

## Tema 5. XML Schema.

1. Introducción. ....	3
2. XML Schema. ....	3
2.1. Referencia a un XML Schema. ....	3
3. Estructura de un XML Schema. ....	4
3.1. El elemento <xs:schema>. ....	4
3.2. El elemento <xs:element>. ....	5
3.3. El elemento <xs:attribute>. ....	6
4. Tipos de datos. ....	6
4.1. Tipos de datos predefinidos. ....	6
5. Tipos de datos simples y complejos. ....	8
5.1. Definición de tipos simples por restricción. ....	8
5.1.1. Restricciones de longitud de un texto. ....	9
5.1.2. Cifras de un número. ....	9
5.1.3. Valores máximo y mínimo de un rango. ....	10
5.1.4. Restricciones sobre los espacios en blanco. ....	10
5.1.5. Enumeraciones. ....	10
5.1.6. Plantillas. ....	11
5.2. Definición de tipos simples por unión de contenidos. ....	12
5.3. Definición de tipos simples mediante listas. ....	12
6. Definición de tipos de datos complejos. ....	12
6.1. Distintas posibilidades de elementos complejos. ....	14
6.2. Elementos vacíos. ....	14
6.3. Elementos que sólo contienen texto y atributos. ....	14
6.4. Elementos sólo con elementos descendientes y atributos. ....	14
6.5. Elementos con contenido simple y elementos descendientes. ....	15
6.6. Extensiones de tipo complejo. ....	16
7. Grupos. ....	17
8. Documentación. ....	17
9. Modelos de diseño de esquemas. ....	18
9.1. Modelo de diseño plano. ....	18
9.2. Modelo con tipos de nombres reutilizables. ....	19
9.3. Modelo de diseño anidado. ....	20



# Tema 5. XML Schema.

## 1. Introducción.

Además de los DTD estudiados en el tema anterior, existen otras formas de validar documentos XML tales como los **XML Schema (XSchema)**.

Al igual que un DTD, un XSchema define los elementos que puede contener un documento XML, su organización, sus tipos y sus atributos. Sin embargo, presenta las siguientes ventajas:

- Usa **sintaxis XML**, por lo que no hay que aprender una sintaxis nueva para utilizarlo y además es analizable como cualquier documento XML.
- Permite especificar **tipos de datos**, por ejemplo, podemos indicar si un elemento o un atributo es un número entero, una fecha, una cadena de caracteres, etc.
- Permite utilizar **diferentes espacios de nombres (namespace)**, lo que implica que las reglas para validar ciertos elementos estarán definidas en cierto sitio, mientras que otras reglas podrán estar en otro.
- Es **extensible**, es decir, el conjunto de etiquetas que lo constituye se puede ampliar según las necesidades que surjan.

## 2. XML Schema.

Los XML Schema están basados en el lenguaje **XML Schema Definition (XSD)**, desarrollado por el World Wide Web Consortium (W3C), alcanzando el nivel de recomendación en mayo de 2001. Posteriormente, se creó una segunda edición revisada, disponible desde octubre de 2004.

Esta última recomendación (<http://www.w3.org/XML/Schema>), se expone en tres partes:

- **XML Schema Parte 0 Primer:** es una introducción no normativa al lenguaje, que proporciona una gran cantidad de ejemplos y explicaciones detalladas para una primera aproximación a XML Schema.
- **XML Schema Parte 1 Structures:** es una extensa descripción de los componentes del lenguaje.
- **XML Schema Parte 2 Datatypes:** complementa la parte 1 con la definición de los tipos de datos incorporados en XML Schema y sus restricciones.

Los documentos XML que se validan mediante un esquema se denominan **instancias del esquema**.

### 2.1. Referencia a un XML Schema.

Para que un documento XML siga las reglas definidas en un esquema, se usan atributos especiales en el elemento raíz del documento XML. Mediante estos atributos se definen dos espacios de nombres: el correspondiente al documento XML (que se suele usar sin prefijo, es decir, como espacio de nombres por defecto), y el espacio de nombres de XML Schema (que suele utilizar el prefijo **xs** o  **xsi**, aunque se pueden utilizar otros).

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

El atributo **schemaLocation** (acompañado del prefijo asociado al espacio de nombres de XMLSchema) indica la localización del documento XMLSchema que contiene la definición de las reglas a cumplir por el documento. Es un par formado por el espacio de nombres que será validado por el esquema y por la ruta al documento XMLSchema (con extensión **xsd**).

Por ejemplo:

```
<raiz xmlns="http://www.ejemplo.org"
  xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:schemaLocation="http://www.ejemplo.org esquema.xsd">
```

Se pueden indicar varios esquemas de validación, por lo que habría que especificar a qué espacio se aplica cada uno de ellos. A tal fin, el atributo **schemaLocation** permite contener una lista de espacios de nombres y localizaciones de esquemas, separados éstos por espacios en blanco.

A continuación, se muestra un ejemplo en el que se permite la validación de elementos y atributos situados en dos espacios de nombres distintos mediante dos esquemas diferentes:

```
<raiz xmlns:doc="http://www.ejemplo.org/doc"
      xmlns:img="http://www.ejemplo.org/img"
      xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
      xs:schemaLocation="http://www.ejemplo.org/doc esquemaDoc.xsd
                        http://www.ejemplo.org/img esquemaImg.xsd">
```

Por otra parte, con el atributo **noNamespaceSchemaLocation** se permite indicar un esquema para el documento sin que éste utilice espacio de nombres alguno. Esto, como veremos, no es aconsejable, pero se usa bastante para hacer pruebas; de hecho, lo usaremos en los ejemplos de los apuntes.

Por ejemplo:

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

### 3. Estructura de un XML Schema.

Un esquema es un documento XML que se guarda en un archivo con extensión **xsd** y, como tal, presenta la estructura habitual de cualquier documento de tipo XML con la particularidad de que el elemento raíz se llama **schema**. Por tanto, la estructura general de un XML Schema será:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
  ...
</xs:schema>
```

Donde **xs** es el espacio de nombres que identifica al espacio de nombres de XML Schema:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
```

Los componentes básicos de un esquema son:

- **xs:schema**, elemento raíz de todo esquema.
- **Elementos**, definidos con etiquetas **xs:element**, indican los elementos permitidos en los documentos que sigan el esquema.
- **Atributos**, con la etiqueta **xs:attribute**, permiten indicar los atributos de los elementos.

