UNIDAD 4 DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS EN XML

(PARTE 2 ESQUEMAS)

1 ESQUEMAS

1.1 Introducción a los esquemas XML

Los DTD que hemos visto en la parte 1 tienen algunas limitaciones. En 1999, el W3C empezó a desarrollar los esquemas XML, como una forma mas avanzada para describir documentos XML.

Veamos antes de empezar un sencillo ejemplo, vamos a utilizar el ejemplo de la parte 1, el de los videojuegos.

```
<?xml version="1.0" encoding="UTF-8" ?>
<videojuego>
     <titulo>Resident Evil</titulo>
</videojuego>
```

Usando esquemas, el documento anterior se podría describir de la siguiente forma:

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

Guardaremos el documento anterior como videojuego.xsd.

Hay que tener en cuenta que, en todos los esquemas XML, el elemento raíz es "schema". Ahora bien, para escribirlo es muy común utilizar el prefijo xsd o xs.

Con `xmlns:xs="http://www.w3.org/2001/XMLSchema"` se ha indicado que

- Los elementos y tipos de datos utilizados en el esquema pertenecen al espacio de nombres http://www.w3.org/201/XMLSchema.
- Dichos elementos y tipos de datos deben llevar el prefijo xs (xs:schema, xs:element, ...)

Podemos observar que un esquema sí que es un documento XML válido por sí mismo. Para indicar que el documento XML sigue las reglas dictadas por el anterior esquema, añadimos al documento XML:

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

- Para vincular un esquema a un documento XML, es obligatorio que este último haga referencia al espacio de nombres `http://www.w3.org/2001/XMLSchema-instance`. Para ello, habitualmente se utiliza el prefijo xsi.
- El atributo `noNamespaceSchemaLocation` permite referenciar a un archivo con la definición de un esquema que no tiene ningún espacio de nombres asociado. En este caso, el archivo es videjuego.xsd

Los esquemas definen el vocabulario y las restricciones que controlan la creación de nuevos documentos XML. Un esquema tiene la extensión .xsd

Para validar nuestros documentos XML juntos con sus esquemas asociados vamos a utilizar el editor XML Copy Editor, aunque también se pueden utilizar validadores online.

1.2 Comparación con DTD

Existen una serie de ventajas y desventajas con respecto a los DTD:

Ventajas de los esquemas

- Los esquemas son documentos XML, como tales siguen la sintaxis de XML, por lo que son analizables como cualquier otro documento XML.
- Soportan los espacios de nombres.
- Los esquemas son más potentes, es posible describir con mucho más detalle un documento XML utilizando esquemas que con DTD. Por ejemplo, podemos utilizar distintos tipos de datos en lugar de utilizar sólo cadena de texto como en DTD.
- Soportan conceptos avanzados como herencia y sustitución.

Desventajas de los esquemas

- No es posible describir entidades utilizando esquemas.
- Son más complejos de entender que los DTD.
- Presentan más incompatibilidades con software que las DTD.

1.3 Estructura de los esquemas

Un esquema es un documento XML que tiene la extensión xsd. Al ser un documento XML tiene la estructura habitual de un documento XML con la obligación de que el elemento raíz se llame schema.

Etiqueta schema

Esta etiqueta identifica la raíz del documento XML Schema. En esta etiqueta se declara el espacio de nombres estándar que utilizan los esquemas y que permite diferenciar las etiquetas XML del esquema, respecto a las del documento XML.

En su forma más simple tiene uno de los siguientes formatos:

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

En la declaración de el espacio de nombres se está utilizando el prefijo xs o xsd, con esto se indica que es el alias para referirse a todas las etiquetas de este espacio de nombres Nosotros usaremos la segunda forma.

La etiqueta schema puede tener los siguientes atributos:

- `attributeFormDefault`, puede ser `qualified`, si hacemos que sea obligatorio poner el espacio de nombres antes de los atributos o `unqualified` no lo hacemos. Por defecto es `unqualified`.
- `elementFormDefault`, sirve para definir si es necesario el espacio de nombres delante del nombre puede tomar los valores `qualified` y `unqualified`.
- `targetNamespace`, aquí se indica que los elementos definidos en el esquema provienen de un espacio de nombres.

