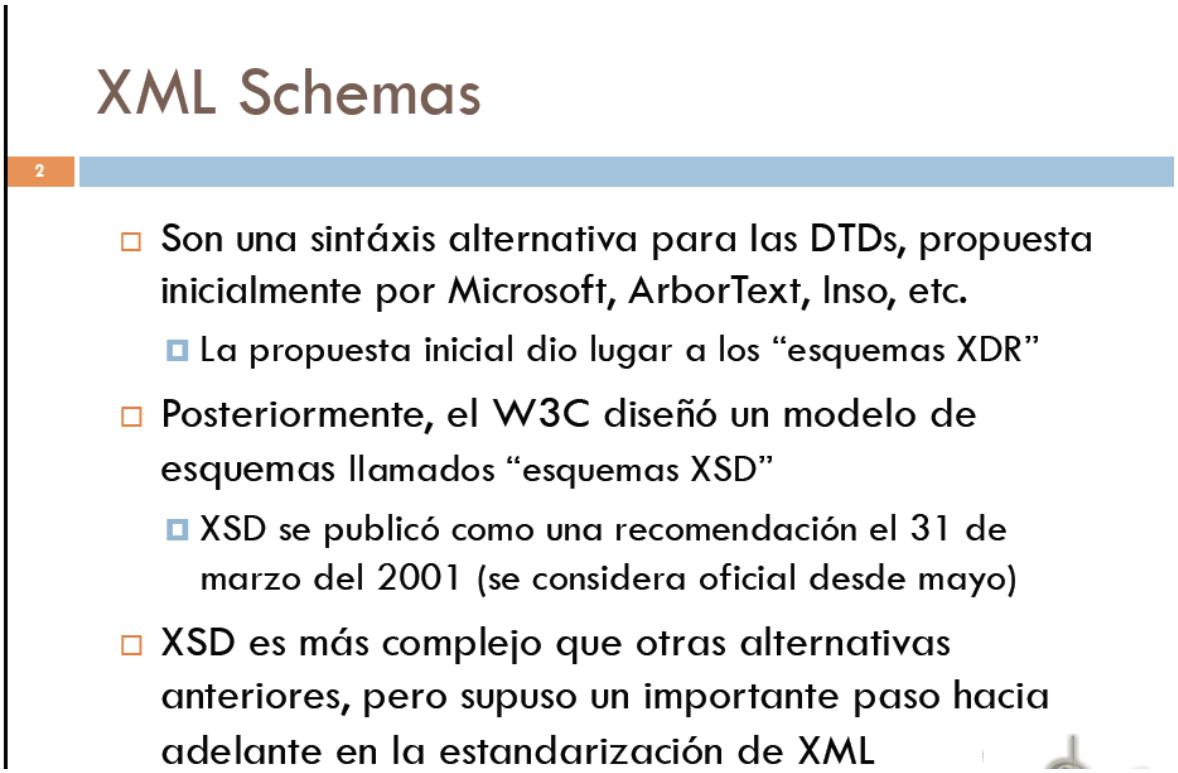
ESQUEMAS: XSD

• ESQUEMAS: CONCEPTOS GENERALES



Aunque el concepto de DTD forme parte de la Recomendación XML, hay que resaltar que una DTD se restringe a describir la estructura del documento, presentando importantes limitaciones a la hora de definir el contenido permitido de los elementos, para lo que se basa prácticamente en el uso de PCDATA. Un XSD permite expresar con mayor potencia, gramáticas más complejas utilizando la misma sintaxis de XML.

