (sin name) y local:

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

Otra alternativa, sería crear un tipo complejo con nombre global a través del atributo *name* y reutilizarlo con el atributo *type* del elemento con name="cancion" de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion">
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_cancion">
        <xs:sequence>
            <xs:element name="titulo" type="xs:string" />
            <xs:element name="interprete" maxOccurs="unbounded" type="xs:string" />
            <xs:element name="anho" type="xs:gYear" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Esta segunda forma, con tipos complejos con nombre, es la **preferida**, pues:

- permite la reutilización de tipos
- personalmente me resulta más legible, en caso de documentos con un mayor nivel de anidación.

2. **Los elementos que tienen atributos**

Para el caso de un elemento que solo tiene atributos, como el fichero top-billboard_atributos.xml:

El resultado de la generación automática será el elemento *cancion* como tipo complejo anónimo local:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="top" type="xs:integer" use="required" />
                        <xs:attribute name="title" use="required" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Una versión equivalente con tipo complejo con *name* y global sería la siguiente:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion">
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_cancion">
        <xs:attribute name="top" type="xs:integer" use="required" />
        <xs:attribute name="title" use="required" />
    </xs:complexType>
</xs:schema>
```

3. Los elementos vacíos

Imaginemos que tenemos un documento XML con elementos vacíos como top-billboard_cancion_empty.xml.
Para representar los elementos vacíos se pueden usar**:**

- **elementos complejos anónimos vacíos**:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded">
                    <xs:complexType></xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- o simplemente la **declaración del elemento vacía:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

4. Los elementos que tienen contenido mixto (texto y subelementos)

Por ejemplo, el documento XML **top-billboard_mixto.xml** que mezcla texto libre (para el título de la canción) con subelementos *interprete* y anho.

En ese caso, **complexType debe llevar el atributo mixed="true".**
En la siguiente imagen, se ve la versión anónima y local del complexType:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded">
                    <xs:complexType mixed="true">
                        <xs:sequence>
                            <xs:element name="interprete" type="xs:string" />
                            <xs:element name="anho" type="xs:string" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

O equivalentemente, un complexType global con name:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion">

                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType mixed="true" name="tipo_cancion">
        <xs:sequence>
            <xs:element name="interprete" type="xs:string" />
            <xs:element name="anho" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

5. **Los elementos que tienen atributos y pueden contener texto entre la etiqueta de apertura y cierre**

En el documento XML top-billboard_atributos-texto.xml vemos un ejemplo.

En este caso, el complexType, debe contener un elemento *simpleContent* que a su vez contiene un elemento *extension* (*hereda* de un tipo base) cuya base es el tipo de cadena de texto: string . El atributo va dentro de esa extensión.

En la siguiente imagen, se puede ver el *complexType* anónimo y local:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="top" type="xs:integer" use="required" />
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

La versión equivalente global con *name* :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion"></xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_cancion">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="top" type="xs:integer" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

Tipos complejos: Compositores de subelementos

Un tipo complejo complexType puede contener los siguientes compositores:

- **sequence:** define un **grupo ordenado** de subelementos que deben **seguir el orden en el que aparecen.** Por ejemplo, dentro del elemento *cancion* de *tipo_cancion*, deben aparecer *titulo, interprete* y *anho* ense orden:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion">
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_cancion">
        <xs:sequence>
            <xs:element name="titulo" type="xs:string" />
            <xs:element name="interprete" maxOccurs="unbounded" type="xs:string" />
            <xs:element name="anho" type="xs:gYear" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

- **choice:** define un grupo de subelementos de entre los cuales **solo puede aparecer uno de ellos**

Por ejemplo, en el documento agenda.xml, un contacto puede tener como 4º elemento o telefono o email:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="agenda">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="contacto" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nombre" type="xs:string" />
                            <xs:element name="apellido1" type="xs:string" />
                            <xs:element name="apellido2" type="xs:string" />
                            <xs:choice>
                                <xs:element name="telefono" minOccurs="0" type="xs:string"/>
                                <xs:element name="email" minOccurs="0" type="xs:string" />
                            </xs:choice>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- **all:** define un grupo de subelementos que puede aparecer **en cualquier orden** (maxOccurs solo puede ser 0 o 1). En este caso se indica que deben aparecer los 3 subelementos de cancion, pero esta vez en cualquier orden, por ejemplo, all_cancion.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="top_billboard_espanhol">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" maxOccurs="unbounded" type="tipo_cancion">
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_cancion">
        <xs:all>
            <xs:element name="titulo" type="xs:string" />
            <xs:element name="interprete" type="xs:string" />
            <xs:element name="anho" type="xs:gYear" />
        </xs:all>
    </xs:complexType>
</xs:schema>
```

