Introducción a XML Schema

Jorge Castellanos Vega

Introducción

Lenguaje también conocido como XSD (XML Schema Definition).

Su objetivo es describir la estructura de un documento XML:

- Elementos y atributos que pueden aparecer.
- Número y orden de los elementos hijo.
- Tipos de datos de los elementos y atributos.
- Valores por defecto y fijos para cada elemento y atributo.

Ventajas

Soporta tipos de datos por lo tanto es más sencillo:

- Describir los elementos que puede contener el archivo.
- Validar los datos.
- Definir restricciones.
- Definir patrones y formatos de datos.
- Convertir datos entre diferentes tipos.

Ventajas

Usa sintaxis XML, por lo tanto:

- Se pueden usar editores y parsers XML para trabajar con el.
- Se puede trabajar sobre el esquema con el DOM XML.
- Se puede transformar el esquema con XSLT.

Los esquemas XML son extensibles puesto que están escritos en XML, por lo tanto:

- Es posible reutilizar esquemas.
- Crear nuestros propios tipos de datos a partir de tipos estándar.
- Hacer referencia a múltiples esquemas en el mismo documento.

Ventajas

Comunicaciones seguras.

- XML esquema permite al emisor describir cómo son los datos que va a recibir. Por ejemplo si enviamos a un receptor una fecha como 04-11-2018 puede interpretarse como:
  - 4 de noviembre
  - 11 de abril

Dependiendo del país en el que nos encontremos. Sin embargo si tenemos un documento XML con un elemento como:

<fecha type="date">2018-11-04</fecha>

Estaremos asegurando el entendimiento ya que el tipo de datos "date" exige ajustarse al formato "YYYY-MM-DD" (año, mes, día).

DTD vs XML SCHEMA

Supongamos el siguiente documento XML

```
<?xml version="1.0"?>
<email>
  <de>Jorge</de>
  <para>Bea</para>
  <asunto>Fiesta el viernes</asunto>
  <cuerpo>Recuerda que el viernes no hay clase!</cuerpo>
</email>
```

DTD vs XML SCHEMA

Un DTD para definir el archivo anterior podría ser:

```
<!ELEMENT email (de, para, asunto, cuerpo)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT cuerpo (#PCDATA)>
```

Y se haría referencia desde el archivo XML así:

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email>
  <de>Jorge</de>
  <para>Bea</para>
  <asunto>Fiesta el viernes</asunto>
  <cuerpo>Recuerda que el viernes no hay clase!</cuerpo>
</email>
```

DTD vs XML SCHEMA

El equivalente XSD al DTD del ejemplo anterior sería:

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

DTD vs XML SCHEMA

Y se haría referencia desde el archivo XML así:

Indica al parser XML que el documento debe ser validado frente a un esquema

```
<?xml version="1.0"?>
<email xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="email.xsd">
<de>Jorge</de>
  <para>Bea</para>
  <asunto>Fiesta el viernes</asunto>
  <cuerpo>Recuerda que el viernes no hay clase!</cuerpo>
</email>
```

Especifica dónde está el esquema (misma carpeta que el archivo XML en el archivo **email.xsd**)

¿qué son esas etiquetas y cómo se utilizan? Dependen de si se utilizan **espacios para nombre** o no

Espacios para nombres

En XML los nombres de los elementos y atributos son definidos por el desarrollador, esto puede ocasionar un problema cuando se intentan mezclar documentos XML de diferentes orígenes.

Veamos estos dos archivos XML.