Para comprobar la necesidad de llegar a concretar más la estructura de un documento, de lo que se hace una DTD, supóngase que en un documento relacionado con Geografía se quiere usar el elemento localización y que por razones fáciles de entender, se requiere que para que sea válido deba cumplir condiciones como las siguientes :

1. Estar compuesto por : longitud, latitud y un error asociado a la medida.

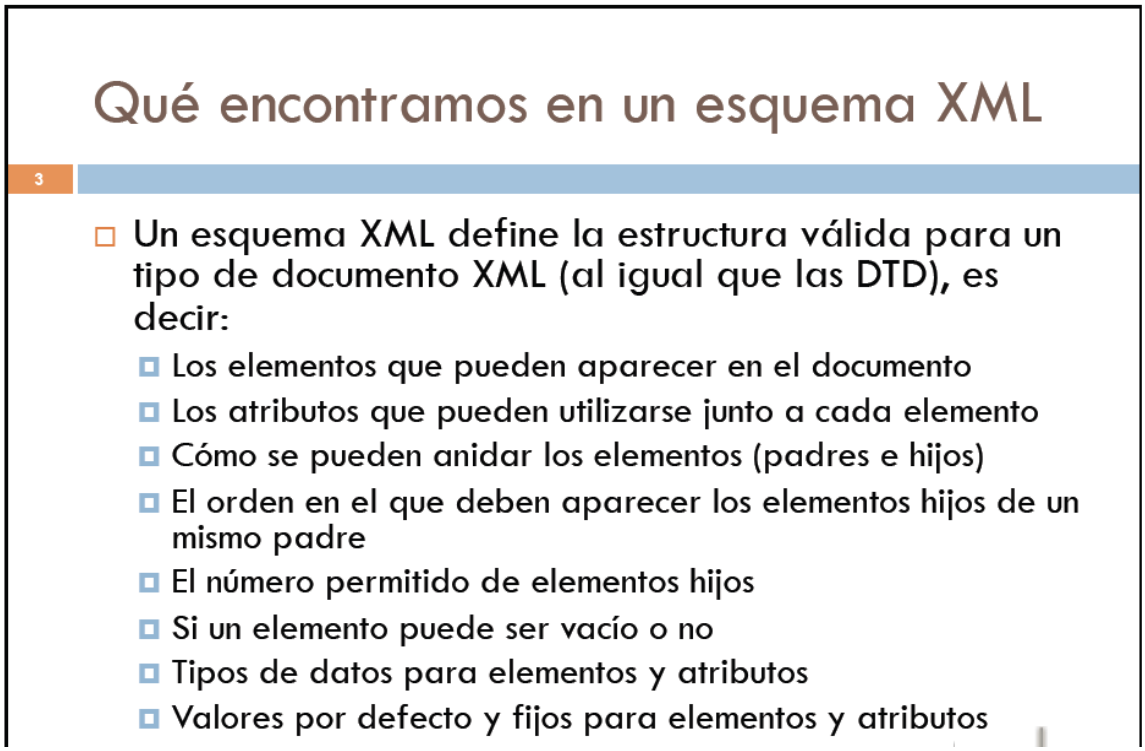
2. Los elementos latitud y longitud deben tener seis cifras decimales y unos rangos respectivos de, -90 a +90, y -180 a + 180.

3. El error en la localización debe ser un entero no negativo.

Si se consiguen especificar estas condiciones, los valores de los componentes de localización podrán ser tales como:

```
<localización>
        <latitud>32.904237</latitud>
         <longitud>73.620290</longitud>
        <incertidumbre unidad=”metros”>2</incertidumbre>
</localización>
```

Para asegurar que las restricciones se cumplen, es importante contar con un mecanismo que asegure la validez de los correspondientes valores, algo que una DTD no puede hacer. Por ello y con el objeto de superar estas carencias surgió la idea de generalizar las DTDs utilizando la propia sintaxis XML a la hora de definir y validar las características de un documento. Los Esquemas XML son la alternativa seguida actual haya otras opciones (como RelaxNG) y sobre la que trabajan los desarrolladores actuales, lo que significa que en estos momentos son multitud los Esquemas XML utilizados en los más diversos campos.