Tanto **choice** como **sequence** pueden combinarse y anidarse entre sí para crear estructuras más complejas. Por ejemplo, esto sería el equivalene de una DTD

<!ELEMENT contacto (nombre, apellido1, apellido2, (telefono*|email) >

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="agenda">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="contacto" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nombre" type="xs:string" />
                            <xs:element name="apellido1" type="xs:string" />
                            <xs:element name="apellido2" type="xs:string" />
                            <xs:choice>
                                <xs:element name="telefono" minOccurs="0" maxOccurs="unbounded" type="tc_ti
                                <xs:element name="email" minOccurs="0" type="xs:string" />
                            </xs:choice>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tc_tipo-telefono">
        <xs:simpleContent>
            <xs:extension base="ts_tipo_telefono">
                <xs:attribute name="tipo" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:simpleType name="ts_tipo_telefono">
        <xs:restriction base="xs:string">
            <xs:pattern value="(9\d{8})|(6\d{8})"></xs:pattern>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Sin embargo, **all no puede combinarse ni con choice ni con sequence**, solo puede tener como hijos elementos **element** y sus **minOccurs** y **maxOccurs** solo pueden tomar valores 0 ó 1.

Grupos

Es posible crear un **grupo de elementos o un grupo de atributos que se repitan** y luego hacer referencia a los grupos mediante su nombre y el atributo ***ref***.
Se declaran a nivel global con con **xs:group** y **xs:attributeGroup** respectivamente.

Grupo de elementos

Dentro de un grupo de elementos encontramos los compositores:

- sequence
- all
- choice
- o combinaciones de los compositores

Por ejemplo, para el documento carta_el_base.xml

```xml
<carta>
    <emisor>
        <nombre>Ana </nombre>
        <apellidos>López Mar</apellidos>
        <tipo_via>Avenida</tipo_via>
        <nombre_via>de Vigo</nombre_via>
        <numero>2</numero>
        <piso>1</piso>
        <letra>D</letra>
    </emisor>
    <receptor>
        <nombre>Beatriz </nombre>
        <apellidos>Luz Serrano</apellidos>
        <tipo_via>Calle</tipo_via>
        <nombre_via>San Juan</nombre_via>
        <piso>Bajo</piso>
        <letra>E</letra>
    </receptor>
</carta>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="carta">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="emisor">
                    <xs:complexType>
                        <xs:group ref="gr_direccion"></xs:group>
                    </xs:complexType>
                </xs:element>
                <xs:element name="receptor">
                    <xs:complexType>
                        <xs:group ref="gr_direccion"></xs:group>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:group name="gr_direccion">
        <xs:sequence>
            <xs:element name="nombre" type="xs:string" />
            <xs:element name="apellidos" type="xs:string" />
            <xs:element name="tipo_via" type="xs:string" />
            <xs:element name="nombre_via" type="xs:string" />
            <xs:element name="numero" type="xs:string" />
            <xs:element name="piso" type="xs:string" />
            <xs:element name="letra" type="xs:string" />
        </xs:sequence>
    </xs:group>
</xs:schema>
```

Se crea un elemento global ***group*** *que envuelve,* en este caso al compositor *sequence*. El grupo de elementos se utiliza dentro de un ***complexType*** a través del atributo ***ref.***

Grupos de atributos

Dentro de un grupo de atributos ***attributeGroup*** , encontraremos los elementos *attribute* correspondientes:
Por ejemplo, para el archivo carta_base.xml:

```xml
<carta>
    <emisor>
        <nombre>Teresa</nombre>
        <direccion calle="Gran vía" numero="25" ciudad="Vigo"  ></direccion>
    </emisor>
    <receptor>
        <nombre>Lucía</nombre>
        <direccion calle="Toledo"  numero="11" ciudad="Santiago de Compostela"></direccion>
    </receptor>
</carta>
```

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

    <xs:attributeGroup name="gr_ats_direccion">
        <xs:attribute name="calle" use="required" />
        <xs:attribute name="numero" type="xs:integer" use="required" />
        <xs:attribute name="ciudad" use="required" />
    </xs:attributeGroup>

    <xs:complexType name="tipo_interlocutor">
        <xs:sequence>
            <xs:element name="nombre" type="xs:string" />
            <xs:element name="direccion">
                <xs:complexType>
                    <xs:attributeGroup ref="gr_ats_direccion"></xs:attributeGroup>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

De nuevo, el grupo de atributos se utiliza dentro de un ***complexType*** a través del atributo ***ref***.