El ámbito de aplicación de cada uno de estos componentes dentro de un esquema queda determinado por el lugar en el que se encuentra su definición. De aquí que se distingan dos posibilidades a la hora de declarar componentes:

- De **ámbito global**. Se trata de componentes del esquema situados dentro de la etiqueta raíz **schema** pero fuera de cualquier otra. Estos componentes se pueden utilizar en cualquier parte del esquema.
- De **ámbito local**. Se trata de componentes definidos dentro de otros. En ese caso, se pueden utilizar sólo dentro del componente en el que están incluidos pero no en otras partes del documento. Por ejemplo, si dentro de la definición de un elemento colocamos la definición de un tipo de datos, este tipo de datos sólo se puede utilizar dentro del componente **xs:element** en el que se encuentra dicha definición.

El orden de los componentes que se encuentran en el mismo ámbito de un esquema no es significativo, es decir, las declaraciones se pueden hacer en cualquier orden.

#### 3.1. El elemento <xs:schema>.

Este componente es el que realiza la declaración del esquema, siendo el elemento raíz de todo el esquema. Puede tener algunos atributos. De hecho, generalmente aparece de una forma similar a ésta:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.empresa.com"
            xmlns="http://www.empresa.com"
            elementFormDefault="qualified">
  ...
</xs:schema>
```

donde:

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` declara el espacio de nombres que utilizan los esquemas (lugar en el que se definen los elementos y tipos de un esquema), lo que permite diferenciar las etiquetas XML del esquema, respecto a otras. Además, asigna a este espacio de nombres un prefijo, en este caso, **xs**. A veces también se usa el prefijo **xsd**.
- `targetNamespace="http://www.empresa.com"` es una referencia a un espacio de nombres para indicar que los elementos definidos en el esquema (elementos, atributos, etc. del documento XML) pertenecen a dicho espacio de nombres. Si se usan varios espacios de nombres, aparecerán separados con espacios.
- `xmlns=http://www.empresa.com`. Normalmente, un esquema XML se aplica a un espacio de nombres privado, correspondiente a la entidad a la que se quiere aplicar el esquema. Este espacio de nombres se puede declarar como espacio de nombres por defecto (es lo habitual) o usar un prefijo, incluso, se pueden indicar varios espacios de nombres.
- `elementFormDefault="qualified | unqualified"`. Permite indicar si todos los elementos usados en el documento XML que se declaren en este esquema deberán clasificarse dentro del espacio de nombres mediante un prefijo (qualified) o no (unqualified).
- `version`. Indica la versión del esquema.

Ojo. **xmlns** y **targetNamespace** no sirven para lo mismo; el primero declara espacios de nombres y el segundo sirve para indicar a qué espacio de nombres pertenecen los elementos definidos en el esquema. Aunque en la práctica ciertamente el contenido de ambos atributos es el mismo.

### 3.2. El elemento <xs:element>.

Permite la declaración de elementos y representa la aparición de un elemento en el documento XML. Los atributos principales son:

- **id**="identificador único para el elemento".
- **name**="nombre del elemento". Obligatorio si el elemento padre es <xs:schema>.
- **type**="tipo de datos del elemento".
- **ref**="referencia a otro elemento el cual realiza la descripción". No se usa si el elemento padre es <xs:schema>
- **form**="qualified|unqualified" Fuerza a que el elemento sea o no clasificado dentro del espacio de nombres mediante un prefijo.
- **minOccurs**="número mínimo de ocurrencias que puede haber del elemento". Valores: 0, 1, 2, ... unbounded
- **maxOccurs**="número máximo de ocurrencias que puede haber del elemento". Valores: 0, 1, 2, ... unbounded
- **default**="valor por defecto para el elemento cuando en el documento XML no haya recibido ningún valor".
- **fixed**="indica el único valor que puede tomar el elemento".

Algunos ejemplos de declaraciones de elementos simples son:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
  <xs:element name="articulo" type="xs:string" />
  <xs:element name="unidades" type="xs:nonNegativeInteger"/>
  <xs:element name="fecha" type="xs:date"/>
  ...
</xs:schema>
```

Aquí tenemos algunos elementos de XML que cumplen las restricciones anteriores:

```
<articulo>Tornillo</articulo>
<unidades>50</unidades>
<fecha>2012-02-14</fecha>
```

El atributo `ref` se usa para hacer referencia a otro elemento que puede llevar prefijo. Por ejemplo:

```
<xs:element ref="articulo" />
y, posteriormente, definir el elemento referenciado:
<xs:element name="articulo" type="xs:string" />
```

Los elementos simples pueden tener un valor por defecto o un valor prefijado. Un valor por defecto se asigna automáticamente al elemento cuando no especificamos ninguno. En este ejemplo, se asigna *tuerca*:

```
<xs:element name="articulo" type="xs:string" default="tuerca"/>
```

Un elemento con valor prefijado siempre tiene el mismo valor y no podemos asignarle otro distinto:

```
<xs:element name="articulo" type="xs:string" fixed="arandela"/>
```

### 3.3. El elemento <xs:attribute>.

Permite la declaración de atributos y representa la aparición de un atributo de un elemento en el documento XML. Los atributos principales son:

- **id**="identificador único para el atributo".
- **name**="nombre del atributo". Este atributo no puede aparecer simultáneamente con **ref**.
- **type**="tipo de dato del atributo".
- **ref**="permite hacer referencia a otro lugar en donde se realiza la descripción".
- **form**="qualified|unqualified" Fuerza a que el atributo sea o no clasificado dentro del espacio de nombres mediante un prefijo.
- **use**="indica si el atributo es opcional, obligatorio o prohibido (**optional**, **required**, **prohibited**)".
- **default**="valor por defecto para el atributo cuando en el documento XML no haya recibido ningún valor".
- **fixed**="indica el único valor que puede tomar el atributo". No puede aparecer simultáneamente con **default**.