Ejemplo

```
<?xml version="1.0"?>

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

targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

Asociar un esquema a un documento XML

Para que un documento XML siga las reglas definidas en un esquema, no disponemos de etiqueta ! DOCTYPE; en su lugar utilizamos atributos especiales en el elemento raíz del documento XML.

Primero, al igual que en el documento XMLSchema, necesitamos definir los dos espacios de nombres, el correspondiente al documento XML (que se suele usar sin abreviatura, es decir como espacio por defecto) y el espacio de nombres de XML Schema (que suele utilizar el prefijo **xs**, aunque se puede utilizar otro).

Además es necesario indicar dónde está el archivo XMLSchema que contiene las reglas de validación que se aplican al documento. Esto se hace gracias al atributo llamado **schemaLocation** (perteneciente al espacio de nombres del esquema, por lo que se usa normalmente como **xs:schemaLocation**).

El atributo schemaLocation tiene dos valores separados por un espacio. El primer valor hace referencia a un espacio de nombres En el segundo valor, se tiene que indicar la ubicación de un archivo donde hay un esquema de ese espacio de nombres.

Ejemplo:

```
<?xml version="1.0"?>

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml/  note.xsd">
       <to>Tove</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don't forget me this weekend!</body>
</note>
```

Nosotros usaremos lo que nos da por defecto el programa XML Copy Editor a la hora de asociar un documento XML con un esquema:

```
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="videojuego.xsd">
```

Partes de un esquema XML

Las partes de las que se compone un esquema son las siguientes:

- Elementos, se definen con la etiquetas `xs:element`. Es para indicar los elementos permitidos en los documentos que sigan el esquema.
- Tipos simples, que permiten definir los tipos simples de datos que podrá utilizar el documento XML. Usaremos la etiqueta `xs:simpleType`
- Elementos compuestos, utilizaremos la etiqueta `xs:complexType`
- Atributos, utilizaremos la etiqueta `xs:attribute`.
- Documentación, información utilizable por aplicaciones que manejen los esquemas. Se utilizan las etiquetas `xs:annotation`, `xs:documentation` y `xs:appInfo`.

1.4 Componentes locales y globales

El orden de los elementos en un esquema no es significativo, es decir las declaraciones se pueden hacer en cualquier orden. Pero sí que hay que tener en cuenta que dependiendo de dónde coloquemos la definición de los elementos del esquema, varía su ámbito de aplicación. Se distinguen dos posibilidades **de declarar elementos:**

- En **ámbito global**. Se trata de los elementos del esquema que se coloquen dentro de la etiqueta raíz **schema** y que no están dentro de ninguna otra. Estos elementos se pueden utilizar en cualquier parte del esquema.
- En **ámbito local**. Se trata de elementos definidos dentro de otros elementos. En ese caso se pueden utilizar sólo dentro del elemento en el que están inmersos y no en todo el documento. Es decir si, por ejemplo, si dentro de la definición de un atributo colocamos la definición de un tipo de datos, este tipo de datos sólo se puede utilizar dentro del elemento **xs:attribute** en el que se encuentra la definición del tipo de datos.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:doc="http://www.pepito.net/doc"
       targetNamespace="http://www.pepito.net/doc">
<xs:element …>      <!--Definición global -->
  <xs:simpleType …> <!--Definición local -->
       …
  </xs:simpleType>
…
</xs:element >
<xs:simpleType …> <!--Definición global -->
       …
</xs:simpleType …>
</xs:schema>
```

1.5 Elementos simples

En XML Schema la definición de un elemento XML se realiza mediante la etiqueta xs:element.

Un elemento simple es un elemento XML que solo puede contener texto, no puede contener otros elementos o atributos.

