Tema 3c: XML Schema

XML Schema Definition (XSD)

3º Parcial - LMSGI

1º de Ciclo Superior de DAM

Ies de Teis (Vigo) – Curso 2020/21

Tema 3c: XML Schema

Objetivos: *crear XML Schemas para validar documentos XML, reconociendo su potencia y versatilidad respecto a los DTD´s*

- **Limitaciones de los DTD´s**
- **Necesidad de XML Schemas frente a DTD´s**
- **Características**
- **Ventajas**
- **Inconveniente**: complejidad
- **Sintaxis básica:**
  - **Estructura**
  - **Diseño**
- **Tipos de datos:**
  - **Simples**
  - **Complejos**

Limitaciones de los *DTD´s*

- **Limitaciones de los DTD´s:**
  - ✓ **Esquema cerrado** (closed): **no** se permite una reutilización ni una ampliación del documento mediante elementos no definidos => **no es extensible**
  - ✓ **No se basa en XML**: los ***parsers*** no pueden comprobar si están bien formados ya que no utiliza la sintaxis XML
  - ✓ Al **no soportar tipado** de elementos, **No** se pueden definir declaraciones globales ni fijar restricciones sobre los **tipos de datos** de elementos y atributos: tipo de datos, tamaño, etc
  - ✓ **No soporta los *espacios de nombres (XMLNS)*** (no los puede validar)
  - ✓ **No se pueden enumerar** los posibles valores de los **elementos**
  - ✓ Los **elementos no** pueden tener **valores por defecto**
  - ✓ **Control limitado** sobre las **cardinalidades** de los elementos (0, 1, ∞)
  - ➢ **Indica**: ¿cómo se definen los elementos que deben almacenar un nº de expediente, una fecha, una altura, un peso, o incluso el indicar si se cumple o no un determinado requisito? ¿qué implica dicha definición de **tipo**? ¿los **atributos** tienen las mismas restricciones?

Limitaciones de los *DTD´s*

- **Sol:** todos los elementos pueden contener cualquier cadena de caracteres, sin poder validar si el contenido asignado es el adecuado

```
<!ELEMENT garderia (neno*)>
<!ELEMENT neno (nome, dataNacemento, peso, altura, vacinas)>
<!ATTLIST neno numeroExpediente CDATA #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT dataNacemento (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
<!ELEMENT altura (#PCDATA)>
<!ELEMENT vacinas (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo UD7_1  -->
<!DOCTYPE garderia SYSTEM "garderia.dtd">
<garderia>
  <neno numeroExpediente="2344">
    <nome>Noa Blanco Coello</nome>
    <dataNacemento>2010-10-12</dataNacemento>
    <peso>13.5</peso>
    <altura>78</altura>
    <vacinas>true</vacinas>
  </neno>
</garderia>
```

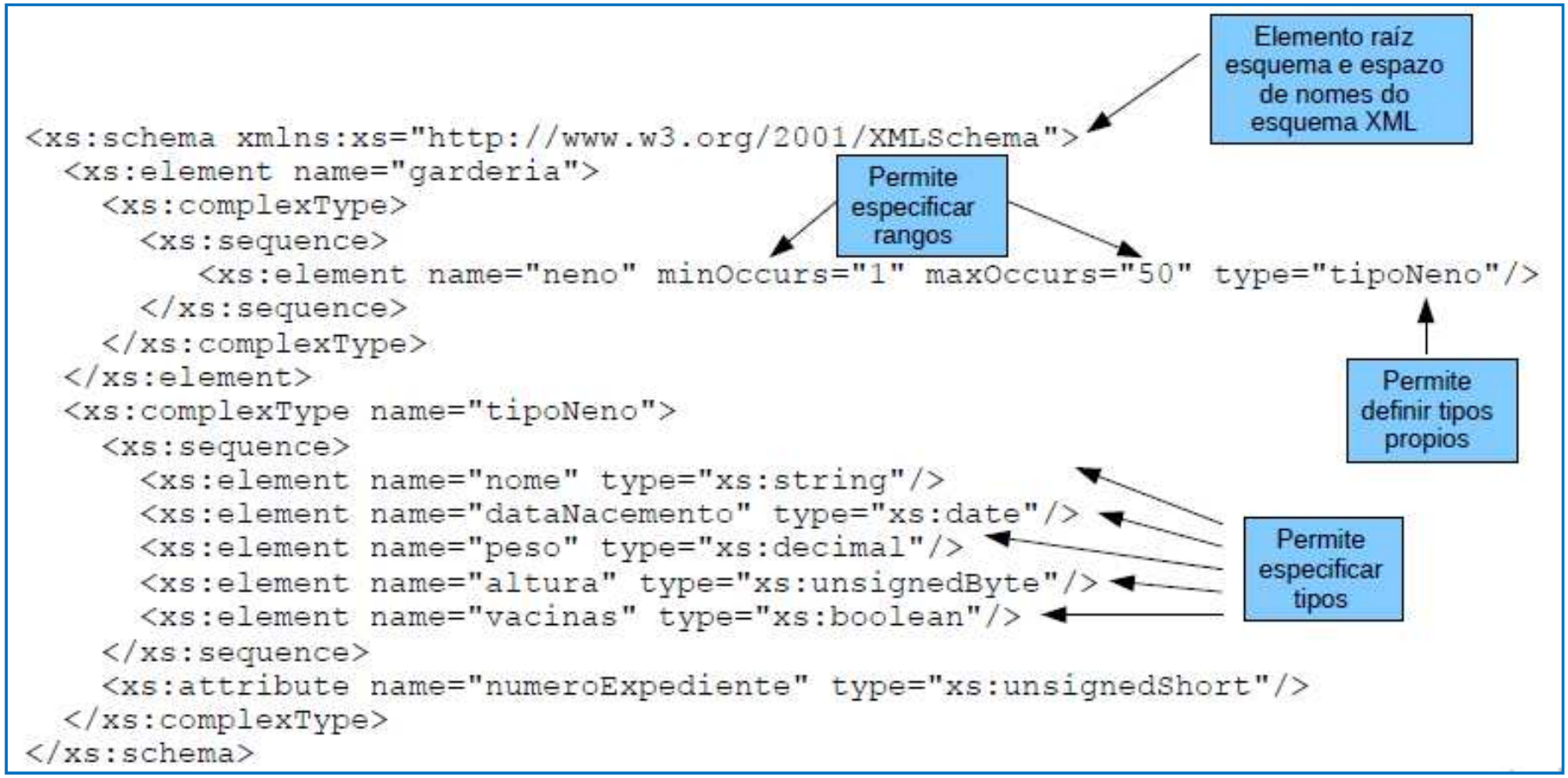
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Exemplo UD7_1  -->
<!DOCTYPE garderia SYSTEM "garderia.dtd">
<garderia>
  <neno numeroExpediente="popeye">
    <nome>cousas</nome>
    <dataNacemento>1.1.1</dataNacemento>
    <peso>non o sei</peso>
    <altura>falso</altura>
    <vacinas>no</vacinas>
  </neno>
</garderia>
```

XML Schema

- **XML Schema**:
  - ✓ Recomendación desarrollada por la **W3C** alrededor de las tecnologías de XML
  - ✓ Permite definir la **estructura** y el **contenido** d un documento XML
  - ✓ Se puede definir un **lenguaje de marcas** basado en **XML**
  - ✓ Usa **sintaxis XML** más **fácil** de entender
  - ✓ Permiten definir **tipos de datos**
  - ✓ Permiten **secuencias desordenadas**
  - ✓ Permiten **definiciones sensibles al contexto**. (comportamiento diferente de un elemento según donde esté situado)
  - ✓ Soportan ***espacios de nombres***
  - ✓ Soportan **referencias cruzadas** para formar claves a partir de varios atributos o nombres
  - ✓ Tecnología similar a **DTD**, pero más **potente** y **versátil** ya que resuelve varios problemas de los que presenta

XML Schema

- **Ej. de un XML Schema**:




```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="garderia">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="neno" minOccurs="1" maxOccurs="50" type="tipoNeno"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tipoNeno">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="dataNacemento" type="xs:date"/>
      <xs:element name="peso" type="xs:decimal"/>
      <xs:element name="altura" type="xs:unsignedByte"/>
      <xs:element name="vacinas" type="xs:boolean"/>
    </xs:sequence>
    <xs:attribute name="numeroExpediente" type="xs:unsignedShort"/>
  </xs:complexType>
</xs:schema>
```

Ventajas de XML Schema

- **Ventajas de XML Schema**:
  - ✓ **Tipado fuerte**: con tipos de datos similares a los de lenguajes de programación y bd´s
  - ✓ Permite **extender los tipos de datos**: se pueden crear **nuevos tipos** de datos a partir de los tipos ya existentes
  - ✓ Ofrece **características de orientación a objetos**: tipos de datos, *herencia* y *reutilización* de tipos
  - ✓ **Cardinalidad precisa**: permite controlar con precisión el nº de repeticiones de cada elementos, y también indicar el valor mínimo y máximo de cada elemento
  - ✓ Es un **documento XML**: podemos utilizar la **sintaxis** ya aprendida de XML
  - ✓ Representación de **claves primarias** y **ajenas**: permite una verdadera representación y uso de claves primarias y ajenas
  - ➢ **Recuerda**: ¿qué mecanismo similar se utiliza en los DTD´s?

Otras Alternativas a DTD

- **Otras alternativas a DTD**:
  - Además de **XML Schema**, podemos encontrar **otras tecnologías/lenguajes** para validar documentos XML:
  - ✓ *Documento Content Description for XML* (*DCD*)
  - ✓ *XML Data Reduce* (*XDR*)
  - Microsoft:
    - *XML-Data*
    - *BizTalk*
  - ✓ *Document Definition Markup Language* **(***DDML***)**
  - ✓ *Schema for O-O XML* (*SOX*)
  - ✓ *Relax NG*
  - ✓ *Schematron*

Especificación de XML Schema

- **Especificación de XML Schema**:
  - Desarrollada por la ***World Wide Web Consortium*** (**W3C**) a partir de 1998
  - Dividida en **tres partes** o documentos que son recomendación oficial desde Octubre de 2004
  - ✓ **XML Schema Part 0**: es una introducción **no oficial** al lenguaje, que proporciona una gran cantidad de **ejemplos** y explicaciones detalladas para una primera aproximación a XML Schema. Sirve de **referencia** para los siguientes partes.
  - ✓ **XML Schema Part 1 Structures**: es una extensa descripción de los componentes del lenguaje. Es **oficial**. Describe los **mecanismos de construcción** de las **estructuras** de datos y define las **reglas** de **validación** que se deben aplicar.
  - ✓ **XML Schema Part 2 Datatypes**: complementa la Parte 1 con la definición de los **tipos de datos primitivos** incorporados en XML Schema y sus restricciones. Es **oficial**.

Estructura de XML Schema

- **Sintaxis básica de XML Schema: estructura**
  - ✓ Define los **elementos** que hay que utilizar y su contenido
  - ✓ Define los **atributos** que se pueden emplear en cada elemento con valores válidos
  - ✓ Define la **estructura**: **orden** en el que se van añadiendo los elementos en un documento.
  - La forma más sencilla de entender cómo se crea la **estructura de un XML Schema** es partir de un **DTD sencillo**
  - **Ej:** a partir de este DTD iremos creando el esquema correspondiente

```
<!ELEMENT BIBLIOTECA (LIBRO+)>
<!ELEMENT LIBRO (TITULO, AUTOR*, ANYO?,
  EDITORIAL?)>
<!ATTLIST LIBRO COD CDATA #REQUIRED>
<!ELEMENT AUTOR (#PCDATA)>
<!ELEMENT TITULO (#PCDATA)>
<!ELEMENT EDITORIAL (#PCDATA)>
<!ELEMENT ANYO (#PCDATA)>
```

Estructura de XML Schema

- **Indicaciones generales**:

1) Por convenio, la **extensión** de los ficheros XML Schema es **.xsd** (*XML Schema Definition*)

2) Un XML **Schema** es un **documento XML** =>

   - ✓ Contiene una **declaración** de documento XML
   - ✓ Tiene un **elemento raíz**, pero siempre se utiliza como elemento global <**schema**>
   - ✓ Dicho elemento raíz viene con el **prefijo** (**xs** o **xsd**) referido al ***espacio de nombres*** que se **define** a continuación
   - ✓ Se utiliza la **declaración** de ***espacio de nombres*** de XML Schema para sus ***elementos***

- **Ej:**

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

Estructura de XML Schema

- **Indicaciones generales**:

  3) **Comentarios**:

  - ✓ Se pueden añadir desde **cualquier parte** del documento de dos **formas**:

  a) **<!-- comentario -->**

  b) Dentro del elemento <**annotation**>: opción recomendada, se pueden **estructurar** e incluso **procesar** como un fragmento de documento Xml

  i. <**documentation**> para introducir indicaciones dirigidas a los usuarios, como la fecha de creación, copyright, etc.

  ii. <**appinfo**> para añadir información extra que manejan los ordenadores, como indicar el fichero css que se enlaza

- **Ej:**

```
<?xml version="1.0" encoding="UTF-8“ ?>
<xs:schema xmlns:xs=“http://www.w3.org/2001/XMLSchema”>
   <xs:annotation>
       <xs:documentation>
             Info relacionada con el doc XML: biblioteca
        </xs:documentation>
    <xs:annotation>
</ xs:schema >
```

Estructura de XML Schema

- **Indicaciones generales**:
  - 4) **Definición básica de Elementos** => <**element**> `<xs:element name=“”></xs:element>`
    - **Tipos de Elementos**
      - a) **Simples**: sólo contienen **datos** de uno de los ***tipos de datos*** admitidos
        - **Sintaxis:** `<xs:element name=“” type=“” />`
        - **Ej**: `<xs:element name="año" type="xs:integer" minOccurs="0" maxOccurs="1”>`
      - b) **Complejos**: pueden contener **atributos**, **elementos hijos** y **contenido mixto**
        - **Sintaxis:** <**complexType**>

```
<xs:element name=“”>
  <xs:complexType>
     …..
  </xs:complexType>
</xs:element>
```

        - **Ej:** definición del **elemento complejo** *biblioteca* que contiene un elem. hijo

```
<xs:element name=“biblioteca”>
  <xs:complexType>
    <xs:sequence>
      <xs:element name=“libro" minOccurs=“1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Estructura de XML Schema

- **Indicaciones generales**:

  5) **Formas de indicar los elementos hijos de un elemento complejo** =>

  a) **sequence <sequence>**: define una **secuencia ordenada** de subelementos, como los elem q aparecen separados por **,** en DTD

  - Permite indicar el mínimo/máximo nº d veces que aparece un elemento con los atributos **minOccurs** y **maxOccurs**, pudiéndole asignar valores entre **0** y **unbounded** como valor mínimo/máximo posible

  - **Ej:**

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

  b) **choice <choice>**: define un grupo de **elementos** mutuamente **excluyentes**, equivale a **|** en DTD

  - Permite elegir un valor mínimo y un valor máximo

  c) **all <all>**: define un **grupo no ordenado** de elementos, **no** es obligatorio que aparezcan todos ni en el mismo orden

  - Cada elemento puede aparecer un **máximo de 1** vez (entre 0 y 1 **minOccurs** y 1 para **maxOccurs**)

Estructura de XML Schema

- **Indicaciones generales:**

  5) **Ej. de agrupación de elementos hijos =>**

```
<xs:element name="camiseta">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="color" type="xs:string"/>
      <xs:element name="talla" type="xs:string"/>
    <xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="vehiculoMotor">
  <xs:complexType>
    <xs:choice>
      <xs:element name="coche" type="xs:string"/>
      <xs:element name="moto" type="xs:string"/>
      <xs:element name="furgoneta" type="xs:string"/>
      <xs:element name="camion" type="xs:string"/>
    <xs:choice>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="camiseta">
  <xs:complexType>
    <xs:all>
      <xs:element name="color" type="xs:string"/>
      <xs:element name="talla" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Estructura de XML Schema

- **Indicaciones generales**:

  6) **Indicar el elemento raíz del documento XML en el Schema**
     - NO hay ningún método
     - Se suele hacer definiendo un **único elemento declarado a *nivel global* como elemento hijo** del **elemento raíz** <schema>
     - **Ej:** definir el elemento <**biblioteca**> dentro de <**schema**>

  7) **Crear elementos simples**
     - Ya vimos que hay que indicar su nombre con el atributo **name**, su tipo con **type**, y el nº de veces mínimo y máximo que puede aparecer con **minOccurs** y **maxOccurs**. Sólo pueden contener **datos simples**.
     - Cuando no hay límite en el nº de ocurrencias el valor es **unbounded**
     - **fixed** para especificar un valor fijo del elemento
     - **default** para indicar un valor como predeterminado
     - **Ej:** ¿son equivalentes las siguientes declaraciones del elemento título? ¿por qué?