Ejemplo: para la línea XML siguiente:

```
<articulo codigo="A10">clavo</articulo>
```

La declaración del atributo codigo podría ser:

```
<xs:attribute name="codigo" type="xs:string" use="required"/>
```

Ejemplo: atributo de nombre *divisa*, tipo de dato *texto* y con valor por defecto *Euros*.

```
<xs:attribute name="divisa" type="xs:string" default="Euros"/>
```

## 4. Tipos de datos.

En los ejemplos anteriores declaramos elementos y atributos de ciertos tipos de datos. Los datos se pueden clasificar de diferentes formas. Una de ellas es la siguiente:

- Datos **predefinidos**. Son los que vienen integrados en la propia especificación de XML Schema.
- Datos **construidos**. Son definidos por el usuario basándose en algún tipo predefinido o en otro construido previamente.

### 4.1. Tipos de datos predefinidos.

Existen muchos tipos de datos predefinidos. Se organizan de forma jerárquica, al estilo de la jerarquía de clases en la POO. Cada tipo será igual que su tipo padre más alguna característica propia. A medida que descendemos por la jerarquía, los tipos son más precisos o concretos.

Existen dos tipos especiales que no se suelen usar porque están en la parte superior de la jerarquía y, por lo tanto, son demasiado genéricos:

- **xs:anyType**. Es el tipo raíz del que derivan todos los demás tipos existentes. Para los elementos, es el tipo por defecto si no se especifica ninguno. Como, en realidad, es un tipo complejo, no podría usarse para los atributos.
- **xs:anySimpleType**. Representa a cualquier tipo simple en general y es el tipo por defecto para los atributos.

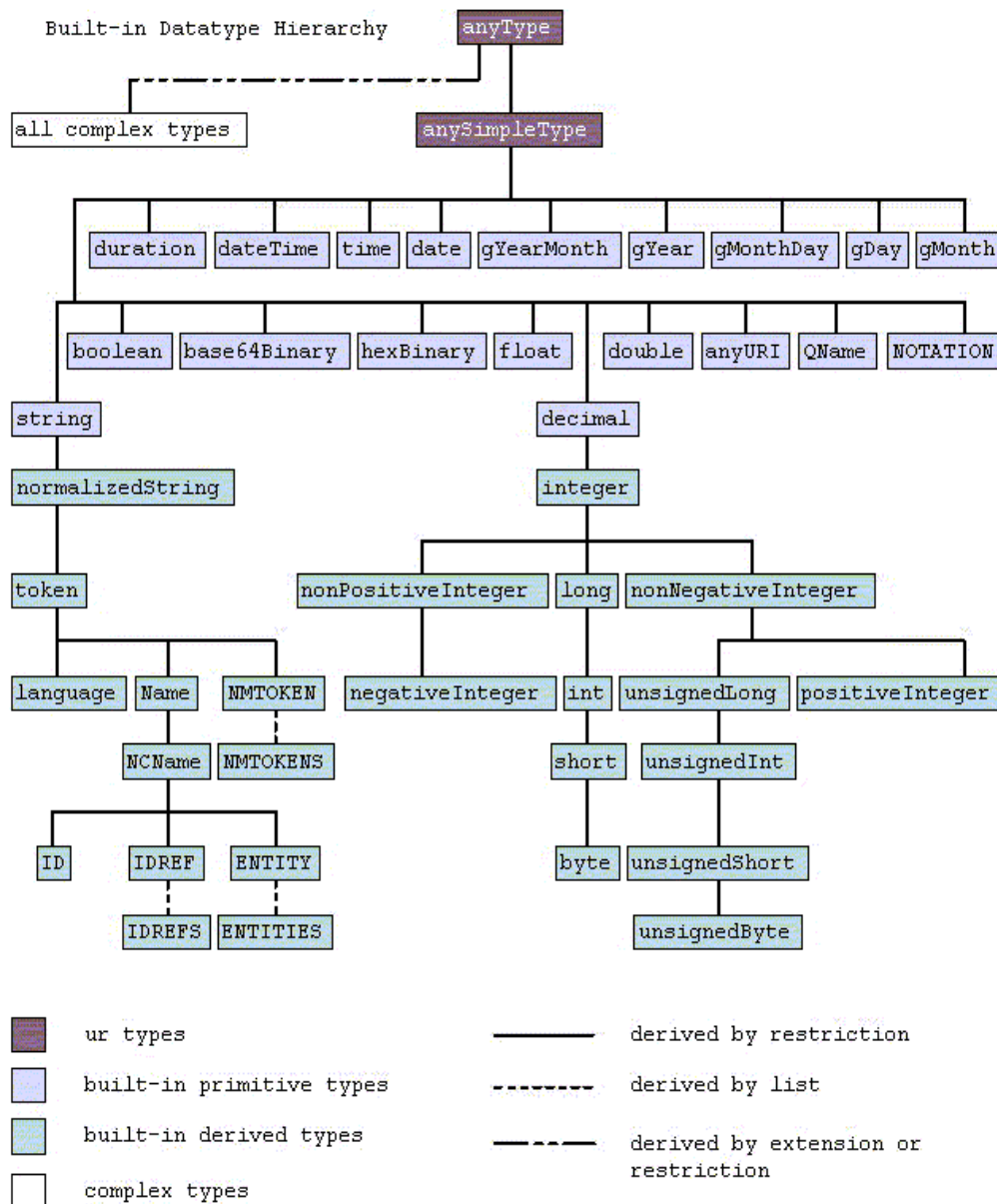
Los tipos predefinidos también se dividen en:

- **Primitivos**, son los que derivan directamente de **xs:anySimpleType**.
- **Derivados** o **no primitivos**, son los que derivan de algunos de los primitivos.

Se agrupan según su tipo de contenido en *numéricos*, *fecha*, *texto*, *binarios* y *booleanos*. La figura de la página siguiente (<http://www.w3.org/TR/xmlschema-2/>) muestra la jerarquía de datos predefinidos (*primitivos*, *derivados* y *especiales*).

Los tipos predefinidos que se usan más habitualmente son:

- **xs:string**, para textos con cualquier contenido, excepto *<*, *>*, *&* y comillas (simples y dobles), para los que se ha de utilizar la entidad correspondiente.