```
<email>
  <de>Jorge</de>
  <para>Bea</para>
  <asunto>Fiesta el viernes</asunto>
  <cuerpo>Recuerda que el viernes no hay clase!</cuerpo>
</email>
```

```
<agenda>
     <nombre>Pedro</nombre>
          <email>
          <usuario>pedro</usuario>
          <dominio>educa.madrid.org</dominio>
          </email>
</agenda>
```

En ambos casos hay un elemento email, sin embargo en cada caso tiene diferente contenido y significado. Si hubiese que unir estos dos archivos XML sería complicado detectar y manejar las diferencias.

Espacios para nombres

Podríamos resolver el problema usando un prefijo:

```
<e:email>
    <e:de>Jorge</e:de>
    <e:para>Bea</e:para>
    <e:asunto>Fiesta el viernes</e:asunto>
    <e:cuerpo>Recuerda que el viernes no hay clase!</e:cuerpo>
</e:email>
<a:agenda>
    <a:nombre>Pedro</a:nombre>
    <a:email>
        <a:usuario>pedro</a:usuario>
        <a:dominio>educa.madrid.org</a:dominio>
    </a:email>
</a:agenda>
```

Espacios para nombres

En XML cuando usamos prefijos debe definirse un espacio para nombres para el prefijo

Se define mediante el atributo **xmlns** (XML NameSpace) en la etiqueta inicial del elemento.

- También puede definirse en el elemento raiz

La declaración tiene la siguiente sintaxis: **xmlns:prefijo="URI"**

URI – Uniform Resource Identifier (Identificador Uniforme de Recursos). Cadena de caracteres que identifica unívocamente un recurso.

El parser en ningún caso busca información en el URI, simplemente se utiliza para dar un nombre único al espacio para nombres.

- Algunas compañías usan el URI para apuntar a una página web en la que habrá información sobre el espacio para nombres.

Espacios para nombres

Diferentes posibilidades de definición.

```
<raiz>
<e:email xmlns:e="http://educa.madrid.org/correos/">
<e:de>Jorge</e:de>
      <e:para>Bea</e:para>
      <e:asunto>Fiesta el viernes</e:asunto>
      <e:cuerpo>Recuerda que el viernes no hay clase!</e:cuerpo>
</e:email>
<a:agenda xmlns:e="http://educa.madrid.org/jorge/agenda/">
<a:nombre>Pedro</a:nombre>
      <a:email>
            <a:usuario>pedro</a:usuario>
            <a:dominio>educa.madrid.org</a:dominio>
      </a:email>
</a:agenda>
</raiz>
```

```
<raíz xmlns:e="http://educa.madrid.org/correos/"
xmlns:e="http://educa.madrid.org/jorge/agenda/">
<e:email >
<e:de>Jorge</e:de>
      <e:para>Bea</e:para>
      <e:asunto>Fiesta el viernes</e:asunto>
      <e:cuerpo>Recuerda que el viernes no hay clase!</e:cuerpo>
</e:email>
<a:agenda > <a:nombre>Pedro</a:nombre>
      <a:email>
            <a:usuario>pedro</a:usuario>
            <a:dominio>educa.madrid.org</a:dominio>
      </a:email>
</a:agenda>
</raiz>
```

Vinculando archivos XML y XSD

Sin espacio para nombres

XML

```
<raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="esquemaAsociado.xsd">
<!-- ... --> </raiz>
```

XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="raiz">
<!-- ... -->
</xsd:element>
</xsd:schema>
```

Vinculando archivos XML y XSD

Con espacio para nombres

XML

```
<raiz xmlns="http://ejemplo.com/ns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ejemplo.com/ns esquemaAsociado.xsd">
<!-- ... -->
</raiz>
```