```
<xs:element name="titulo" type="xs:string" minOccurs="1" maxOccurs="1">
<xs:element name="titulo" type="xs:string">
```

- **1ª aproximación a la creación de un elemento complejo**: ¿cómo hay que declarar el elemento *libro* si contiene como elementos hijos simples *título, autor, año* y *editorial*?

Ejercicio de elementos de XML Schema

➢ **Sol.:**

- Para indicar que es un **elemento complejo**: <**complexType**>

```
<xs:element name="LIBRO" minOccurs="1"
  maxOccurs="unbounded">
<xs:complexType>
  <xs:sequence>
  …
  </xs:sequence>
</xs:complexType>
</xs:element>
```

- Para indicar la **secuencia** y **orden** de elementos que lo componen: <**sequence**>

```
<xs:element name="TITULO" type="xs:string"
  minOccurs="1" maxOccurs="1" />
<xs:element name="AUTOR" type="xs:string"
  minOccurs="0" maxOccurs="unbounded" />
<xs:element name="ANYO" type="xs:string"
  minOccurs="0" maxOccurs="1" />
<xs:element name="EDITORIAL"
  type="xs:string" minOccurs="0"
  maxOccurs="1" />
```

- Para ir definiendo cada uno de los **elementos simples**: <**element name**=“” **type**=“” **minOccurs**=“” **maxOccurs**=“”>
- Los **tipos** fijos de los elementos deben llevar el prefijo correspondiente al **espacio de nombres** que se esté utilizando en el documento, si es **xs => type=“xs:tipo”**

Estructura de XML Schema

- **Indicaciones generales**:

  8) **Asignar un atributo a un elemento: <attribute>**

  - Los atributos aportan información a los elementos
  - Cada atributo queda identificado por su nombre a través de **name** y por el tipo de dato que se le asigna con **type**
  - **Ej**:

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

  - La declaración de atributos para un elemento deben aparecer al **final** del bloque de la declaración de dicho elemento (antes del cierre de la etiqueta <**complexType**>
  - **Ej:** definir el atributo **codigo** del elemento <**libro**>

```
<xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <!--elementos d libro: titulo, autor, año y editorial -->
        </xs:sequence>
        <xs:attribute name="codigo" type="xs:string" />
    </xs:complexType>
</ xs:element >
```

Estructura de XML Schema

- **Indicaciones generales**:

  8) **Reglas para la declaración de atributos**

  - ✓ Se definen al **final del bloque de la declaración** del elemento asociado
  - ✓ **No** pueden contener **hijos**
  - ✓ Se consideran como elementos de **tipo simple** (sólo pueden contener texto)
  - ✓ Su declaración **no** impone un **orden** de uso (se asignan a los elementos en cualquier orden)
  - ✓ Si no se indica el tipo de valor que se le asigna, su **tipo por defecto** es ***anySimpleType***, que representa cualquier **cadena de caracteres XML válidos**
  - ✓ **No** se puede especificar la **cardinalidad**: sólo aparecen **1 vez** en el elemento
  - ✓ Si no se indica nada, los atributos son **opcionales** (**valor por defecto**)
  - **Restringir** los **valores** del **atributo** a definir:
    - ✓ **use**: con los valores **optional** (valor predetermindo, puede o no aparecer), **required** (es obligatorio) o **prohibited** (para que **NO** aparezca en el elemento)
    - ✓ **default**: para definir el **valor por defecto** del atributo, cuando no se añade el atributo, el procesador XML empleará dicho valor para ese atributo. **Sí** aparece en el DOM del documento
    - ✓ **fixed**: para definir un **valor fijo** del atributo. Puede aparecer o no en el elemento, pero si aparece, sólo puede tener ese valor.

Estructura de XML Schema

- **Indicaciones generales**:

  9) **Enlace a un esquema sin espacio de nombres propio**

  a) **Definición en el esquema (.xsd)**:

  - Incluir la declaración del **espacio de nombres** al que pertenecen los componentes incluidos en el esquema

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

  b) **Enlace en el documento *instancia* XML (.xml)**:

  - Indicar el **elemento raíz** del documento xml
  - Añadir el atributo **noNamespaceSchemaLocation** para indicar el fichero esquema que validará dicho documento XML

```
<?xml version="1.0" ... ?>
<elementoRaíz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:noNamespaceSchemaLocation="esquema.xsd">
...
</elementoRaíz>
```

Ejemplo de un documento XML y su XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<alumnado xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="elementosEsquema.xsd">
    <nombre>Ramon</nombre>
    <edad>34</edad>
    <altura>1.78</altura>
    <fecha_nac>2004-10-09</fecha_nac>
    <conduce>false</conduce>
    <hora_llegada>12:30:00</hora_llegada>
</alumnado>
```

El elemento **<schema>** es el elemento raíz

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="alumnado">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nombre" type="xs:string" />
                <xs:element name="edad" type="xs:integer" />
                <xs:element name="altura" type="xs:decimal" />
                <xs:element name="fecha_nac" type="xs:date" />
                <xs:element name="conduce" type="xs:boolean" />
                <xs:element name="hora_llegada" type="xs:time" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:annotation>
        <xs:documentation>
            Ejemplo en el que incluimos un elemento llamado "alumnado“
             que será el elemento raíz y se define como un elemento
            complejo para poder incluir la secuencia de elementos
            que contiene
        </xs:documentation>
    </xs:annotation>
</xs:schema>
```

Ejercicio de elementos de XML Schema

- **Ejercicio1:** Con las indicaciones **generales** vistas hasta ahora, ya podemos crear nuestro primer esquema (**.xsd**). La información que hay que validar es la que se almacena sobre los libros que contiene una **biblioteca**.
  - El **atributo** código del elemento libro es obligatorio
  - Recuerda que los elementos que contienen cadenas de caracteres se definen como tipo *string*, con el prefijo del espacio de nombres correspondiente (**xs:string**)
  - Crea un documeto XML y valídalo con el esquema creado
  - **Ej**:

```
−<biblioteca xsi:noNamespaceSchemaLocation="biblioteca.xsd">
  −<libro codigo="114">
      <titulo>Pequeña Historia de España</titulo>
      <autor>Manuel Fernández Álvarez</autor>
      <año>2008</año>
      <editorial>Espasa</editorial>
    </libro>
  −<libro codigo="345">
      <titulo>El loco cuento de todos los cuentos</titulo>
      <autor>Jose María Plaza</autor>
      <año>2013</año>
      <editorial>Santillana</editorial>
    </libro>
</biblioteca>
```

Ejercicio de elementos de XML Schema

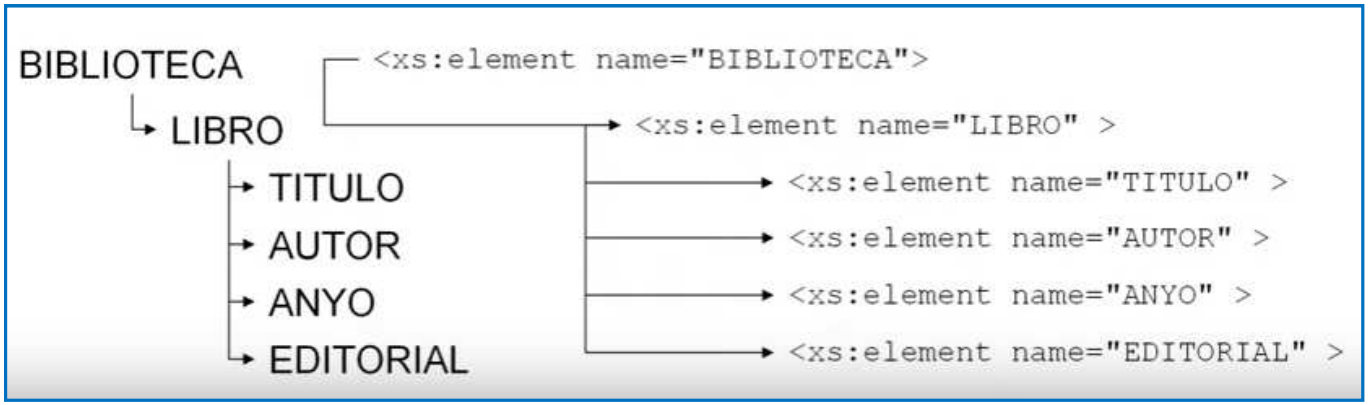
- **Sol.: definición del esquema de *biblioteca***

- El **elemento complejo** *biblioteca*, está a su vez formado por otro elemento complejo *libro*, que contiene varios elementos hijos simples => ***diseño muñecas rusas***

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="biblioteca">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
    <!--elementos de libro: titulo, autor, año y editorial -->
     <xs:complexType>
     <xs:sequence>
      <xs:element name="titulo" type="xs:string" />
      <xs:element name="autor" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
      <xs:element name="año" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="editorial" type="xs:string" minOccurs="0" maxOccurs="1" />
     </xs:sequence>
     <xs:attribute name="codigo" type="xs:string" use="required" />
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Ejercicio de elementos de XML Schema

- **Sol.: definición del esquema de *biblioteca***
  - **Representación en árbol** del documento XML y del esquema:



  - **Visualización** desde el **navegador**:

```
<?xml version="1.0" encoding="UTF-8"?>
- <biblioteca xsi:noNamespaceSchemaLocation="biblioteca.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  - <libro codigo="114">
      <titulo>Pequeña Historia de España</titulo>
      <autor>Manuel Fernández Álvarez</autor>
      <año>2008</año>
      <editorial>Espasa</editorial>
    </libro>
  - <libro codigo="345">
      <titulo>El loco cuento de todos los cuentos</titulo>
      <autor>Jose María Plaza</autor>
      <año>2013</año>
      <editorial>Santillana</editorial>
    </libro>
</biblioteca>
```

Estructura de XML Schema

- **Indicaciones generales**:

  10) Tipos de declaración de elementos y atributos

  a) Declaración global:

  - El elemento/atributo a declarar tiene que ser **hijo directo** del elemento raíz del esquema <**schema**>
  - Cualquier otro tipo complejo puede hacer **uso de esa declaración** mediante una ***referencia*** => permite la **reutilización** de código
  - **ref**: atributo que permite referenciar un elemento/atributo global, sustituye a **name**
  - **Ej:** declarar a **nivel global** los **atributos** *system* y *dim* y utilizarlos en el **tipo nuevo de datos** *SizeType* a través de **ref** indicando el nombre del atributo global

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:attribute name="system" type="xsd:string"/>
<xsd:attribute name="dim" type="xsd:integer"/>

<xsd:complexType name="SizeType">
  <xsd:attribute ref="system" use="required"/>
  <xsd:attribute ref="dim"/>
</xsd:complexType>
</xsd:schema>
```

Estructura de XML Schema

- **Indicaciones generales**:

  10) Tipos de declaración de elementos y atributos

  b) Declaración local:

  - El elemento/atributo a declarar NO es **hijo directo** de <**schema**>
  - Su declaración está anidada **dentro** de la estructura del esquema, es decir, en el **interior** de una declaración de un elemento complejo o atributo.
  - Al ser locales **NO se pueden reutilizar** en otro elemento
  - **Ej:** declarar a **nivel local** los **atributos** *system* y *dim* del tipo *SizeType*

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="SizeType">
  <xsd:attribute name="system" type="xsd:string"
   use="required"/>
  <xsd:attribute name="dim" type="xsd:integer"/>
</xsd:complexType>
</xsd:schema>
```

- **IMPORTANTE**: Según como declaremos los elementos/atributos, a nivel **local** o a nivel **global**, se utilizará un tipo de **diseño** de esquema diferente.
- ¿Cómo se han declarado los elementos del ejercicio anterior?

Ejercicio de elementos globales de XML Schema

- **Ejercicio2:** modifica el ejercicio anterior y declara los elementos *título*, *autor*, *año* y *editorial* como **elementos globales** para poder reutilizarlos.
  - En la propia definición del elemento complejo *libro*, crea las ***referencias*** adecuadas para que el elemento quede bien definido
  - Modifica el nº de apariciones de los elementos globales donde consideres oportuno de forma que el único elemento que se puede repetir hasta 5 veces es el de *autor*
  - Copia el documeto XML anterior, vincúlalo con el nuevo esquema, y valídalo

```
−<biblioteca xsi:noNamespaceSchemaLocation="biblioteca2.xsd">
 −<libro codigo="114">
    <titulo>Pequeña Historia de España</titulo>
    <autor>Manuel Fernández Álvarez</autor>
    <autor>Jvlivs</autor>
    <año>2008</año>
    <editorial>Espasa</editorial>
  </libro>
 −<libro codigo="345">
    <titulo>El loco cuento de todos los cuentos</titulo>
    <autor>Jose María Plaza</autor>
    <año>2013</año>
    <editorial>Santillana</editorial>
  </libro>
</biblioteca>
```

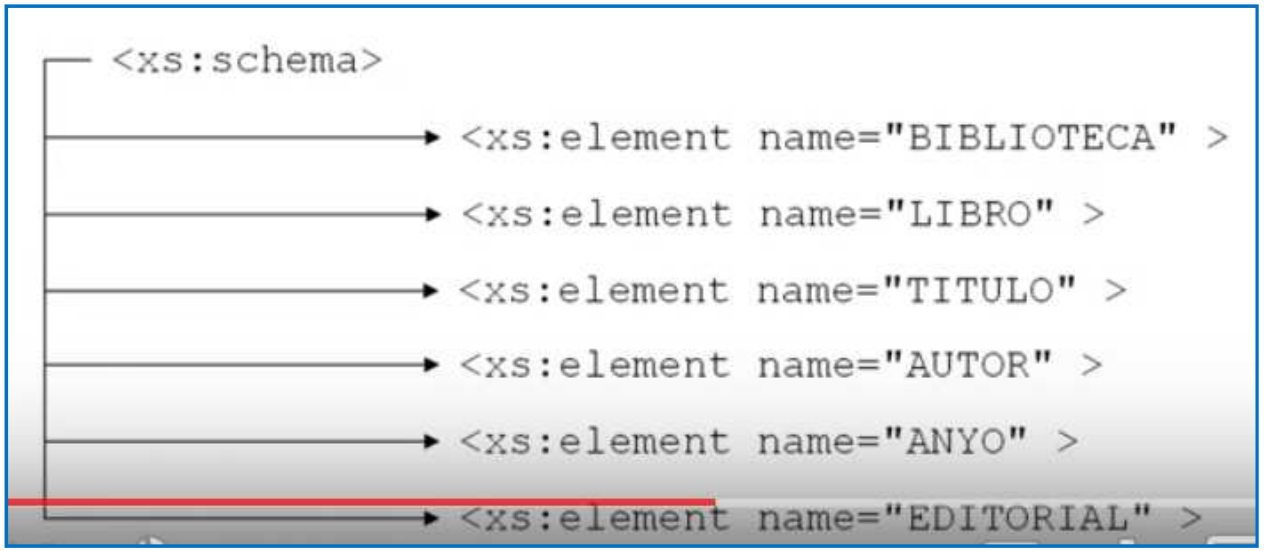
Ejercicio de elementos globales de XML Schema

- **Sol. Ej2: declaración de los elementos hijos de libro como globales**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="biblioteca">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
    <!—elementos de libro: titulo, autor, año y editorial—>
     <xs:complexType>
     <xs:sequence>
      <xs:element ref="titulo" />
      <xs:element ref="autor" maxOccurs="5" />
      <xs:element ref="año" />
      <xs:element ref="editorial" />
     </xs:sequence>
     <xs:attribute name="codigo" type="xs:string" use="required" />
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
<xs:element name="titulo" type="xs:string" />
 <xs:element name="autor" type="xs:string" />
 <xs:element name="año" type="xs:string" />
 <xs:element name="editorial" type="xs:string" />
</xs:schema>
```