- **xs:normalizedString**, texto sin retornos de carro, saltos de línea o tabuladores.
- **xs:token**, texto sin retornos de carro, saltos de línea, tabuladores, espacios al principio o al final, o dos o más espacios seguidos a lo largo de él.
- **xs:boolean**, sólo puede contener los valores verdadero y falso, escritos como *true* y *false* o *1* (verdadero) y *0* (falso).
- **xs:integer**, para números enteros.
- **xs:decimal**, para números reales en coma fija.
- **xs:float**, para números reales en coma flotante y simple precisión. Se usa punto decimal.
- **xs:double**, para números reales en coma flotante y doble precisión. También utiliza el punto como separador decimal.
- **xs:date**, para fechas en formato *aaaa-mm-dd*.
- **xs:time**, para horas en formato *hh:mm:ss*.

Algunos tipos de **datos derivados de los primitivos** definidos por XSD son:

- **xs:negativeInteger**, enteros estrictamente menores que cero. ( $z < 0$ )
- **xs:positiveInteger**, enteros estrictamente mayores que cero. ( $z > 0$ )
- **xs:nonNegativeInteger**, enteros mayores o iguales que cero. ( $z \geq 0$ )
- **xs:nonPositiveInteger**, enteros menores o iguales que cero. ( $z \leq 0$ )

- **xs:language**, texto que contiene el nombre de un idioma (en, en-EN, en-US, es, es-ES, etc.)
- Los equivalentes a los siguientes atributos DTD, para mantener la compatibilidad con ellos:
  - **xs:ID**, **xs:IDREF**, **xs:IDREFS**.
  - **xs:ENTITY**, **xs:ENTITIES**.
  - **xs:NMTOKEN**, **xs:NMTOKENS**.
  - **xs:NOTATION**.

Como veremos más adelante, el usuario puede construir sus **propios tipos de datos** a partir de los predefinidos mediante uniones de tipos de datos, restricciones a los tipos de datos y listas de datos. Para ello se usará, como base, el elemento **xs:simpleType**.

Los tipos de datos predefinidos pueden ser asignados a elementos y a atributos, y pertenecen al espacio de nombres <http://www.w3.org/2001/XMLSchema>.

En la web <http://www.w3.org/TR/xmlschema-0> se puede encontrar la relación completa de tipos de datos definidos en XSD.

## 5. Tipos de datos simples y complejos.

Los datos también se pueden clasificar en simples y complejos.

Todos los tipos de datos predefinidos vistos anteriormente son simples y, en general, representan valores atómicos. Se pueden asignar tanto a elementos como a atributos. También se pueden construir datos simples partiendo de un tipo predefinido añadiéndole restricciones.

Los tipos de datos complejos sólo se pueden asignar a elementos que tengan descendientes y/o atributos. Como luego veremos, los tipos complejos pueden estar vacíos, contener sólo texto y no a otros elementos, contener sólo a otros elementos o contener tanto texto como otros elementos. Además, en todos estos casos pueden tener o no atributos.

### 5.1. Definición de tipos simples por restricción.

La definición de tipos simples se realiza con el componente `<xs:simpleType>`, y una de las formas de definirlos es mediante una restricción sobre un tipo base. Con el elemento `<xs:restriction>` podemos limitar el rango de valores.

La sintaxis es la siguiente:

```
<xs:simpleType name="nombreDelTipo">
  <xs:restriction base="tipoBase">
    Definición de la(s) restricción(es)
  </xs:restriction>
</xs:simpleType>
```

El atributo `base` indica el tipo de datos sobre el que se establecen las posibles restricciones.

Ejemplo:

```
<xs:simpleType name="T_Longitud">
  <xs:restriction base="xs:float" />
</xs:simpleType>
```

A partir de este momento el tipo de dato `T_Longitud` se podrá usar en todo el esquema y como, en realidad, no se ha definido con ninguna restricción, será equivalente al tipo `xs:float`

```
<xs:element name="distancia" type="T_Longitud"/>
```

Al elemento `distancia` se le ha asignado el tipo `T_Longitud`

Ejemplo:

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

<xs:element name="password" type="T_Clave"/>
```

Se ha definido el tipo `T_Clave` restringiendo el rango de valores permitidos al tipo base `xs:string`.



Cuando definimos un tipo de datos, éste podrá ser referenciado repetidas veces. Otra solución pero menos ventajosa podría ser la siguiente:

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

En este caso, el elemento password queda perfectamente descrito, pero no hemos definido un tipo que, posteriormente, pudiéramos usarlo para definir otros elementos o atributos.

A continuación, vemos los diferentes tipos de restricciones (facetas) que se pueden imponer. En función del tipo base se podrán aplicar una u otras facetas.

FACETA	USO	DATOS DONDE SE USA
xs:minInclusive	Indica el límite inferior del rango de valores, incluido el propio valor	Numéricos y fecha/hora
xs:maxInclusive	Indica el límite superior del rango de valores, incluido el propio valor	Numéricos y fecha/hora
xs:minExclusive	Indica el límite inferior del rango de valores, excluido el propio valor	Numéricos y fecha/hora
xs:maxExclusive	Indica el límite superior del rango de valores, excluido el propio valor	Numéricos y fecha/hora
xs:enumeration	Indica una lista de valores permitidos	Todos
xs:pattern	Indica una expresión regular que deben cumplir los valores	Texto
xs:whiteSpace	Indica cómo tratar los espacios en blanco. Los posibles valores son <i>preserve</i> , <i>replace</i> y <i>collapse</i>	Texto
xs:length	Indica el número de caracteres o elementos de una lista permitidos	Texto
xs:minLength	Indica el número mínimo de caracteres o elementos de una lista permitidos	Texto
xs:maxLength	Indica el número máximo de caracteres o elementos de una lista permitidos	Texto
xs:fractionDigits	Indica el número máximo de cifras de la parte decimal permitidas	Numéricos con parte decimal
xs:totalDigits	Indica el número máximo de dígitos totales permitidos	Numéricos

#### 5.1.1. Restricciones de longitud de un texto.

- **xs:length**, indica un número determinado de caracteres para el texto.
- **xs:maxLength**, indica el número máximo de caracteres para el texto.
- **xs:minLength**, indica el número mínimo de caracteres para el texto.

En el siguiente ejemplo se define el tipo T\_nombre como un texto con un número de caracteres comprendido entre 2 y 20.