Los esquemas XML tienen mayor poder para definir un documento XML válido. Las ventajas que presentan son:

Ventajas de los Esquemas XML

6

- Los esquemas son preferibles a los DTD's porque:
  - Soportan tipos de datos
  - Están escritos en XML
  - Son extensibles a futuras modificaciones
  - Soportan espacios de nombres
  - Son más ricos y potentes que los DTDs

- **Soportan Tipos de Datos**
- Con soporte para tipos de datos:
  - Es más fácil describir el contenido permitido de un documento.
  - Es más fácil validar la corrección de los datos.
  - Es más fácil trabajar con datos desde una base de datos.
  - Es más fácil definir facetas (restricciones sobre los datos)
  - Es más fácil definir patrones (formato sobre los datos)
  - Es más fácil realizar conversiones sobre diferentes tipos de datos.

- **Están escritos en XML**
- Beneficios de los esquemas como documentos XML
  - No es necesario aprender un nuevo lenguaje
  - Se pueden utilizar editores XML para editar los esquemas
  - Se pueden utilizar analizadores XML para analizar los esquemas
  - Los esquemas se pueden manipular haciendo uso del DOM XML
  - Los esquemas se pueden transformar haciendo uso de XSLT

- **Los esquemas son extensibles**
- Los esquemas XML son extensibles porque XML lo es. Con una definición de esquema extensible podemos:
  - Reutilizar un esquema para la definición de otros esquemas.
  - Crear tipos de datos propios a derivados a partir de los tipos estándar.
  - Referenciar múltiples esquemas desde un mimo documento.

• ESTRUCTURA DE UN ESQUEMA XML. Componentes

Referencia a un esquema

10

- La referencia a un DTD desde un documento XML se incluye *delante* del elemento raíz:

```
<?xml version="1.0"?>
<!DOCTYPE rootElement SYSTEM "url">
<rootElement> ... </rootElement>
```

- La referencia a un Esquema desde un documento XML se incluye en el elemento raíz:

```
<?xml version="1.0"?>
<rootElement
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            (se requiere la referencia al XML Schema Instance)
   xsi:noNamespaceSchemaLocation="url.xsd">
            (localización dónde el Esquema XML Schema reside)
   ...
</rootElement>
```

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

Etiqueta Schema

La etiqueta **schema** identifica **la raíz** de un documento XML Schema. En esta etiqueta se declara el espacio de nombres estándar que utilizan los esquemas (y que permite diferenciar las etiquetas XML del esquema, respecto a las del documento XML), el cual se puede definir

como el espacio de nombres por defecto, definir un prefijo **xs** para él (es la forma habitual) o bien definir un prefijo **xsd**. Es decir estas tres posibilidades:

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

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

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

A partir de ahí las etiquetas pertenecientes a XML Schema, usarán el prefijo indicado en su espacio de nombres (normalmente **xs**).
Además en la misma etiqueta se define el espacio de nombres al que se aplica el esquema. Es decir, 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 se puede declarar como espacio de nombres por defecto (es lo habitual) o usar un prefijo; incluso se pueden indicar varios espacios de nombres (sólo uno podrá ser definido por defecto como mucho) a los que aplicar el esquema. Como siempre la etiqueta que declara el espacio de nombres es **xmlns**.
Además el atributo **targetNamespace** permite indicar el espacio de nombres sobre el que se aplica el esquema (si se aplica a varios espacios de nombres, aparecerán separados con espacios), es decir a qué documentos se aplicará el esquema.

Ejemplos de etiquetas schema:

<xs:schema

xmlns:xs=”http://www.w3.org/2001/XMLSchema“

xmlns=”http://www.jor.net/doc”

targetNamespace="http://www.jor.net/doc">

En el ejemplo anterior, las etiquetas correspondientes al espacio estándar de XML Schema usarán el prefijo **xs**, mientras que las etiquetas pertenecientes a los documentos XML correspondientes al espacio privado ***jor.net/doc*** usarán el espacio de nombres por defecto.