Espacio de nombres a usar

Ubicación del archivo xsd para usar con el espacio para nombres

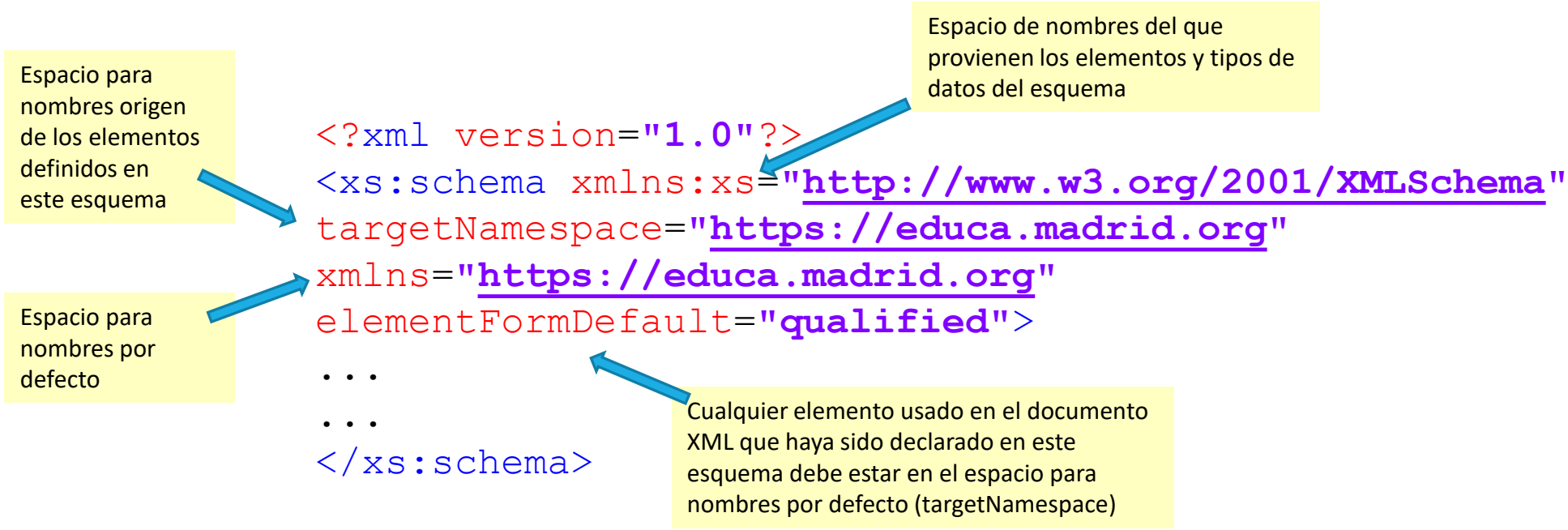
XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ejemplo.com/ns">
<xsd:element name="raiz">
<!-- ... -->
</xsd:element>
</xsd:schema>
```

El elemento <schema>

Elemento raíz de cada esquema XML

Puede contener varios atributos



Elementos simples XSD

Un elemento simple contiene solo texto, no puede contener otros elementos o atributos.

Definición:

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

name es el nombre del elemento y **type** el tipo de datos al que se corresponde. Entre los tipos de datos más comunes encontramos:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

default y **fixed** son opcionales, **default** representan el valor por defecto que se asigna al elemento cuando no recibe ningún valor y **fixed** indica un valor automático que se asigna al elemento y que no permite especificar otro.

Atributos XSD

Los atributos se declaran siempre como tipos simples.

Definición:

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

name es el nombre del atributo y **type** el tipo de datos al que pertenece. Al igual que los elementos simples puede pertenecer a tipos de datos como:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

default y **fixed** son opcionales, **default** representan el valor por defecto que se asigna al elemento cuando no recibe ningún valor y **fixed** indica un valor automático que se asigna al elemento y que no permite especificar otro.

Los atributos son opcionales. Para indicar que es obligatorio se debe especificar mediante `use="required"`

Restricciones XSD

Las **restricciones (o facetas)** permiten definir el rango de valores aceptable para elementos o atributos XML.

Un ejemplo: Restricción para los valores del elemento minutos entre 0 y 59.