```
<xs:simpleType name="T_nombre">
  <xs:restriction base="xs:string">
    <xs:minLength value="2"/>
    <xs:maxLength value="20"/>
  </xs:restriction>
</xs:simpleType>
```

#### 5.1.2. Cifras de un número.

- **xs:totalDigits**, indica el número máximo de cifras de un número.
- **xs:fractionDigits**, indica el número máximo de cifras decimales de un número.

En el ejemplo siguiente se define el tipo T\_nota como un número de, como máximo, cuatro cifras de las que, como mucho, dos son decimales.

```
<xs:simpleType name="T_nota">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="4"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

### 5.1.3. Valores máximo y mínimo de un rango.

- **xs:minExclusive**, abarca el rango de valores estrictamente mayores que el indicado.
- **xs:maxExclusive**, abarca el rango de valores estrictamente menores que el indicado.
- **xs:minInclusive**, abarca el rango de valores mayores o iguales al indicado.
- **xs:maxInclusive**, abarca el rango de valores menores o iguales al indicado.

Para que el elemento edad sea un entero comprendido entre 18 y 65, escribiremos:

```
<xs:element name="edad">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="65"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

### 5.1.4. Restricciones sobre los espacios en blanco.

Se lleva a cabo mediante **xs:whiteSpace** con tres posibles valores para el atributo value:

- **preserve**, para que se conserven espacios en blanco, tabuladores, saltos de línea y retornos de carro.
- **replace**, para que cada doble espacio, tabulador, salto de línea o retorno de carro se sustituya por un espacio.
- **collapse**, es similar al anterior pero, además, elimina espacios a izquierda y derecha.

Con esta declaración se validaría el código que aparece a continuación:

```
<xs:element name="texto">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
      <xs:length value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<text>      </text>
```

### 5.1.5. Enumeraciones.

Permiten indicar, uno a uno, todos los posibles valores que forman parte de un tipo.

En este ejemplo, se define el tipo T\_EstadoCivil con todos sus posibles valores.

```
<xs:simpleType name="T_EstadoCivil">
  <xs:restriction base="xs:string">
    <xs:enumeration value="soltero"/>
    <xs:enumeration value="casado"/>
    <xs:enumeration value="separado"/>
    <xs:enumeration value="divorciado"/>
    <xs:enumeration value="viudo"/>
  </xs:restriction>
</xs:simpleType>
```

### 5.1.6. Plantillas.

Permiten definir la secuencia exacta de caracteres que se admiten en un tipo de dato. Para ello, se utiliza la etiqueta **<xs:pattern value="expresion"/>**, donde **expresion** es una expresión regular con estas normas:

Patrón	Significado	Ejemplo
.	Cualquier carácter.	;
\w	Cualquier letra	K
\d	Un dígito	5
\D	Cualquier carácter no dígito	@
\s	Cualquier carácter de espacio (tabulador, espacio, etc.)	
\S	Cualquier carácter de no espacio	H
(a b)	Alternativa entre 2 expresiones	b
texto	Sólo se admite dicho "texto".	texto
[xyz]	Permite elegir uno entre los caracteres x, y, z.	y
[^xyz]	Prohíbe usar los caracteres x, y, z.	a
[a-z]	Permite cualquier carácter comprendido entre a y z.	h
[A-E-[C]]	Substracción de un carácter de un rango	B
+	Sucesión de 1 o más ocurrencias.	
?	Sucesión de 0 ó 1 ocurrencias.	
*	Sucesión de 0 o más ocurrencias.	
{n}	Acepta que el carácter precedente aparezca exactamente n veces.	
{n,}	Acepta que el carácter precedente aparezca, al menos, n veces.	
{n,m}	Acepta que el carácter precedente aparezca entre n y m veces.	
^	Inicio de línea.	
\$	Final de línea.	
\carácter	Permite escribir caracteres reservados, como ^, \$, +, *, -, (, ), ?,	

A continuación, se define el tipo de dato T\_DNI:

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

También podemos definir el tipo T\_sexo con dos posibles valores, *hombre* o *mujer*, de la forma:

```
<xs:simpleType name="T_sexo">
  <xs:restriction base="xs:string">
    <xs:pattern value="hombre|mujer"/>
  </xs:restriction>
</xs:simpleType>
```

Una definición de tipo para los teléfonos en formato de 9 dígitos separados por puntos cada tres, como por ejemplo 958.225.431 sería

```
<xs:simpleType name="T_Telefono">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}\.\d{3}\.\d{3}"/>
  </xs:restriction>
```

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

## 5.2. Definición de tipos simples por unión de contenidos

Se puede definir un nuevo tipo de dato a partir de una colección de datos simples previamente definidos. Para ellos se usa el componente **xs:union** que indica que un valor es válido si, al menos, es válido para al menos uno de los tipos de datos que forman la unión.

Por ejemplo, podemos crear el tipo de dato T\_enteroNoCero de la siguiente forma:

```
<xs:simpleType name="T_enteroNoCero">
  <xs:union memberTypes="xs:NegativeInteger xs:PositiveInteger"/>
</xs:simpleType>
```

## 5.3. Definición de tipos simples mediante listas.

El componente **xs:list** permite definir tipos cuyos valores no son atómicos o indivisibles, sino múltiples formados por una secuencia finita de valores de un tipo de datos base. Cada valor de la lista debe aparecer separado por espacios.

En este ejemplo, estamos creando el tipo de dato listaFechas como una lista de ítems de tipo xs:date.

```
<xs:simpleType name="listaFechas">
  <xs:list itemType="xs:date"/>
</xs:simpleType>

<fechas>2020-01-25 2020-02-04 2019-07-14</fechas>
```

En el siguiente ejemplo, maximas sería de tipo lista, donde los valores de la lista serían números reales, y el tamaño de la lista se fija a 8 valores. En el ejemplo, el elemento maximas contendrá las temperaturas máximas de un día de cada una de las provincias de Andalucía.