Ejercicio de elementos globales de XML Schema

- **Ejercicio2b:** modifica el ejercicio anterior y declara todos los elementos y atributos como **elementos globales**.
  - Crea las **referencias** adecuadas donde sea necesario para que todos los elementos queden bien definidos
  - Copia el documeto XML, indica el nuevo esquema y valídalo
  - El **diseño** empleado para la creación de este ejercicio recibe el nombre de "***diseño plano***" o "***flat design***"
  - **Ej:** representación en árbol del esquema. Todos los elementos son hijos del elemento raíz <schema>



Ejercicio de elementos globales de XML Schema

- **Sol. Ej2b: declaración de todos los elementos/atributos globales** => ***diseño plano***

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:attribute name="codigo" type="xs:string" />
 <xs:element name="biblioteca">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="libro" minOccurs="1" maxOccurs="unbounded" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="libro">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="titulo" />
    <xs:element ref="autor" maxOccurs="5" />
    <xs:element ref="año" minOccurs="0" />
    <xs:element ref="editorial" minOccurs="0" />
   </xs:sequence>
   <xs:attribute ref="codigo" use="required" />
  </xs:complexType>
 </xs:element>
 <xs:element name="titulo" type="xs:string" />
 <xs:element name="autor" type="xs:string" />
 <xs:element name="año" type="xs:string" />
 <xs:element name="editorial" type="xs:string" />
</xs:schema>
```

Estructura de XML Schema

- **Indicaciones generales**:

  11) Definir el contenido de un elemento => *Modelo de contenido*

  a) **Texto:** el elemento sólo puede contener una **secuencia de caracteres**

  - Es lo que ya hemos visto sobre los **elementos simples**
  - **Sintaxis:** <**xs:element name**="nombre" **type**="tipo"
    [**minOccurs**="valor"] [**maxOccurs**="valor"] [**default**="valor"] [**fixed**="valor"]/>

  b) **Elementos:** el elemento contiene **elementos hijos**

  - Es lo que ya hemos visto sobre los **elementos complejos**
  - **Sintaxis:**

```
<xs:element name="nombre">
    <xs:complexType>
        <xs:[sequence/choice/all]>
            <xs:element name="hijo1" type="tipo" />
            <xs:element name="hijo2" type="tipo" />
        </xs:[sequence/choice/all]>
    </xs:complexType>
</xs:element>
```

Estructura de XML Schema

- **Indicaciones generales**:

  11) Definir el contenido de un elemento => *Modelo de contenido*

  c) **Vacío:** el elemento **no** puede contener **datos** de tipo carácter **ni** elementos **hijos**.

  - Sí puede tener **atributos =>** declararlo como **tipo complejo**
  - **Sintaxis:** <xs:**element name**="nombre">
    <**xs:complexType**>
    <**xs:attribute name**="nombre" **type**="tipo"
    [**use**="valor"] [**default**="valor"] [**fixed**="valor"]/>
    </xs:**complexType**>
    </xs:**element**>
  - **Ej:** definición de un elemento vacío en Schema y uso en un documento XML

```
<xs:element name="beca" >
   <xs:complexType>
      <xs:attribute name="valor" type="xs:string" />
   </xs:complexType>
</xs:element>
```

```
<beca valor="SI" />
```

Estructura de XML Schema

- **Indicaciones generales**:

  11) Definir el contenido de un elemento => *Modelo de contenido*

  d) Mixto: el elemento puede contener **datos** de tipo carácter, elementos **hijos** y **atributos**

  - Son útiles cuando se necesita enviar texto con elementos
  - Añadir el atributo **mixed** con valor ***true*** al elemento <complexType>
  - **Sintaxis:** <**xs:element name**="nombre">
    <**xs:complexType mixed="true"**>
    <xs:**[sequence/choice/all]**>
    <xs:**element name**="hijo1" **type**="tipo" />
    <xs:**element name**="hijo2" **type**="tipo" />
    </xs:**[sequence/choice/all]**>
    </xs:**complexType**>
    </xs:**element**>

  ➢ **Ej**: crea la definición de esquema del elemento <analisis> correspondiente al siguiente contenido en XML

```
<analisis>
El videojuego <titulo>Dragon Quest</titulo> desarrollado por
<desarrolladora>Level 5</desarrolladora> es uno de los mejores juegos del grupo de desarrolladores.
Lanzado originalmente en <lanzamiento>2009-07-11</lanzamiento>.
</analisis>
```

Estructura de XML Schema

- **Indicaciones generales**:

  11) Definir el contenido de un elemento => *Modelo de contenido*

  d) Mixto:

  ➢ **Sol**: definición del elemento mixto <analisis>

```
<xs:element name="analisis" minOccurs="1" maxOccurs="unbounded">
<xs:complexType mixed="true">
  <xs:sequence>
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="desarrolladora" type="xs:string"/>
    <xs:element name="lanzamiento" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

```
<analisis>
    El videojuego
    <titulo>Dragon Quest</titulo>
    desarrollado por
    <desarrolladora>Level 5</desarrolladora>
    es uno de los mejores juegos del grupo de desarrolladores. Lanzado originalmen
    <lanzamiento>2009-07-11</lanzamiento>
    .
</analisis>
<analisis>
    El videojuego
    <titulo>El juego de los tronos</titulo>
    desarrollado por
    <desarrolladora>Marvell</desarrolladora>
    es un buen juego a nivel competitivo. Lanzado originalmente en
    <lanzamiento>2020-03-11</lanzamiento>
    .
</analisis>
```

Tipos de datos

- **Tipos de datos**:
- Recordando los **tipos de elementos**:
  - **Simples**: sólo pueden contener tipos simples, como texto.
  - **Complejos**: pueden contener otros elementos –elem. hijos-, texto y atributos-

➢ **Deduce**: ¿los **atributos** son elementos **simples** o **complejos**?

- Los **tipos de datos**, simples/complejos, se pueden **definir** de dos formas distintas:
  - **a través de un nombre**, con lo que pueden **reutilizarse** (declaración **global**)
  - como **anónimos**, con una definición **interna** y **no reutilizable** (dec. **local**)
- **type:** atributo que permite definir el **contenido** de un **atributo** o de un **elemento** indicando el **tipo de dato** que va a contener
- Hay **44 tipos de datos simples predefinidos**, se dividen en 2 **grupos**:

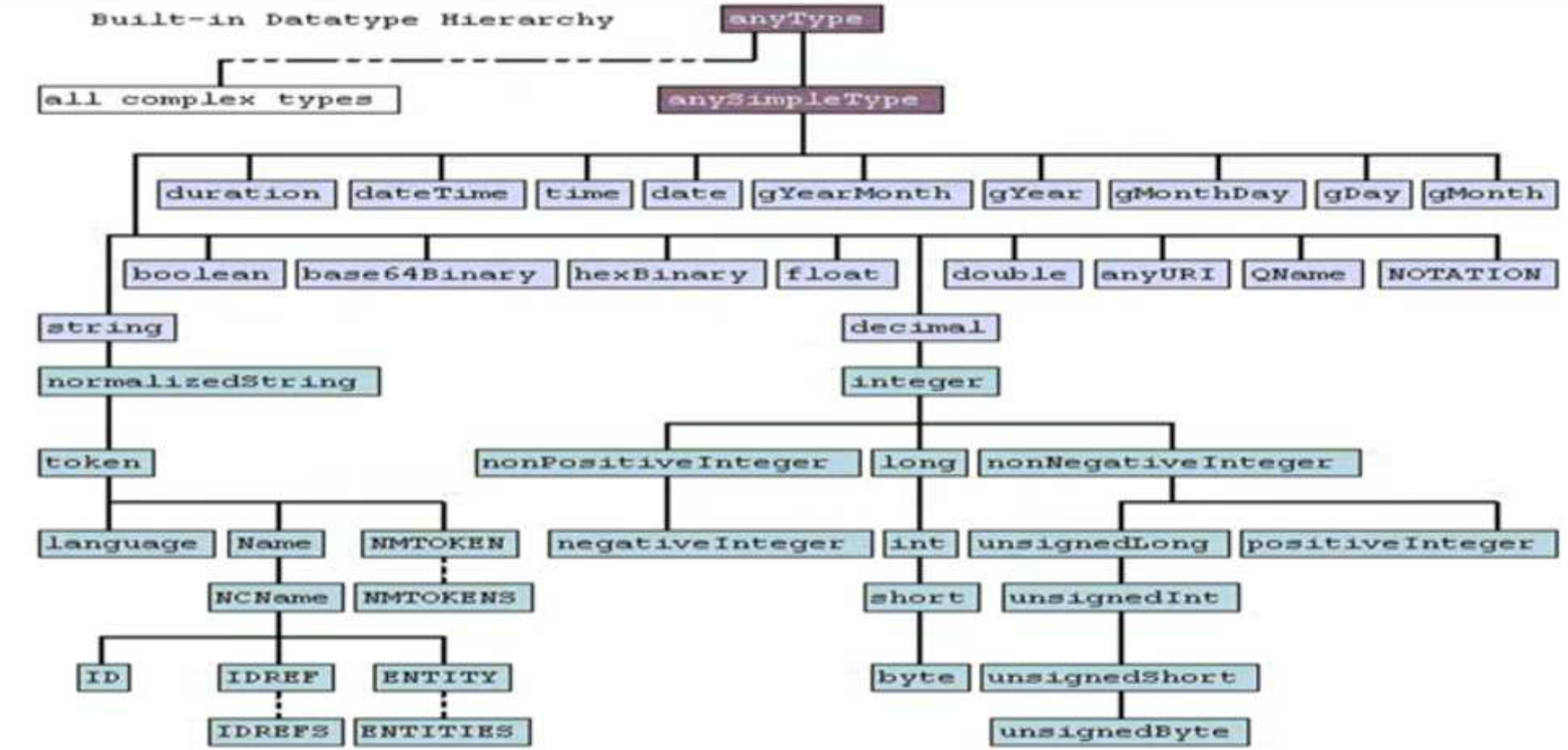
a) **Tipos primitivos**: no se definen en función de otros tipos, existen de por sí

b) **Tipos derivados**: se definen en función de otros tipos de datos, primitivos o derivados

- Se pueden **crear nuevos tipos de datos simples** por ***derivación*** de los existentes, utilizando el elemento <**simpleType**>

Tipos de datos simples

- **Tipos de datos simples**:
- Se pueden representar mediante un **árbol jerárquico** que contiene todos los tipo de datos simples, **primitivos** (primer bloque) y **derivados** (segundo bloque)
- Su raíz es el tipo <**anyType**> que no impone ninguna restricción en cuanto al posible contenido



Tipos de datos simples

- **Tipos simples predefinidos: numéricos**

decimal	Números reales
float	Número en punto flotante de precisión simple (32 bits)
double	Número en punto flotante de precisión simple (64 bits)
integer	Números enteros arbitrariamente largos
nonPositiveInteger	Números enteros negativos más el 0
negativeInteger	Números enteros negativos
positiveInteger	Números enteros positivos
unsignedLong	Números enteros positivos (64 bits)
unsignedInt	Números enteros positivos (32 bits)
unsignedShort	Números enteros positivos (16 bits)
unsignedByte	Números enteros positivos (8 bits)
long	Números enteros (64 bits)
Int	Números enteros (32 bits)
short	Números enteros (16 bits)
byte	Números enteros (8 bits)

- **Tipos simples predefinidos: de texto**

string	Cadenas de texto

Tipos de datos simples

- **Tipos simples predefinidos: de fecha y hora**

duration	Duración en años+meses+días+horas+minutos+segundos PnYnMnDTnHnMnS
dateTime	Fecha y hora aaaa-mm-dd T hh:mm:ss
date	Fecha aaaa-mm-dd
time	Hora hh:mm:ss
gDay	Solo el día ---dd
gMonth	Solo el mes --mm
gYear	Solo el año yyyy
gYearMonth	Año y mes yyyy-mm
gMonthDay	Mes y día --mm-dd

  - **Ej1:** el tipo ***duration*** permite especificar un período de tiempo indicando cuántos años, meses, días y horas, como P1Y4M21DT8H
  - **Ej2:** crear el elemento **estado2** referido al segundo estado de alarma e indicar el tiempo de vigencia: <xs:element name="estado2" type="xs:timeDuration> y <estado2>P6M1D</estado2>

- **Tipos simples predefinidos: booleanos**

boolean	4 valores 0, 1, false y true

Tipos de datos derivados

- **Tipos de datos derivados**:
- Una de las características más interesante de los esquemas es que se pueden **crear nuevos tipos de datos** a partir de los tipos ya existentes
- El atributo **base** indica el tipo del cual se está derivando
- Hay **2 formas de crear tipos** de datos **derivados**
- **Mecanismos de derivación**:
  a) **Por restricción**: para definir restricciones a los posibles valores que puede tomar un elemento/atributo. Cada **restricción** puede incluir varios aspectos o *facetas*. **Dependiendo** del **tipo base** se podrán aplicar unas u otras restricciones.
  b) **Por extensión**: no permite crear tipos de datos simples, sólo **complejos**
- Para **evitar que se deriven nuevos tipos de datos** a partir de un tipo ya existente, se utiliza el atributo **final**, en el que se indicará qué tipo de derivación no se permite, una de ellas o ambas (*extension*, *restriction*, *#all*)
- Si se emplea en la etiqueta <schema> se puede indicar el valor de **final** para todo el schema

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**:

- La forma más sencilla de crear un nuevo tipo a partir de uno ya existente es añadir **condiciones** a alguno de los **tipos predefinidos** en el **XML Schema**
- **Sintaxis**: <xs:**restriction base**="">...</xs:restriction>
- Hay **3 mecanismos** para restringir los posibles valores que puede tomar un elemento/atributo

i. **Por valor:** se restringen los posibles valores del tipo de dato

- **Ej**: definir un tipo de dato derivado como **anónimo (no** puede **reutilizarse)** en el que el nº máximo de caracteres es de 32

```
<xs:element name="NOMBRE">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ii. **Por enumeración:** el rango se debe escribir como la **lista** de valores válidos

iii. **Unión:** unión de 2 o más tipos de datos

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**: <xs:**restriction base**="">...</xs:restriction>

i. **Por valor:** se restringe el valor de los posibles valores del tipo de dato mediante:

- **Limitando la longitud de una cadena de texto**: indicando la longitud, la longitud máxima y mínima
  - **Restricciones**: <**length value**=""/>, <**maxLength value**=""/> y <**minLength**... />
  - **Ej**: declaración **global** de un nuevo tipo de dato (puede **reutilizarse**)