```
<xs:element name="minutos">
    <xs:simpleType>
          <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="59"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restricciones XSD

Restringir los valores de un elemento para que tengan que estar dentro de un conjunto de valores.

```
<xs:element name="deporte">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fútbol"/>
      <xs:enumeration value="Baloncesto"/>
      <xs:enumeration value="Tenis"/>
      <xs:enumeration value="Balonmano"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones XSD

Se pueden definir **restricciones de manera independiente al elemento** para que puedan ser usados por otros elementos.

```
<xs:element name="deporte" type="tipoDeporte">

<xs:simpleType name="tipoDeporte">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fútbol"/>
      <xs:enumeration value="Baloncesto"/>
      <xs:enumeration value="Tenis"/>
      <xs:enumeration value="Balonmano"/>
    </xs:restriction>
  </xs:simpleType>
```

El tipo del elemento debe ser igual al nombre de la restricción

Restricciones XSD

Restricciones de patrones. Es posible definir un patrón al que deben ajustarse los valores introducidos.

- En este caso solo se podrá introducir una letra minúscula

```
<xs:element name="minúscula">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones XSD

Otras restricciones de patrones.

```
<!-- 3 LETRAS MAYÚSCULAS Y 3 NÚMEROS P.E:ABC123 -->
<xs:pattern value="[A-Z][A-Z][A-Z][0-9][0-9][0-9]"/>
<!-- 2 LETRAS MAYÚSCULAS O MINÚSCULAS seguidas de una entre xyz, P.E:bCy -->
<xs:pattern value="[a-zA-Z][a-zA-Z][xyz]"/>
<xs:pattern value="([a-z])*"/> <!-- 0 ó más letras minúsculas -->
<xs:pattern value="([a-z])+"/> <!-- 1 ó más letras minúsculas -->
<!-- exactamente 10 caracteres, cada uno de
     ellos podrá ser un número, letra mayúscula o minúscula -->
<xs:pattern value="[a-zA-Z0-9]{10}"/>
```

Restricciones XSD

Restricciones de longitud. Es posible establecer la longitud exacta de un elemento, también se puede especificar un mínimo y un máximo.

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

```
<xs:element name="nombreUsuario">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="6"/>
      <xs:maxLength value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones XSD

Tabla resumen.

Restricción	Descripción
enumeration	Define una lista de valores aceptados
fractionDigits	Máximo número de decimales.
length	Número exacto de caracteres.
maxExclusive	Valor máximo para elementos numéricos
maxInclusive	Valor máximo para elementos numéricos (el propio límite también es aceptado)
maxLength	Máximo número de caracteres
minExclusive	Valor mínimo para elementos numéricos
minInclusive	Valor mínimo para elementos numéricos (el propio límite también es aceptado)
minLength	Mínimo número de caracteres
pattern	Patrón al que deben ajustarse los valores introducidos
totalDigits	Número total de dígitos permitidos.

Extensiones

Es posible derivar tipos a partir de otros tipos de datos (ya sean simples o complejos) mediante la creación de extensiones. Por ejemplo dados estos dos archivos:

```
<?xml version="1.0" encoding="UTF-8"?>
<talla xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="extensiones1.xsd">S</talla>
```

El código XML superior valida contra el documento XSD de la derecha

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

<xs:element name="talla" type="tallaCamiseta"/>

<xs:simpleType name="tallaCamiseta">
  <xs:restriction base="xs:string">
    <xs:enumeration value="S" />
    <xs:enumeration value="M" />
    <xs:enumeration value="L" />
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Extensiones

En el ejemplo creamos un nuevo tipo complejo **tipoUniforme** a partir del anterior.

- El nuevo tipo extiende el tipo base **tallaCamiseta**
- Además añade un atributo nuevo
- Un ejemplo de XML que valida con este archivo XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<miUniforme
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="extensiones2.xsd"
colorPantalón="rojo">S</miUniforme>
```

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

<xs:element name="miUniforme" type="tipoUniforme"/>

<xs:simpleType name="tallaCamiseta">
 <xs:restriction base="xs:string">
  <xs:enumeration value="S" />
  <xs:enumeration value="M" />
  <xs:enumeration value="L" />
 </xs:restriction>
</xs:simpleType>