```
<xs:element name="maximas" type="listaOchoMaximas">
  <xs:simpleType name="listaMaximas">
    <xs:list itemType="xs:float"/>
  </xs:simpleType>
  <xs:simpleType name="listaOchoMaximas">
    <xs:restriction base="listaMaximas">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
```

Por tanto, en el documento XML correspondiente podría aparecer un elemento semejante al siguiente en la que los ocho valores están separados por espacios en blanco:

```
<maximas>15.5 22.1 18.0 21.4 25.6 20.2 22.6 23.9</maximas>
```

## 6. Definición de tipos de datos complejos.

Estos tipos de datos usan el componente **complexType** y sólo pueden ser asignados a elementos. Los elementos complejos son aquellos que contienen a otros **elementos descendientes y/o atributos**.

En general, se definen con esta sintaxis:

```
<xs:complexType name="nombre" mixed="true|false">
  Definición del elemento ...
</xs:complexType>
```

Donde name es el nombre del tipo que será obligatorio si el componente es hijo de xs:schema, de lo contrario no está permitido. El atributo mixed indica si se intercala texto con los elementos descendientes. El valor por defecto es false.

El contenido de un elemento XML, es decir, lo que hay entre las etiquetas de apertura y cierre puede ser:

- **Contenido simple** (xs:simpleContent). Sólo contiene texto, sin elementos descendientes.
- **Contenido complejo** (xs:complexContent). Contiene elementos descendientes y puede o no contener texto.

Cuando hablamos de elementos descendientes, aparece el concepto de **modelo de contenido**. El modelo de contenido indica la relación o distribución que tienen los elementos descendientes. Hay tres posibilidades las cuales se declaran con un componente distinto.

#### ❖ Secuencia

Los elementos aparecen unos detrás de otros en un orden fijo. Se declaran con el componente **xs:sequence**. Dicha secuencia puede aparecer un número configurable de veces estableciéndose con los atributos minOccurs y maxOccurs cuyo valor por defecto, en ambos casos, es uno.

Ejemplo. El nombre de una persona <nombre> con los elementos descendientes <npila>, <ape1> y <ape2>.

```
<xs:complexType name="T_Nombre">
  <xs:sequence>
    <xs:element name="npila" type="xs:string"/>
    <xs:element name="ape1" type="xs:string"/>
    <xs:element name="ape2" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="nombre" type="T_Nombre"/>
```

#### ❖ Alternativa

Se establece con el componente **xs:choice** y permite que se pueda escoger uno sólo entre varios elementos. La alternativa puede aparecer un número configurable de veces, estableciéndose con los atributos minOccurs y maxOccurs cuyo valor por defecto, en ambos casos, es uno.

En este ejemplo una empresa puede ser cliente o proveedor.

```
<xs:element name="empresa">
  <xs:complexType>
    <xs:choice>
      <xs:element name="cliente" type="xs:string"/>
      <xs:element name="proveedor" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

#### ❖ Todos

Indica que los elementos hijos pueden aparecer en cualquier orden pero, tan sólo, una vez. Se usa el componente **xs:all**.

En el ejemplo, el nombre y código deben aparecer los dos y en el orden que se quiera.

```
<xs:element name="identificador">
  <xs:complexType>
    <xs:all>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="codigo" type="xs:positiveInteger"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Para indicar que estos elementos puedan no aparecer, es necesario utilizar el atributo minOccurs="0" en la etiqueta <xs:all> de la siguiente forma:

```
<xs:all minOccurs="0">
```

En general, para los elementos descendientes de otros, cuando se desea establecer el número de veces que aparecerán, se hace uso de los indicadores minOccurs y maxOccurs. Por defecto, el valor de ambos es 1 pero podemos cambiarlo a cualquier número entero mayor o igual que cero. Además, para expresar un valor ilimitado se usa la palabra unbounded.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="nombrehijo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## 6.1. Distintas posibilidades de elementos complejos.

Las diferentes posibilidades de contenido para los elementos complejos son:

- Estar vacíos, es decir, no contener texto ni otros elementos. (*Empty*)
- Contener sólo texto y no otros elementos (*Simple Content*), los que, en una DTD serían #PCDATA.
- Contener sólo otros elementos descendientes. (*Complex Content*).
- Contener, tanto texto, como otros elementos descendientes. (*Mixed Content*)

Además, en todos los casos anteriores cabe la posibilidad de que los atributos también pueden estar presentes.

## 6.2. Elementos vacíos.

Se declara con un tipo de datos complejo (**xs:complexType**) sin contenido y que no contiene elementos descendientes.

La siguiente línea de código XML puede validarse con el código XSD que aparece a continuación:

```
<articulo codigo="14" fabricante="Acme"/>

<xs:element name="articulo">
  <xs:complexType>
    <xs:attribute name="codigo" type="xs:positiveInteger"/>
    <xs:attribute name="fabricante" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

O bien, de esta otra forma:

```
<xs:element name="articulo" type="T_Articulo"/>

<xs:complexType name="T_Articulo">
  <xs:attribute name="codigo" type="xs:positiveInteger"/>
  <xs:attribute name="fabricante" type="xs:string"/>
</xs:complexType>
```

Si no se desearan atributos para el elemento, entonces simplemente no existiría el componente `<xs:attribute>`, el resto se mantendría:

```
<xs:complexType name="T_Articulo" />
```

## 6.3. Elementos que sólo contienen texto y atributos.

Se declara con un tipo de datos complejo (**xs:complexType**) con contenido simple (sólo texto) mediante el componente **xs:simpleContent**. Este componente es el que indica que el elemento complejo no deba tener elementos descendientes.

Para indicar que el elemento va a contener atributos se usa el componente **xs:extension**. Su misión es extender el tipo de datos indicado como base para que el contenido incluya declaración de atributos.

Por ejemplo:

```
<xs:complexType name="T_Articulo">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="codigo" type="xs:positiveInteger"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<articulo codigo="14">Tornillo</articulo>
```

## 6.4. Elementos sólo con elementos descendientes y atributos.

Estos contenidos se refieren a los elementos que contienen otros elementos (pero nunca texto libre) y que puedan contener atributos. Se declara con un tipo de datos complejo (**xs:complexType**) con contenido complejo que contenga algún elemento descendiente.