La sintaxis para definir un elemento simple es:

```
<xs:element
      name=”nombre del elemento”
      type=”tipo de dato”
      ref=”declaración del elemento gobal”
      id=”identificador”
      minOccurs=”int (0,1, ...)”
      maxOccurs=”int(0, 1, unbounded)”
      default=”valor”
      fixed=”valor”
>
```

Veamos un poco el significado de algunos de los elementos de la sintaxis:

- Con el atributo **minOccurs** podemos indicar el número mínimo de veces que debe aparecer el elemento, por defecto es 1.
- Con el atributo **maxOccurs** podemos indicar el número máximo de veces, si indicamos unbounded, estamos diciendo que sin límites.

Veamos un pequeño ejemplo:

```
<curso>
      <alumno>Luis</alumno>
      <alumno>Ana</alumno>
      <alumno>Eva</alumno>
</curso>
```

Para describir el anterior documento XML utilizaríamos el siguiente esquema:

```
<xs:element name=”curso”>
    <xs:complexType>
       <xs:sequence>
         <xs:element name=”alumno” type=”xs:string” maxOccurs=”unbounded”>
       </xs:sequence>
    </xs:complexType>
</xs:element>
```

Nota: Más adelante veremos que el elemento curso es un elemento complejo.

Con los atributos **fixed** y **default**, podemos declarar valores por defecto, solo se utilizan para elementos simples, no para elementos que contengan a su vez otros elementos.

Ejemplo:

```
<xs:element name=”apellido” type=”xs:string” default=”Pérez”/>
```

De esta manera cuando un esquema de validación encontrase el elemento <apellido> en el documento XML insertará el valor por defecto, si no tuviese contenido. De esta manera sería validos los dos siguientes tipos de etiquetas:

```
<apellido>López</apellido>
<apellido/>
```

Si ponemos un elemento con un valor fijo, el validador del esquema buscará si el elemento contiene ese valor fijo, si está vacío se insertará ese valor.

1.6 Tipos simples de datos

Tenemos dos variantes

- Primitivos, los tipos de datos más básicos en XML, ya definidos en el lenguaje.
- Derivados, tipos de datos más complejos creados a partir de los anteriores.

Tipos de datos primitivos

Para poder usarlos los indicaremos en el atributo type de una etiqueta element o attribute.

Textos	
string	Representan textos con cualquier contenido, excepto los símbolos <, >, & y las comillas simples y dobles.

lógicos	
boolean	Solo puede contener los valores verdadero o falso, escritos como true o false, o como 1 o 0

números	
integer	Para números enteros
float	Permite utilizar números decimales en formato de coma flotante de precisión simple. El separador decimal es el punto. Admite el formato científico (1E+3), el INF, -INF NaN (Not A Number)
double	Números decimales en formato de coma flotante de precisión doble.
decimal	Representa números decimales en coma fija. Ocupan más internamente, pero se representan de forma exacta. No admite los valores INF, NaN ni el formato científico.
hexBinary	Representa números binarios codificados en notación hexadecimal
base64Binary	Representa números binarios (usando base 64)

fechas	
duration	Representa duraciones de tiempo en formato ISO 8601 PnYnMnDTnHnMnS, donde n son números Ejemplo:P1Y (1 año). La P es obligatoria y la T sirve para separar los valores fecha de los valores hora (P1Y2M3DT12H30M40.5S)
dateTime	Representa fechas según el formato **ISO 8601.** El formato es **yyyy-mm-ddThh:mm:ss**, por ejemplo *1998-07-12T16:30:00.000* (12 de julio de 1998 a las 16:30). La **T** es obligatoria para separar la fecha de la hora.
time	Representa horas en el formato *hh:mm:ss*
date	Representa fecha en formato *yyyy-mm-dd*
gYearMonth	Representa un mes y un año en formato *yyyy-mm*

gYear	Representa un año usando cuatro cifras.
gMonthDay	Representa un mes y un día en formato *--mm-dd*
gDay	Representa un día. Hay que hacerlo indicando tres guiones por delante (por ejemplo *---12*)
gMonth	Representa un mes en formato *–mm*, por ejemplo *--05*

especiales	
anyURI	Representa una dirección URI
QName	Nombre cualificado. Representa un nombre XML válido para identificar nombres incluyendo el prefijo de espacio de nombres. Por ejemplo *doc:cabecera*
Notation	Representa notaciones de estilo **NOTATION** XML 1.0 segunda edición. Solo se debe utilizar para crear tipos de datos derivados de éste
anyType	No restringe el contenido en modo alguno.