<xs:complexType name="tipoUniforme">
 <xs:simpleContent>
  <xs:extension base="tallaCamiseta">
   <xs:attribute name="colorPantalón">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="rojo" />
      <xs:enumeration value="azul" />
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>

</xs:schema>-
```

Elementos complejos XSD

Un elemento complejo contiene otros elementos y/o atributos.

Hay cuatro tipos:

- Elementos vacíos.
- Elementos que contienen otros elementos.
- Elementos que contienen solo texto.
- Elementos que contienen otros elementos y texto.

```
<coche matrícula="1345HVF"/>
```

```
<email>
  <de>Jorge</de>
  <para>Bea</para>
  <asunto>Fiesta el viernes</asunto>
  <cuerpo>Recuerda que el viernes no hayclase!</cuerpo>
</email>
```

```
<producto talla="L"/>
Camiseta fitness
</producto>
```

```
<producto talla="L"/>
   Camiseta fitness
   <marca> blanca</marca>
</producto>
```

Elementos complejos XSD

Para definir un elemento complejo podemos usar dos métodos.

- Declarando el elemento raíz y definiendo los elementos que lo forman

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

Elementos complejos XSD

Usando un atributo **type** que indique el tipo de datos complejo que forma el elemento.

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

<xs:element name="email" type="estructuraCorreo" />

<xs:complexType name="estructuraCorreo">
    <xs:sequence>
        <xs:element name="de" type="xs:string"/>
        <xs:element name="para" type="xs:string"/>
        <xs:element name="asunto" type="xs:string"/>
        <xs:element name="cuerpo" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

Este método permite que múltiples atributos hagan referencia al mismo tipo complejo simplemente especificando su nombre en el atributo type

Elementos vacíos

Para definir un tipo sin contenidos es necesario definir un tipo que permite elementos en su contenido pero sin declarar ninguno.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="ejemplo3.xsd"
          numero="12345" />
```

XSD

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

Elementos vacíos

Al igual que ocurría en las facetas es posible independizar la declaración del tipo del propio elemento permitiendo reutilizar el tipo en varios elementos.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<empleado xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="ejemplo3bis.xsd"
          numero="12345" />
```

XSD

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

 <xs:element name="empleado" type="tipoEmpleados" />

  <xs:complexType name="tipoEmpleados">
    <xs:attribute name="numero" type="xs:positiveInteger"/>
  </xs:complexType>

</xs:schema>
```

Elementos que solo contienen elementos

La etiqueta <**xs:sequence**> especifica que el orden en el que aparecen los elementos definidos dentro de ella debe respetarse en el documento XML.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="ejemplo4.xsd">
    <nombre>Pedro</nombre>
    <apellido1>García</apellido1>
    <apellido2>Sánchez</apellido2>
</alumno>
```

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="alumno">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido1" type="xs:string"/>
    <xs:element name="apellido2" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Elementos que solo contienen elementos

Evidentemente y al igual que en ejemplos anteriores también es correcto declarar el tipo de manera independiente.

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

  <xs:complexType name="tipoAlumno">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido1" type="xs:string"/>
      <xs:element name="apellido2" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Elementos complejos que solo contienen texto

En este caso debemos definir una extensión con el tipo de elemento contenido.

- También es posible definir restricciones, pero este caso solo cuando se desea limitar el tipo base utilizado para el elemento.

Por ejemplo, queremos validar este archivo XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<jugador xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ejemplo5.xsd"
         numero="11">
    Alberto Herreros
</jugador>
```

Elementos complejos que solo contienen texto

Extensión del elemento contenido.

- El tipo base al que corresponde el elemento contenido es **string**. Lo definimos como tipo base y lo extendemos.