Recordemos que los elementos descendientes podían aparecer según uno de los tres modelos de contenido: secuencias, alternativa y todos.

- Ejemplo:

```
<xs:complexType name="T_Factura">
  <xs:sequence>
    <xs:element name="numero" type="xs:integer"/>
    <xs:element name="cliente" type="xs:string"/>
    <xs:element name="importe" type="xs:float"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="facturacion" type="T_Factura"/>
```

Si además de elementos descendientes se definen atributos, éstos se indican siempre al final, justo antes de la etiqueta de cierre `</xs:complexType>`.

- Ejemplo:

```
<xs:complexType name="T_Factura">
  <xs:sequence>
    <xs:element name="numero" type="xs:integer" />
    <xs:element name="cliente" type="xs:string" />
    <xs:element name="importe" type="xs:float" />
  </xs:sequence>
  <xs:attribute name="fecha" type="xs:date" />
</xs:complexType>

<xs:element name="facturacion" type="T_Factura" />
```

El siguiente fragmento XML cumpliría con la declaración del esquema anterior

```
<facturacion fecha="2012-03-1">
  <numero>4523</numero>
  <cliente>Obdulio</cliente>
  <importe>67.89</importe>
</facturacion>
```

- Ejemplo:

```
<xs:element name="empresa">
  <xs:complexType>
    <xs:choice>
      <xs:element name="cliente" type="xs:string"/>
      <xs:element name="proveedor" type="xs:string"/>
    </xs:choice>
    <xs:attribute name="codigo" type="xs:integer" use="required" />
  </xs:complexType>
</xs:element>
```

El siguiente fragmento XML cumpliría con la declaración del esquema anterior

```
<empresa codigo="45">
  <proveedor>ACME</proveedor>
</empresa>
```

## 6.5. Elementos con contenido simple y elementos descendientes.

Se declara con un tipo de datos complejo (**xs:complexType**) con contenido complejo que contenga algún elemento descendiente y contenido simple.

Para indicar que un elemento de tipo complejo es mixto, es decir, que puede contener texto y otros elementos descendientes se usa el atributo `mixed` en el componente `xs:complexType` de la forma:

```
<xs:complexType mixed="true">
```

Este código XML podría validarse con el siguiente código XSD:

```
<carta>
  Estimado señor o señora <nombre>Luis de la Casa</nombre>:
  Su pedido <numpedido>1503</numpedido>
  le será enviado el <fechaenvio>2012-05-10</fechaenvio>
</carta>

<xs:element name="carta">
  <xs:complexType mixed="true">
```

```

    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="numpedido" type="xs:positiveInteger" />
      <xs:element name="fechaenvio" type="xs:date" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

En caso de que el elemento también tuviera atributos, se indicarían al final, justo antes de la etiqueta de cierre `</xs:complexType>`.

```

<carta referencia="145" autor="MariPili">
  Estimado señor o señora <nombre>Luis de la Casa</nombre>:
  Su pedido <numpedido>1503</numpedido>
  le será enviado el <fechaenvio>2012-05-10</fechaenvio>
</carta>

<xs:element name="carta">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="numpedido" type="xs:positiveInteger" />
      <xs:element name="fechaenvio" type="xs:date" />
    </xs:sequence>
    <xs:attribute name="referencia" type="xs:integer" use="required" />
    <xs:attribute name="autor" type="xs:string" />
  </xs:complexType>
</xs:element>

```

## 6.6. Extensiones de tipo complejo.

Se pueden añadir nuevos elementos y atributos a tipos complejos ya existentes. Estos elementos añadidos aparecerán al final de los elementos del tipo base.

- Ejemplo. Supongamos que disponemos de un tipo complejo `T_Nombre` definido de la forma:

```

<xs:complexType name="T_Nombre">
  <xs:sequence>
    <xs:element name="npila" type="xs:string"/>
    <xs:element name="ape1" type="xs:string"/>
    <xs:element name="ape2" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

Ahora queremos declarar un tipo complejo denominado `T_Alumno` que almacene, además del nombre y sus dos apellidos, tres elementos para almacenar domicilio, localidad y provincia; y un atributo numérico entero para el número de matrícula del alumno.

Por lo tanto, se trata de utilizar un nuevo tipo complejo, cuyo contenido es de tipo complejo (`xs:complexContent`) al que se le añade una extensión (`xs:extension`) a cuyo atributo base se le asigna el tipo de datos que sirve de base (`T_Nombre`)

```

<xs:element name="alumno">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="T_Nombre">
        <xs:sequence>
          <xs:element name="domicilio" type="xs:string"/>
          <xs:element name="localidad" type="xs:string"/>
          <xs:element name="provincia" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="matricula" type="xs:positiveInteger" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

```

Así, por ejemplo, la siguiente descripción de elementos sería válida según la descripción anterior:

```

<alumno matricula="7012">
  <npila>Juan</npila>
  <ape1>Ruiz</ape1>

```



```

    <ape2>Morales</ape2>
    <domicilio>Pablo Picasso,5</domicilio>
    <localidad>La Palma del Condado</localidad>
    <provincia>Huelva</provincia>
  </alumno>

```

## 7. Grupos.

A menudo ocurre que un mismo grupo de elementos aparece repetidas veces a lo largo del código de un esquema. Con el fin de simplificar su escritura, podemos declararlo una sola vez, y referenciarlo posteriormente desde otros elementos complejos tantas veces como sea necesario.

Los grupos no son tipos de datos, sino contenedores que incluyen un conjunto de elementos o atributos que pueden ser usados en la definición de tipos complejos. Para ello, se utilizan los componentes `xs:group` y `xs:attributeGroup`.

Cuando un elemento desea incorporar un grupo, utiliza el componente `xs:group` y, con el atributo `ref`, se indica el nombre del grupo definido anteriormente. He aquí un ejemplo de uso:

```

<xs:group name="infoPedido">
  <xs:sequence>
    <xs:element name="cliente" type="xs:string" />
    <xs:element name="detallepedido" type="xs:string" />
    <xs:element name="facturado_a" type="xs:string" />
    <xs:element name="enviado_a" type="xs:string" />
  </xs:sequence>
</xs:group>

<xs:element name="pedido">
  <xs:complexType>
    <xs:group ref="infoPedido" />
    <xs:attribute name="estado" type="xs:string" />
  </xs:complexType>
</xs:element>

```

Al igual que los elementos, los atributos también se pueden agrupar para ser referenciados en bloque desde uno o varios elementos complejos.