Tipos derivados

Son datos que se han definido a partir de los anteriores, pero forman parte de XMLSchema, es decir que en la práctica se usan igual que los anteriores (al igual que en los primitivos, cuando se usan en un esquema hay que añadir el prefijo del espacio de nombres del esquema, por ejemplo *xs:normalizedString*).

textos	
normalizedString	Se basa en el tipo **string**. Texto donde los caracteres de retorno de línea, tabulador y retorno de carro se convierten a espacios antes de procesar el esquema.
token	Se basa en el anterior. Textos en los que no hay más de un espacio en blanco seguido, ni tabuladores ni saltos de línea; en todos esos casos se convierte el texto a un único espacio
language	Texto que contiene el nombre de un lenguaje según lo definido en la especificación oficial **RFC 1766.** Son los posibles valores de los atributos **xml:lang** de XML 1.0 que coinciden con el formato normalizado de lenguajes habitual en las páginas web; por ejemplo el español se codifica con *es* (a veces con *es-ES*), el catalán *ca*, el gallego *gl*, el portugués *pt*, el euskera *eu*, el inglés *en*, alemán *de* y francés *fr*.
Name	Solo admite nombres compatibles con la forma de poner nombres de **XML.** Admite los dos puntos pero para manejar nombres con prefijo (nombres cualificados) el tipo idóneo es **QName.**
NCName	Nombres, basado en **Name**, pero sin admitir los dos puntos de los prefijos de espacios de nombres.

números	
nonPositiveInteger	Representa números enteros que no son positivos (es decir cero y negativos)
negativeInteger	Representa números enteros negativos (no vale el cero).
nonNegativeInteger	Representa números enteros que no son negativos (es decir cero y positivos).
positiveInteger	Representa números enteros positivos puros (no vale el cero).
long	Representa números enteros de alto rango (de *-9223372036854775898* a *9223372036854775897*).
unsignedLong	Representa números enteros de alto rango pero usando sólo los positivos y el cero.
int	Representa números enteros de medio rango (de –2147483648 a 2147483647).
unsignedInt	Representa números enteros de medio rango pero usando sólo los positivos y el cero.
short	Representa números enteros de bajo rango (de *–32768* a *32767*).
unsignedShort	Representa números enteros de medio rango pero usando sólo la parte positiva, de cero a *65535*.
byte	Representa números enteros pequeños (de *–128* a *127*).
unsignedByte	Representa números enteros pequeños positivos, de cero a 255.

Tipos de datos equivalentes a los usados en los DTD

Están basados en los tipos XML 1.0 y solo pueden utilizarse en atributos (para mantener la compatibilidad con DTD).

- **ID, IDREF e IDREFS**. Equivalente a los atributos del mismo tipo de XML 1.0 (tienen el mismo significado que en las DTD). Derivan de **NCName**
- **ENTITY, ENTITIES**. Equivalente a los atributos XML 1.0 del mismo nombre. Derivan de **NCName**,
- **NMTOKEN, NMTOKENS**. Permiten indicar textos compatibles con los nombres XML. Derivan de **NCName**.
- **NOTATION**. Es un tipo pensado para hacer anotaciones, su funcionamiento es peculiar y no está pensado para ser usado como tipo básico, sino como base para crear tipos personales.