Extensiones

Las extensiones permiten crear un nuevo tipo de datos a partir de un tipo de datos base. El tipo base puede ser simple o complejo.
Si es simple, se usará *simpleContent* y si es complejo se usará *complexContent*.
Es posible **añadir atributos y/o añadir elementos al tipo base** en el nuevo tipo de datos.

Por ejemplo, en el archivo empleados.xml:

```xml
<empleados xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="empleados.xsd">
    <empleado id="a10001">
        <nombre>Rita</nombre>
        <apellidos>López Ruíz</apellidos>
        <dni>12345678-A</dni>
        <beneficiario>
            <nombre>Sara</nombre>
            <apellidos>Martínez López</apellidos>
            <dni>12345679-J</dni>
        </beneficiario>
        <beneficiario>
            <nombre>José</nombre>
            <apellidos>Martínez López</apellidos>
            <dni>43215679-L</dni>
        </beneficiario>
    </empleado>

    <empleado id="a10002">
        <nombre>Tomás</nombre>
        <apellidos>Hernández Trillo</apellidos>
        <dni>87654321-B</dni>
    </empleado>
</empleados>
```

Cada empleado cuenta con un atributo id único en todo el documento y tiene como subelementos:

- nombre
- apellidos
- dni
- 0 ó n elementos beneficiario que a su vez tienen:
    - nombre
    - apellidos
    - dni

Los subelementos o atributos comunes podemos englobarlos en un **tipo_persona que será un tipo complejo base** y extender ese tipo añadiendo el atributo id y los elementos beneficiario, que en este caso, también son de **tipo_persona**:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="empleados">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="empleado" maxOccurs="unbounded" type="tipo_empleado">
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="tipo_persona">
        <xs:sequence>
            <xs:element name="nombre" type="xs:string" />
            <xs:element name="apellidos" type="xs:string" />
            <xs:element name="dni" type="xs:string" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="tipo_empleado">
        <xs:complexContent>
            <xs:extension base="tipo_persona">
                <xs:sequence>
                    <xs:element name="beneficiario" type="tipo_persona" minOccurs="0" maxOccurs="unbounded"></xs:element>
                </xs:sequence>
                <xs:attribute name="id" type="xs:ID" use="required" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

Se pueden consultar otros dos ejemplos de extensiones aquí

Documentación: Annotation

Como los procesadores de XML no están obligados a conservar los comentarios XML (<!-- esto es un comentario -->), en los esquemas XML, se ha definido un elemento ***annotation***, que puede, a su vez, contener cualquiera de los dos subelementos siguientes (0 ó n veces):

- ***appinfo***: documentación destinada a la **interpretación de programas o aplicaciones**
- ***documentation:*** documentación destinada a la **interpretación por parte de personas**

Ambos subelementos cuentan con un atributo opcional ***source*** para indicar mediante una URL dónde está disponible la documentación en otro formato (HTML, PDF, etc).
Además, ***documentation***, como va dirigiado a los seres humanos, también tiene un atributo opcional ***xml:lang*** para indicar el idioma en el que se encuentra la documentación.

El elemento **annotation** puede situarse:

- a nivel global, como hijo directo de esquema, para indicar información general acerca del esquema
- en cualquier otro elemento, para indicar documentación sobre ese mismo elemento

Pueden contener elementos del esquema XML o cualquier marcado XML bien formado.
No tiene efecto en la validación del documento. Únicamente se pide que ***annotation y sus subelementos*** estén bien formados.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:annotation>
        <xs:appinfo source="http://www.lmsgi.edu/esquemas/documentacion/topbillboard.xml">
            <fecha_creacion>2021-11-01</fecha_creacion>
        </xs:appinfo>
        <xs:documentation xml:lang="es" source="http://www.lmsgi.edu/esquemas/documentacion/topbillboard.pdf">
            <info>Aquí información global sobre el esquema destinado a personas.</info>
        </xs:documentation>
    </xs:annotation>
    <xs:element name="top_billboard_espanhol">
        <xs:annotation>
            <xs:documentation>
                <info>El elemento raíz es top_billboard-español</info>
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="cancion" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="titulo" type="xs:string" />
                            <xs:element name="interprete" minOccurs="0" maxOccurs="unbounded" type="xs:string" />
                            <xs:element name="anho" type="xs:gYear" default="2000" />
                        </xs:sequence>
                        <xs:attribute name="top" type="xs:positiveInteger" use="required" />
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

En la imagen se observan dos elementos **annotation**:

- Uno global con sendos subelementos: **appinfo** y **documentation**
- Uno respecto al elemento raíz, esta vez solo con el subelemento **documentation**.