En el siguiente ejemplo se define el grupo de atributos `codigoIncidencia`, que es usado a continuación en un elemento complejo:

```

<xs:attributeGroup name="codigoIncidencia">
  <xs:attribute name="anno" type="xs:positiveInteger" />
  <xs:attribute name="numero" type="xs:integer" />
</xs:attributeGroup>

<xs:complexType name="T_incidencia">
  <xs:sequence>
    <xs:element name="informa" type="xs:string" />
    <xs:element name="centro" type="xs:string" />
    <xs:element name="fecha" type="xs:date" />
    <xs:element name="estado" type="xs:string" />
  </xs:sequence>
  <xs:attributeGroup ref="codigoIncidencia" />
</xs:complexType>

```

## 8. Documentación.

A un esquema se le pueden añadir componentes de tipo comentario que permitan, posteriormente, con alguna aplicación a medida, generar documentación obtenida del propio esquema.

Estos comentarios se indican con el componente `xs:annotation` y sus subelementos `xs:appinfo` y/o `xs:documentation`.

El componente `xs:annotation` puede incluirse dentro del elemento `xs:schema` o del resto de los componentes de un esquema, pero siempre aparecerá en primer lugar.

El componente `xs:appinfo` permite indicar información que será utilizada por la aplicación; y `xs:documentation` asocia comentarios de texto sobre el esquema. Ambos componentes pueden llevar el atributo `source` para indicar una URI donde localizar información sobre la aplicación.

A continuación se muestra un ejemplo de documentación de un elemento:

```
<xs:element name="producto" type="T_Producto">
  <xs:annotation>
    <xs:documentation xml:lang="es-ES">
      Este producto es de fabricación propia.
    </xs:documentation>
  </xs:annotation>
  <xs:appinfo>
    <app:dbmapping>
      <app:tb>PRODUCT_MASTER</app:tb>
    </app:dbmapping>
  </xs:appinfo>
</xs:annotation>
Resto de declaración del elemento. . .
</xs:element>
```

## 9. Modelos de diseño de esquemas.

A la hora de diseñar esquemas podemos elegir varios métodos de descripción. Todos son válidos, aunque dependiendo de la complejidad de los esquemas y de su tamaño, algunos de ellos podrán ser más eficientes o cómodos para el usuario.

Para construir diferentes esquemas usando diferentes modelos, partiremos del siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<almacen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <articulo cod="23">
    <descripcion>ali-kates</descripcion>
    <proveedor>
      <codigo-proveedor>PR-342</codigo-proveedor>
      <nombre-proveedor>Alfonso</nombre-proveedor>
    </proveedor>
    <stock fecha="2012-12-21">345</stock>
  </articulo>
  <articulo cod="78"> . . . </articulo>
  . . .
</almacen>
```

### 9.1. Modelo de diseño plano.

Se declaran los elementos y los atributos, y posteriormente se indican referencias a sus definiciones que se realizan en otro lugar del documento. Ejemplo:

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

  <xs:element name="almacen">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="articulo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="articulo">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="descripcion"/>
        <xs:element ref="proveedor"/>
        <xs:element ref="stock"/>
      </xs:sequence>
      <xs:attribute ref="cod" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="proveedor">
    <xs:complexType>
      <xs:sequence>
```

```

        <xs:element ref="codigo-proveedor"/>
        <xs:element ref="nombre-proveedor"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

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

<xs:attribute name="cod" type="xs:integer" />

<xs:element name="codigo-proveedor">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="PR-\d{3}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="stock">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:integer">
                <xs:attribute name="fecha" type="xs:date" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
</xs:schema>

```

## 9.2. Modelo con tipos de nombres reutilizables.

Se definen los tipos de datos simples o complejos dándoles un nombre. Cuando se declaren los elementos y los atributos se indicarán que son de algunos de los tipos declarados. Ejemplo:

```

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

    <xs:element name="almacen" type="T_almacen"/>

    <xs:complexType name="T_almacen">
        <xs:sequence>
            <xs:element name="articulo" type="T_articulo"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="T_articulo">
        <xs:sequence>
            <xs:element name="descripcion" type="T_descripcion"/>
            <xs:element name="proveedor" type="T_proveedor"/>
            <xs:element name="stock" type="T_stock"/>
        </xs:sequence>
        <xs:attribute name="cod" type="T_cod" use="required" />
    </xs:complexType>

    <xs:simpleType name="T_descripcion">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>

    <xs:complexType name="T_proveedor">
        <xs:sequence>
            <xs:element name="codigo-proveedor" type="T_codigo_proveedor"/>
            <xs:element name="nombre-proveedor" type="T_nombre_proveedor"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="T_codigo_proveedor">
        <xs:restriction base="xs:string">
            <xs:pattern value="PR-\d{3}"/>
        </xs:restriction>
    </xs:simpleType>

```

```

<xs:simpleType name="T_nombre_proveedor">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:complexType name="T_stock">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="fecha" type="xs:date" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="T_cod">
  <xs:restriction base="xs:integer"/>
</xs:simpleType>

</xs:schema>

```

### 9.3. Modelo de diseño anidado.

Cada elemento y atributo se describe en el mismo lugar donde se declara. Esto puede producir que elementos del mismo tipo se describan varias veces, y que elementos con el mismo nombre tengan distintas descripciones.

Se consiguen esquemas más compactos pero más difíciles de entender y seguir. Ejemplo:

```

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

  <xs:element name="almacen">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="articulo">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="descripcion" type="xs:string"/>
              <xs:element name="proveedor">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="codigo-proveedor">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:pattern value="PR-\d{3}"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                    <xs:element name="nombre-proveedor" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>

              <xs:element name="stock">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:integer">
                      <xs:attribute name="fecha" use="required" type="xs:date"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>

            </xs:sequence>
            <xs:attribute name="cod" use="required" type="xs:integer"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>

```