1.7 Restricciones (Facets)

Permiten establecer reglas complejas que deben de cumplir los datos. En este caso dentro de la etiqueta simpleType se indica una etiqueta restriction, dentro de la cual se establecen las posibles restricciones. Sintaxis:

```
<xs:simpleType name=”nombre”>
      <xs:restriction base=”tipo”>
            … definición de la restricción …
      </xs:restriction>
</xs:simpleType>
```

El atributo obligatorio base sirve para indicar en qué tipo de dato nos basamos al definir la restricción.

Las posibles restricciones que se pueden establecer son:

Facets	Descripción
xs:length	Especifica una longitud fija
xs:minLength	Especifica una longitud mínima
xs:maxLength	Especifica una longitud máxima
xs:pattern	Especifica un patrón de caracteres admitidos
xs:enumeration	Especifica una lista de valores admitidos.
xs:whiteSpace	Especifica cómo se debe tratar a los posibles espacios en blanco, las tabulaciones, los saltos de línea y los retorno de carro que puedan aparecer.
xs:maxInclusive	Especifica que el valor debe ser menor o igual que el indicado.
xs:minInclusive	Especifica que el valor debe ser mayor o igual que el indicado.
xs:maxEclusive	Especifica que el valor debe ser menor que el indicado.
xs:minExclusive	Especifica que el valor debes ser mayor que el indicado
xs:totalDigits	Especifica el número máximo de dígitos que pueden tener un número.
xs:fractionDigits	Especifica el número máximo de decimales que puede tener un número.

Veamos algunos ejemplos:

Ejemplo1:

```
<xs:element name=”mes”>
      <xs:simpleType>
            <xs:restriction base=”xs:integer”>
                  <xs:minInclusive value=”1” />
                  <xs:maxInclusive value=”12” />
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

Veamos cada elemento

- Con xs:simpleType, definimos un tipo simple y especificamos sus restricciones.
- xs:restriction, sirve para definir restricciones de un xs:simpleType También sirve para definir restricciones de un xs:simpleContent o de un xs:complexContent. Los veremos más adelante.
- En el atributo base se indica el tipo de dato a partir del cual se define la restricción.

También se podría haber escrito de esta manera:

```
<xs:element name=”mes” type=”numeroMes” />
<xs:simpleType name=”numeroMes”>
      <xs:restriction base=”xs:integer”>
            <xs:minInclusive value=”1” />
            <xs:maxInclusive value=”12” />
```

```
        </xs:restriction>
    </xs:simpleType>
```

haciendo esto, el tipo `numeroMes` podría ser utilizado por otros elementos, ya que no está contenido en el elemento mes.

Ejemplo 2:

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

Otra manera de escribirlo sería:

```
    <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>
```

Así, `carType` podría ser usado por otros elementos, ya que no va dentro del elemento car.

Ejemplo 3:

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

Ejemplo 4

```
    <xs:element name="address">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:whiteSpace value="preserve"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
```

En `value` se pueden utilizar los siguientes valores:

- `preserve`, los espacios en blanco, tabulaciones y saltos de línea se deben mantener.
- `replace`, para sustituir todas las tabulaciones, saltos de linea y retornos de carro por espacios en blanco.
- `collapse`, para, después de reemplazar todas las tabulaciones, saltos de línea y retornos de carro

por espacios en blanco, eliminar todos los espacios en blanco únicos y sustituir varios espacios en blanco seguidos por un único espacio en blanco.

Ejemplo 5:

`xs:pattern` sirve para definir un patrón de cadenas de caracteres admitidos. Vemos un ejemplo donde se admite solo una única letra minúscula de la "a" a la "z".

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

Veamos en la siguiente tabla más posibilidades de los patrones:

Símbolo	equivalencia
.	Cualquier carácter
\w	Cualquier letra
\d	Cualquier dígito
\D	Cualquier carácter no dígito
\s	Cualquier carácter de espaciado
\S	Cualquier carácter de no espaciado
x{n}	Tiene que haber n veces lo que haya delante de los corchetes
x{n,}	Tiene que haber n o más veces lo que haya delante de los corchetes
x{n,m}	Tiene que haber entre n y m lo que haya delante de los corchetes
[Abc]	Tiene que haber algún carácter de los de dentro.
[A-Z]	Tiene que haber uno de los caracteres de la A a la Z
[^abc]	Negación de un grupo de caracteres
[F-J-[H]]	Sustracción de un carácter de un rango.
(a | b)	Alternativa de dos expresiones
x*	0 o más ocurrencias del elemento x
x+	1 o más ocurrencias del elemento x
x?	0 o 1 ocurrencias del elemento x

Veamos más ejemplos de facets o restricciones sacados de w3schools

Ejemplo 6 – Muestra una restricción para que la etiqueta initials solo muestre tres letras que siempre estén entre la A – Z o a -z