```
<xs:simpleType name="nombreUsuario">
  <xs:restriction base="xs:string">
    <xs:maxLength value="32"/>
  </xs:restriction>
</xs:simpleType>
```

- **Limitando el nº de dígitos de una cifra**: en parte entera y decimal, sólo decimal
  - **Restricciones**: <**totalDigits value**=""/>, < **fractionDigits value**=""/>
- **Limitando los valores máximo y mínimo de un nº**: <=, >=, <, >
  - **Restricciones**: <**minInclusive value**=""/>, <**maxInclusive**.../>, <**minExclusive**... />, <**maxExclusive** ... />
- **Empleando patrones** definidos mediante **expresiones regulares** mediante el elemento <**pattern value**=""/>: indicar el **formato** permitido del valor

Tipos de datos derivados por restricción

- **Listado de restricciones** o ***facetas*** que se pueden aplicar según el tipo base de datos

RESTRICCIÓN	DESCRIPCIÓN
length	Longitud exacta
minLength	Longitud mínima
maxLength	Longitud máxima
pattern	Patrón, expresión regular en la que debe encajar el contenido
enumeration	Lista de valores
whiteSpace	Gestión de los espacios en blanco
minInclusive	Mínimo valor (debe ser mayor o igual)
minExclusive	Mínimo valor (debe ser mayor)
maxInclusive	Máximo valor (debe ser menor o igual)
maxExclusive	Máximo valor (debe ser menor)
totalDigits	Número total de dígitos
fractionDigits	Número de cifras decimales

Tipos de datos derivados por restricción

a) **Por restricción**: <xs:**restriction base**="">...</xs:restriction>

i. **Por valor:** se restringe el valor de los posibles valores del tipo de dato:

- **Empleando patrones** definidos mediante **expresiones regulares**

Patrón	Significado
.	Cualquier carácter
\w	Cualquier letra
\d	Un dígito
\D	Cualquier carácter no dígito
\s	Cualquier carácter de espaciado
\S	Cualquier carácter de no espaciado
\n	Salto de línea
[xyz]	Uno de los caracteres x, y o z en minúscula
[A-Z]	Uno de los caracteres de A hasta Z en mayúscula
[^abc]	Negación de un grupo de caracteres
[F-J-[H]]	Sustracción de un carácter de un rango
(a|b)	Alternativa entre dos expresiones

Tipos de datos derivados por restricción

a) **Por restricción**: <xs:**restriction base**="">...</xs:restriction>

i. **Por valor:** se restringe el valor de los posibles valores del tipo de dato:

- **Empleando patrones** definidos mediante **expresiones regulares**: **cuantificación**

Patrón	Significado
?	0 o 1 ocurrencias
*	0 o más ocurrencias
+	1 o más ocurrencias
{n}	*n* ocurrencias
{n,m}	De *n* a *m* ocurrencias
{n,}	*n* o más ocurrencias
( )	Agrupación de elementos
\(	Carácter "("
\)	Carácter ")"

- **Ej**: crear el tipo "tipoTelefono" aplicando la restricción <**pattern value**=""/>

```
<xs:simpleType name="tipoTelefono">
   <xs:restriction base="xs:string">
      <xs:pattern value="\([0-9]{3}\)(-[0-9]{2}){3}"/>
   </xs:restriction>
</xs:simpleType>
```

Ejercicio de tipos derivados por valor

- **Ej. de creación de tipos de datos simples derivados por valor:**

a) Crea un **nuevo tipo** de dato **reutilizable** "**monedaEUR**" como un subtipo del tipo **base** "**decimal**", en el que el número total de cifras decimales sea de 2

b) Define el **elemento** simple "**edad**" como un nº entero con un valor posible entre 0 y 120. Dicho tipo estará definido como **anónimo**

c) Crea un **nuevo tipo** de dato **reutilizable** "**password**" como un subtipo del tipo **base** "**string**", como una cadena de texto entre 8 y 12 caracteres. Tras haber creado dicho tipo simple, asígnaselo como tipo de dato al elemento "**contraseña**".

d) Define un **atributo** "**dni**" obligatorio y que cumpla el formato de 8 dígitos y una letra mayúscula

Ejercicio de tipos derivados por valor

- **Sol. de creación de tipos de datos simples derivados por valor:**

a) Tipo de dato **reutilizable** "**monedaEUR**"

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

b) Elemento "**edad**" como **anónimo**

```
<xs:element name="edad">
<xs:simpleType>
 <xs:restriction base="xs:integer">
  <xs:minInclusive value="0"/>
  <xs:maxInclusive value="120"/>
 </xs:restriction>
</xs:simpleType>
```

c) Tipo **reutilizable** "**password**" y uso

```
<xs:simpleType name="password">
 <xs:restriction base="xs:string">
  <xs:minLength value="8"/>
  <xs:maxLength value="12"/>
 </xs:restriction>
</xs:simpleType>
```

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

d) Tipo "**dni**"

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

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**:

ii. **Por enumeración:**

- Para limitar el contenido de un elemento XML a un **conjunto de valores**, se utilizará la restricción <**enumeration**>.
- Por cada valor válido, hay que añadir un elemento <enumeration>
- **Sintaxis**: <**xs:enumeration value=“valor”**>
- **Ej**: definir un elemento simple llamado **lengProg** que permita elegir entre las opciones Java, C y Php

```
<xs:element name=“lengProg">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value=“Java"/>
      <xs:enumeration value=“C"/>
      <xs:enumeration value=“PHP"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ejercicio de tipos derivados por restricción de enumeración

- **Ej. de creación de tipos de datos simples derivados por restricción de enumeración:**

a) Crea un **nuevo elemento anónimo** llamado "**coche**" que contenga como posibles valores *Audi, Bmw, Citroen* y *Volkswagen*

b) Crea un **nuevo tipo** de dato **reutilizable** "**tipoCoche**" que contenga como posibles valores *Audi, Bmw, Citroen* y *Volkswagen*

- En los dos casos anteriores, comprueba el funcionamiento a través de un documento XML.

Ejercicio de tipos derivados por restricción de enumeración

- **Sol**. tipos de datos simples derivados por restricción de enumeración:

a) Elemento **anónimo** "**coche**"

```
<xs:element name="coches">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="coche" maxOccurs="unbounded">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Bmw" />
      <xs:enumeration value="Citroen"/>
      <xs:enumeration value="Volkswagen"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<coches xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="coche0.xsd">
  <coche>Audi</coche>
  <coche>Audi</coche>
  <coche>Volkswagen</coche>
</coches>
```

Ejercicio de tipos derivados por restricción de enumeración

- **Sol**. tipos de datos simples derivados por restricción de enumeración:

b) Nuevo tipo de dato **reutilizable** "**tipoCoche**"

```
<xs:element name="coches">
<xs:complexType>
<xs:sequence>
<xs:element name="coche" type="tipoCoche" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:simpleType name="tipoCoche">
<xs:restriction base="xs:string">
<xs:enumeration value="Audi"/>
<xs:enumeration value="Bmw" />
<xs:enumeration value="Citroen"/>
<xs:enumeration value="Volkswagen"/>
</xs:restriction>
</xs:simpleType>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<coches xsi:noNamespaceSchemaLocation="coche1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <coche>Audi</coche>
    <coche>Audi</coche>
    <coche>Volkswagen</coche>
</coches>
```

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**:

iii. **Unión:** permite crear nuevos tipos de datos a partir de la unión de varios tipos

- A partir de un tipo de dato ya definido, gracias la restricción por unión, se le pueden añadir más valores
- Para indicar que se va a producir una unión de tipos, se utiliza el elemento <**union**>
- Los tipos de los elementos se especifican en el atributo **memberTypes**
- **Sintaxis**: <**xs:union memberTypes="tipo1 tipo2"**>
- **Ej**: definir un nuevo elemento simple llamado **moneda** que sea el resultado de unir los tipos *integer* y *float*

```
<xs:element name="moneda">
<xs:simpleType>
  <xs:union memberTypes="integer float" />
</xs:simpleType>
</xs:element>
```

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**: **combinando restricciones**

  - Los diferentes tipos de derivación por restricción se pueden **combinar** entre sí en el ej se usa una deriv definida con un patrón, una lista y una unión
  - **Ej**: nuevo tipo de dato simple derivado llamado "**isbnTipo**" definido utilizando restricciones de patrón, valores enumerados y una unión

```
<xs:simpleType name="isbnTipo">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{10}"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="NAC"/>
        <xs:enumeration value="INT"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

a) **Por restricción**: **combinando restricciones**

  - **Ej2**: crea un nuevo tipo de dato simple derivado llamado "tipoNota" utilizando una unión entre los nº enteros 0 y 10 y el texto NP (de no presentado).
    - Comprueba su funcionamiento en un documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
- <notas xsi:noNamespaceSchemaLocation="tipoNota.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  - <modulo ciclo="Asir" curso="primero" nombreMod="LM">
      <alumno>Juan Alonso</alumno>
      <nota>7</nota>
    </modulo>
  - <modulo ciclo="Asir" curso="segundo" nombreMod="IAW">
      <alumno>Maria Sanz</alumno>
      <nota>NP</nota>
    </modulo>
</notas>
```