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

Elementos complejos con contenido mixto

Supongamos contenidos en los que se alternan elementos, atributos y texto. Para permitir que se mezclen datos es necesario permitir que el atributo **mixed** esté a valor **True**

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<memoria xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ejemplo6.xsd">
    En este proyecto llamado <b>Amadeus</b>
    se ha contado con la colaboración de la empresa <i>Caligames</i>
    cuyo fundador es <u>Marcos Fernández</u>
</memoria>
```

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="memoria">
 <xs:complexType mixed="true">
  <xs:sequence>
   <xs:element name="b" type="xs:string"/>
   <xs:element name="i" type="xs:string"/>
   <xs:element name="u" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Indicadores XSD

Nos permiten controlar cómo se utilizan los elementos dentro de los documentos XML

- Elementos de orden:
  - **All**. Los elementos hijos pueden aparecer en cualquier orden. Cada elemento solo puede aparecer una vez.
  - **Choice**. Solamente puede aparecer uno de los elementos hijos.
  - **Sequence**. Los elementos hijos deben aparecer en el orden especificado.

```
<xs:all>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="apellido1" type="xs:string"/>
  <xs:element name="apellido2" type="xs:string"/>
</xs:all>
```

```
<xs:choice>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="apellido1" type="xs:string"/>
  <xs:element name="apellido2" type="xs:string"/>
</xs:choice>
```

Indicadores XSD

Indicadores de ocurrencias.

- Nos permiten indicar el número de veces que puede aparecer en elemento.
- El número máximo se especifica con **maxOccurs**
- El número mínimo se especifica con **minOccurs**
- Para indicar que no hay límite se usa la palabra **unbounded**

En la siguiente trasparencia podemos ver un ejemplo

Indicadores XSD

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<equipo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ejemplo7.xsd">
<jugador>
  <nombre>Alvaro González</nombre>
  <apodo>Tripleador</apodo>
</jugador>
<jugador>
  <nombre>Toño Fernandez</nombre>
  <apodo>manos blandas</apodo>
  <apodo>elmanco</apodo>
</jugador>
<jugador>
  <nombre>Eva Reneses</nombre>
</jugador>
</equipo>
```

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="equipo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="jugador" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apodo" type="xs:string"
            minOccurs="0" maxOccurs="2"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Elementos complejos XSD

Un ejemplo, vamos a construir el archivo XSD correspondiente a este archivo XML

```
<?xml version="1.0"?>
<equipos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ejemplo2.xsd">
    <equipo nombreEq="Estudiantes">
        <jugadores>
            <jugador numero="12">
                <nombre>Pedro</nombre>
                <fechaNacimiento>12-12-1987</fechaNacimiento>
                <altura>187</altura>
            </jugador>
            <jugador numero="17">
                <nombre>Roberto</nombre>
                <fechaNacimiento>10-02-1985</fechaNacimiento>
                <altura>183</altura>
            </jugador>
        </jugadores>
    </equipo>
</equipos>
```

Extendiendo con elementos no especificados por el esquema

Se utiliza el elemento **any**

El elemento a incorporar podría estar definido en otro archivo xsd

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="ejemplo8.xsd">
        <nombre>Pedro</nombre>
        <apellido1>García</apellido1>
        <apellido2>Sánchez</apellido2>
        <direccion>Calle Pilar 13</direccion>
</alumno>
```

XSD

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

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

 <xs:element name="alumno">
  <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:any minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Extendiendo con atributos no especificados por el esquema

Se utiliza el elemento **anyAttribute**

Funcionamiento similar a any pero con atributos

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="ejemplo9.xsd"
        direccion="Calle Pilar 13">
        <nombre>Pedro</nombre>
        <apellido1>García</apellido1>
        <apellido2>Sánchez</apellido2>
</alumno>
```

XSD

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

<xs:attribute name="direccion" type="xs:string"/>

 <xs:element name="alumno">
  <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:sequence>
   <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Referencias

https://www.w3schools.com/