```
<xs:element name="initials">
      <xs:simpleType>
            <xs:restriction base="xs:string">
```

```
                    <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
              </xs:restriction>
        </xs:simpleType>
  </xs:element>
```

Ejemplo 7 Muestra un elemento password, que debe tener letras minúsculas, mayúsculas y dígitos entre 0 y 9 con una longitud de 8.

```
<xs:element name="password">
      <xs:simpleType>
            <xs:restriction base="xs:string">
                  <xs:pattern value="[a-zA-Z0-9]{8}"/>
            </xs:restriction>
      </xs:simpleType>
</xs:element>
```

1.7.1 Definir tipos simples personales

Otra forma de derivar un tipo de dato consiste en definir uniones o listas de tipos de datos base.

La sintaxis general es:

```
<xs:simpleType name=”nombre”>
      .. definición del tipo
</xs:simpleType>
```

- **Unión,** es un tipo de dato creado a partir de una colección de tipos de datos base. Se construyen con xs:union.

```
<xs:simpleType name=”gMonthC”>
      <xs:union memberTypes=”xs:gMonth xs:gMonthDay”/>
</xs:simpleType>
```

Cuando a cualquier elemento del esquema se le asigne el tipo *gMonthC*, se podrán especificar datos en formato *gMonth* y en formato *gMonthDay*.

```
<xs:simpleType name="SizeType">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="2"/>
        <xs:maxInclusive value="18"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="small"/>
        <xs:enumeration value="medium"/>
        <xs:enumeration value="large"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

- **Lista**, es un tipo de dato compuesto por listas de valores de un tipo de dato base. La lista de valores debe aparecer separada por espacios. Se construye con `xs:list`

```
<xs:simpleType name="precios">
  <xs:list>
      <xs:restriction base=”xs:decimal”>
            <xs:maxInclusive value=”10.1” />
      </xs:restriction>
  </xs:list>
</xs:simpleType>

<xs:element name="historiaprecios" type="precios">

<historiaprecios> 5.32 6.23 6.86 7.01 </historiaprecios>
```

1.8 Atributos en XML Schema

La sintaxis para definir un atributo es la siguiente:

```
<xs:attribute
      name=”nombre del elemento"
      type=”tipo de dato del atributo"
      use=”cómo-se-utiliza"
      default=”value”
      fixed="value"
/>
```

Donde:

- tipo de dato puede ser cualquier tipo de dato: integer, boolean, string, decimal, date, time, ...
- use puede toma tres valores: required, optional, prohibited.
- Si utilizamos default/fixed use debe tomar el valor de optional.

Veamos algunos ejemplos:

```
<xs:attribute name="idioma" type="xs:string" use=”optional” default="ES"/>
<xs:attribute name="idioma" type="xs:string" default="ES"/>
<xs:attribute name="edad" type="xs:integer" use="required"/>
```

1.9 Elementos complejos

Los elementos que tienen a su vez otros elementos y/o atributos. Existen 4 clases de elementos complejos:

- Elementos vacíos

```
<producto id=”1234” />
```

- Elementos que contienen solo otros elementos

```
<empleado>
  <nombre>Juan</nombre>
  <apellidos>Eva</apellidos>
</empleado>
```