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:doc="http://www.jor.net/doc"
targetNamespace="http://www.jor.net/doc">

En este caso XMLSchema usarán el prefijo **xs**, mientras que las etiquetas definidas en el esquema usarán el prefijo **doc** ya asociado a su espacio por defecto:

<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:doc="http://www.jor.net/doc"
xmlns:img=”http://www.jor.net/img”
targetNamespace=”http://www.jor.net/doc”
"http://www.jor.net/img ">

En este caso las etiquetas XMLSchema usarán el espacio por defecto de nombres; mientras que se definen dos espacios de nombres asociados a los prefijos ***doc*** e ***img***, ambos también indicados en el **targetNamespace**; eso indica que el esquema se aplicará a documentos pertenecientes a ambos espacios de nombres.
La razón de repetir la URI de los espacios de nombres reside en que los atributos **xmlns** y **targetNamespace** no sirven para lo mismo; **el primero declara espacios de nombre y el**

segundo sirve para indicar a qué tipo de documentos se aplicarán las reglas del esquema. Aunque en la práctica ciertamente el contenido de ambos atributos es el mismo.

Asociar un esquema a un documento XML

Para que un documento XML siga las reglas definidas en un esquema, no disponemos de etiqueta **!DOCTYPE**; en su lugar utilizamos atributos especiales en el elemento raíz del documento XML.
Primero, al igual que en el documento XMLSchema, necesitamos definir los dos espacios de nombres, el correspondiente al documento XML (que se suele usar sin abreviatura, es decir como espacio por defecto) y el espacio de nombres de XML Schema (que suele utilizar el prefijo **xs**, aunque se puede utilizar otro).

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

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns="http://www.jor.net/doc"
xmlns:xs="http://w3.org/2001/XMLSchema-instance"
xs:schemaLocation="esquema.xsd">
....
</documento>
```

Se indica el espacio por defecto de nombres en el documento (coincide con el declarado en el propio archivo del esquema), se indica el espacio de nombres correspondiente al esquema (siempre es la misma dirección de Internet) y se asocia a este espacio el prefijo **xs** (se puede elegir otro prefijo, pero no es nada conveniente).

Documento XML Schema . esquema.xsd

Pertenece al espacio público estándar http://www.w3.org/2001/XMLSchema

```
<? xml version=”1.0" ?>

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

                xmlns=”http://jor.net/doc”

                targetNamespace=” http://jor.net/doc“ >

...código XMLSchema...

</schema>
```

Documento XML esquema.xsd del documento doc.xml

Pertenece al espacio privado http://jor.net/doc

```
<? xml version=”1.0" ?>

<raíz xmlns:xs= ”http://www.w3.org/2001/XMLSchema-instance”
```

xmlns=”http://jor.net/doc”

xs:schemaLocation=””esquema.xsd”>

...código que cumple las reglas del esquema..

</raíz>

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 extension xsd)

Se pueden indicar varios esquemas de validación, por lo que habría que indicar a qué espacio se aplica cada uno:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns:doc="http://www.jor.net/doc"
xmlns:img="http://www.jor.net/img"
xmlns:xs="http://w3.org/2001/XMLSchema-instance"
xs:schemaLocation="http://www.jor.net/doc esquemaDoc.xsd”
                  “http://www.jor.net/img esquemaImg.xsd2
....
</documento>
```

La dirección del documento se ha puesto en estilo de ruta relativa, que se calcula a partir de la dirección del documento XML (es decir el documento XMLSchema en el ejemplo estará en la misma carpeta del XML). Pero lo habitual es que a los esquemas se acceda por URL completa; por ejemplo ***http://www.obj.com/esq.xsd***
Además podemos indicar un esquema para un documento, pero sin que dicho esquema utilice espacio de nombres. Por ejemplo el esquema podría tener esta cabecera (archivo ***esquema1.xsd***):

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

El archivo XML Schema anterior no indica ningún espacio de nombres al que aplicarse. Un documento que hiciera referencia al esquema podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<descripción xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="esquema1.xsd">
</descripción>
```

El atributo **noNameSchemaLocation** permite indicar un esquema para el documento sin que éste utilice espacio de nombres alguno (lo cual no es nada aconsejable, pero vale para hacer pruebas).

EJEMPLO_1: PELI.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="peli.xsd">
        THE TOURIST
</pelicula>
```

Peli.xsd

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

EJEMPLO_2

Por ejemplo, un schema nos permite definir el tipo del contenido de un elemento o de un atributo, y especificar si debe ser un número entero, o una cadena de texto, o una fecha, etc. Los DTDs no nos permiten hacer estas cosas.