Tipos de datos derivados por restricción

- **Tipos de datos derivados**:

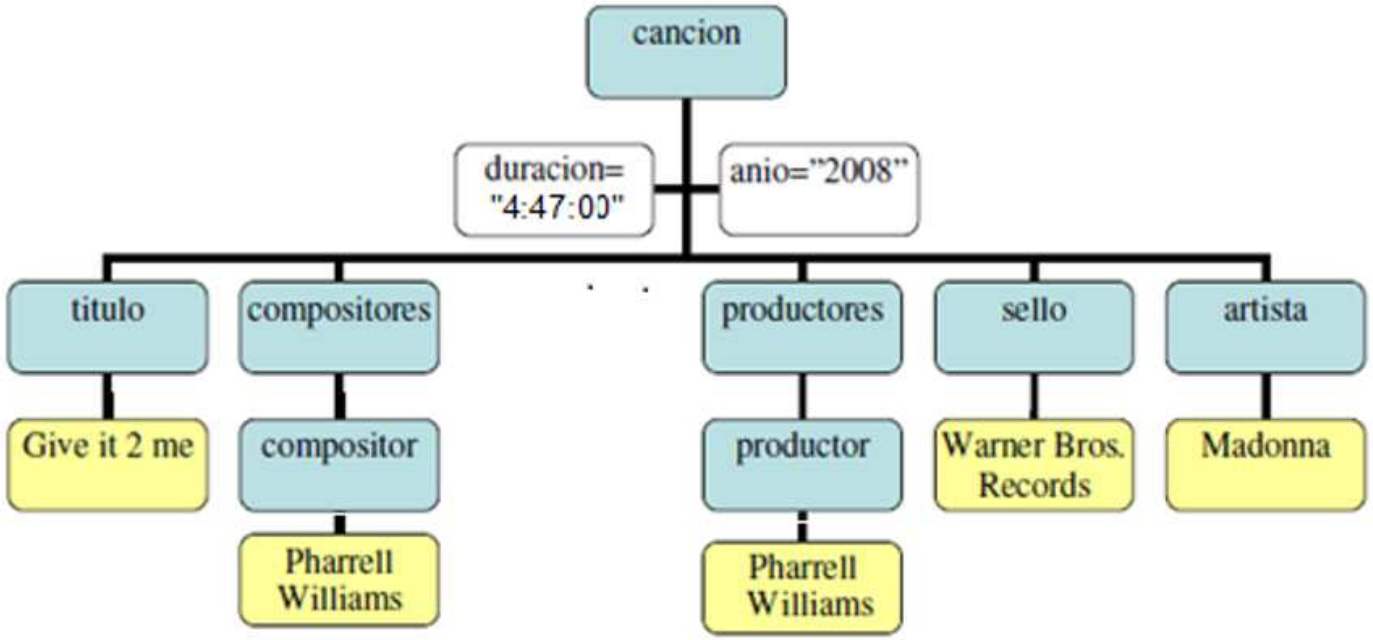
a) **Por restricción**: **combinando restricciones**

  - **Sol. Ej2**: crea un nuevo tipo de dato simple derivado llamado “tipoNota” definido utilizando una unión entre los nº enteros 0 y 10 y el texto NP (de no presentado).
  - Comprueba su funcionamiento en un documento XML

```
<xs:simpleType name="tipoNota">
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="xs:integer">
    <xs:maxInclusive value="1"/>
    <xs:minInclusive value="10"/>
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:enumeration value="NP"/>
   <xs:restriction>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

Ejercicios de XML Schema

- **Ejercicio3:** Construye el XML Schema que se ajuste a la siguiente jerarquía de datos. Valida un documento XML con contenido adecuado con dicho esquema.
  - Este esquema tendrá como elemento raíz **canción** que como elemento complejo tendrá una secuencia de elementos (*titulo*, *compositores*, *productores*, *sello* y *artista*) además de dos atributos (*duración* (tipo *time*) y *año* (tipo *entero*).
  - Recomendable definir de forma global los tipos de los elementos *compositores* y *productores*.



➢ Sol. Ej3:

```
<xs:element name="cancion">
 <xs:complexType>
 <xs:sequence>
  <xs:element name="titulo" type="xs:string" />
  <xs:element ref="compositores" />
  <xs:element ref="productores" />
  <xs:element name="sello" type="xs:string" />
  <xs:element name="artista" type="xs:string" />
 </xs:sequence>
 <xs:attribute name="duracion" type="xs:time" use="required" />
 <xs:attribute name="año" use="required">
  <xs:simpleType>
   <xs:restriction base="xs:integer">
    <xs:minInclusive value="1960" />
    <xs:maxInclusive value="2021" />
   </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="compositores">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="compositor" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<cancion xmlns:xsi="http://www.w3.org/2001/
         xsi:noNamespaceSchemaLocation="
         duracion="00:04:47" año="2000">
  <titulo>Give it 2 me</titulo>
  <compositores>
   <compositor>Pharrel Williams</compositor
   <compositor>Fredy Mercury</compositor>
  </compositores>
  <productores>
   <productor>Pharrel Williams </productor>
  </productores>
  <sello>Warner Bros. Record</sello>
  <artista>Madonna</artista>
</cancion>
```

Resumen: Formas de declarar elementos

- **Formas de declarar elementos**:

❖ **Forma local**: como **tipo anónimo**, introduciendo la declaración del **tipo dentro** de la definición del elemento.

❖ **Forma global**: se declara el **tipo** de modo **independiente** y después se declaran elementos utilizando ese tipo (**type**).

❖ **Empleando referencias**: se declara el **elemento** de modo **independiente** y después se declaran elementos utilizando como referencia dicho elemento (**ref**).

➢ **Deduce**: observa la siguientes declaración e indica a qué forma de declaración de elementos se refiere. Crea el ejemplo adecuado para cada uno de las otras formas

```
<xs:element name="enderezoPrincipal" type="tipoEnderezo"/>
… …
<xs:complexType name="tipoEnderezo">
   <xs:sequence>
      <xs:element name="rua" type="xs:string"/>
      <xs:element name="numero"  type="xs:unsignedShort"/>
      <xs:element name="localidade" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="provincia"  type="xs:NMTOKEN" fixed="Lugo"/>
 </xs:complexType>
```

Resumen: Formas de declarar elementos

- **Sol**:
- **Local**:

```
<xs:element name="enderezoPrincipal">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="rua" type="xs:string"/>
         <xs:element name="numero"  type="xs:unsignedShort"/>
         <xs:element name="localidade" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="provincia"  type="xs:NMTOKEN" fixed="Lugo"/>
   </xs:complexType>
 </xs:element>
```

- **Global**:

```
<xs:element name="enderezoPrincipal" type="tipoEnderezo"/>
… …
<xs:complexType name="tipoEnderezo">
   <xs:sequence>
      <xs:element name="rua" type="xs:string"/>
      <xs:element name="numero"  type="xs:unsignedShort"/>
      <xs:element name="localidade" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="provincia"  type="xs:NMTOKEN" fixed="Lugo"/>
 </xs:complexType>
```

- **Ref**:

```
<xs:element name="enderezos">
   <xs:sequence>
     <xs:element ref="Enderezo"  minOccurs="0"/>
   </xs:sequence>
</xs:element>
… …
<xs:element name="Enderezo">
  <xs:complexType>
   <xs:sequence>
      <xs:element name="rua" type="xs:string"/>
      <xs:element name="numero"  type="xs:unsignedShort"/>
      <xs:element name="localidade" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="provincia"  type="xd:NMTOKEN" fixed="Lugo"/>
 </xs:complexType>
</xs:element>
```