- Elementos que contienen solo texto, pero que contienen atributos

```
<comida type=”postre”>Helado</comida>
```

- Elementos que contienen tanto texto como otros elementos

```
<descripcion>
  Ocurrió en <fecha lenguaje=”español”>03/03/2011</fecha>
</descripcion>
```

Los elementos que tienen a su vez otros elementos se controlan mediante la etiqueta `<complexType>`. La sintaxis para declarar la etiqueta `<complexType>` es:

```
<xs:complexType
    mixed=”true o false”
    name=”nombre del complexType>
```

Elemento vacío

En el ejemplo se ha definido el elemento `<producto id=”1234” />`, un elemento vacío, pero que si contiene un atributo. El esquema lo declararíamos así

```
<xs:element name="product">
    <xs:complexType>
        <xs:attribute name="id" type="xs:positiveInteger"/>
    </xs:complexType>
</xs:element>
```

Elemento que contiene otros elementos

Con nuestro ejemplo anterior el esquema quedaría así

```
<xs:element name="empleado">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellidos" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

O bien, puede dar al elemento complexType un nombre, y dejar que el elemento de "empleado" tiene un atributo de tipo que se refiere al nombre del complexType (si se utiliza este método, varios elementos se refieren al mismo tipo complejo):

```
<xs:element name="empleado" type="personatipo"/>

<xs:complexType name="personatipo">
    <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="apellidos" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Elemento que contiene solo texto, pero con un atributo

El siguiente ejemplo contiene solo texto, pero también un atributo país

```
<talla_zapato pais="francia">35</talla_zapato>
```

El esquema quedaría así

```
<xs:element name="talla_zapato">
      <xs:complexType>
            <xs:simpleContent>
                  <xs:extension base="xs:integer">
                        <xs:attribute name="pais" type="xs:string" />
                  </xs:extension>
            </xs:simpleContent>
      <xs:complexType>
</xs:element>
```

También podríamos dar al elemento complexType un nombre, y dejar que el elemento "talla_zapato" tenga un atributo de tipo que se refiere al nombre del complexType (si se utiliza este método, varios elementos se refieren al mismo tipo complejo):

```
<xs:element name="shoesize" type="shoetype"/>
<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

Elementos con contenido mixto

Pueden contener tanto texto, como atributos, como otros elementos Aquí se usa el atributo mixed con valor true.

Veamos un ejemplo:

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

El siguiente esquema declara el elemento de "letter":

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
```

Nota: Para habilitar los datos de caracteres a aparecer entre los elementos hijos de la "letter", el atributo mixto se debe establecer en "true". La <xs:sequence> significa que los elementos definidos (name, orderid y shipdate) deben aparecer en este orden dentro de un elemento "letter".

También podríamos dar al elemento complexType un nombre, y dejar que el elemento de "letra" tener un atributo de tipo que se refiere al nombre del complexType (si se utiliza este método, varios elementos se refieren al mismo tipo complejo):

```
<xs:element name="letter" type="lettertype"/>
<xs:complexType name="lettertype" mixed="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="orderid" type="xs:positiveInteger"/>
   <xs:element name="shipdate" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
```

1.10 Extensiones

Veamos un ejemplo donde emplear el atributo name:

El elemento <empleado> puede tener un atributo que se refiera al nombre del tipo complejo a usar:

```
<xs:element name="empleado" type="informacionpersonal"/>
<xs:complexType name="informacionpersonal">
  <xs:sequence>
   <xs:element name="nombre" type="xs:string"/>
   <xs:element name="apellido" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Si usamos lo anterior, diferentes elementos pueden referirse al mismo tipo complejo, de esta manera:

```
<xs:element name="empleado" type="informacionpersonal"/>
<xs:element name="estudiante" type="informacionpersonal"/>
<xs:complexType name="informacionpersonal">
 <xs:sequence>
   <xs:element name="nombre" type="xs:string"/>
   <xs:element name="apellido" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

xs:extension (complexContent)

Podemos incluso basarnos en algún elemento complejo existente y añadirles algunos elementos, como en el ejemplo siguiente:

```
<xs:element name="empleado" type="informacionpersonal"/>
<xs:complexType name="informacionpersonal">
 <xs:sequence>
```

```
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="informacionpersonalcompleta">
<xs:complexContent>
<xs:extension base="informacionpersonal">
 <xs:sequence>
   <xs:element name="direccion" type="xs:string"/>
   <xs:element name="ciudad" type="xs:string"/>
 </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

xs:extension(simpleContent)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="size">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small" />
    <xs:enumeration value="medium" />
    <xs:enumeration value="large" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="jeans">
  <xs:simpleContent>
   <xs:extension base="size">
     <xs:attribute name="sex">
        <xs:simpleType>
           <xs:restriction base="xs:string">
              <xs:enumeration value="male" />
              <xs:enumeration value="female" />
           </xs:restriction>
        </xs:simpleType>
     </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

</xs:schema>
```

1.11 Indicadores

Los indicadores permites establecer cómo se van a escribir o utilizar los elementos en un documento XML. Hay siete tipos de indicadores que se pueden clasificar en;

- indicadores de orden secuencia (sequence), todo (all) y elección (choice)
- indicadores de ocurrencia; maxOccurs y minOccurs
- indicadores de grupo: de elementos (group) y de atributos (attributeGroup)

Indicadores de orden

- `xs:all`: han de incluirse todos los elementos, sin importar el orden.
- `xs:sequence`: han de incluirse todos los elementos, respetando el orden.
- `xs:choice`: es necesario incluir uno de los elementos.

Veamos algunos ejemplos

Ejemplo 1:

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Con este ejemplo los elementos firstname y lastname podrían aparecer en cualquier orden.

Ejemplo 2:

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

En este ejemplo especificamos que solo puede aparecer uno de los dos elementos, employee o member.

Ejemplo 3

```
<xs:element name="person">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
     </xs:sequence>
```

```
    </xs:complexType>
</xs:element>
```

Los elementos firstname y lastname deben aparecer en el orden especificado.

Indicadores de ocurrencia

maxOccurs y minOccurs permiten establecer respectivamente, el número máximo y mínimo de veces que puede aparecer un determinado elemento. El valor por defecto para los dos es 1.

Ejemplo 1

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El ejemplo anterior indica que el elemento "child_name" se puede producir un mínimo de un tiempo (el valor por defecto para minOccurs es 1) y un máximo de diez veces en el elemento de "person".

Ejemplo 2

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El ejemplo anterior indica que el elemento "child_name" se puede producir un mínimo de cero veces y un máximo de diez veces en el elemento de "person".

Indicadores de grupo

xs:group sirve para agrupar un conjunto de declaraciones de elementos relacionados

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

<xs:group name="custGroup">
  <xs:sequence>
    <xs:element name="customer" type="xs:string"/>
    <xs:element name="orderdetails" type="xs:string"/>
    <xs:element name="billto" type="xs:string"/>
    <xs:element name="shipto" type="xs:string"/>
  </xs:sequence>
```

```
</xs:group>

<xs:element name="order" type="ordertype"/>

<xs:complexType name="ordertype">
  <xs:group ref="custGroup"/>
  <xs:attribute name="status" type="xs:string"/>
</xs:complexType>

</xs:schema>
```

xs:attributeGroup, sirve para definir un grupo de atributos

```
<xs:attributeGroup name="personattr">
  <xs:attribute name="attr1" type="string"/>
  <xs:attribute name="attr2" type="integer"/>
</xs:attributeGroup>

<xs:complexType name="person">
  <xs:attributeGroup ref="personattr"/>
</xs:complexType>
```

El ejemplo anterior define un grupo de atributos llamado "personattr" que se utiliza en un tipo complejo llamado "person".

BIBLIOGRAFÍA

- XML Juan Diego Gutiérrez
- Beginning XML David Hunter
- W3CSchools