Veamos un ejemplo de un documento bien formado, pero que no está siendo validado ni con DTD ni con XSchema:

```
<?xml version = "1.0" encoding = "UTF-8"?>

<vehiculos>

    <nombre>coche</nombre>

    <nombre>moto</nombre>

    <nombre>carro</nombre>

</vehiculos>
```

Para validar ese documento, necesitaremos dos cosas: un esquema (que debe estar en un **fichero .xsd** diferente) **referenciar a ese esquema dentro del documento XML**

El documento XML para el ejemplo de los vehiculos: **vehiculos.xml**

```
<?xml version = "1.0" encoding = "UTF-8"?>

<vehiculos

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

  xsi:noNamespaceSchemaLocation = "vehiculos.xsd"   >

    <nombre>coche</nombre>

    <nombre>moto</nombre>

    <nombre>carro</nombre>

</vehiculos>
```

El esquema para el ejemplo de los vehiculos: vehiculos.xsd

```
<?xml version = "1.0" encoding = "UTF-8"?>

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

   <xsd:element name = "vehiculos">

      <xsd:complexType>

         <xsd:sequence>

            <xsd:element name = "nombre" type = "xsd:string" maxOccurs = "unbounded"/>

         </xsd:sequence>

      </xsd:complexType>

   </xsd:element>

</xsd:schema>
```

Aclaremos algunas cosas que podemos ver en ese ejemplo. En cuanto a la declaración del espacio de nombres en el fichero XML:

La línea xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" indica que queremos utilizar los elementos definidos en http://www.w3.org/2001/XMLSchema-instance
La línea xsi:noNamespaceSchemaLocation = "vehiculos.xsd" indica que vamos a usar ese fichero que contiene el XSChema, pero sin asociar un espacio de nombres a esas definiciones. Sin esta línea, no tendremos esquema de validación.

En cuanto a la definición del XSchema:
La línea xmlns:xsd = "http://www.w3.org/2001/XMLSchema" indica que todos los elementos del XSchema se nombrarán con el prefijo xsd:.

EJEMPLO_3. PEOPLE.XML

```
<p:persona
    xmlns:p="http://www.prueba.es/persona"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation= "http://www.prueba.es/persona people.xsd"
    >
   <p:nombre>John</p:nombre>
   <p:direccion>John</p:direccion>
</p:persona>
```

PEOPLE.XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.prueba.es/persona"
elementFormDefault="qualified"
```

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

Partes de un esquema

Elementos, definidos con etiquetas **xs:element**. Para indicar los elementos permitidos en los documentos que sigan el esquema.

Atributos, etiqueta **xs:attribute**.

Tipos simples, que permiten definir los tipos simples de datos que podrá utilizar el documento XML. Lo hace la etiqueta **xs:simpleType**.

Tipos complejos, mediante la etiqueta **xs:complexType**.

Documentación, información utilizable por aplicaciones que manejen los esquemas. Etiquetas **xs:annotation**, **xs:documentation** y **xs:appInfo**.

Componentes locales y globales

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

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

Ejemplo:

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

…
</xs:simpleType>
…
</xs:element >
<xs:simpleType …> …
</xs:simpleType …>
</xs:schema>

El componente local definido sólo se podría utilizar en la zona resaltada. Si fuera global se podría utilizar en todo el documento.

ESQUEMAS: ESTRUCTURAS DE DATOS

• ELEMENTOS Y ATRIBUTOS EN ESQUEMAS

A. EL ELEMENTO ELEMENT

Para declarar elementos se utiliza element. Los elementos tienen al menos un nombre y un tipo de dato debidamente definidos.

- Cada ELEMENT de la DTD pasa a ser un elemento element en el Esquema:

Los elementos que van a aparecer en un documento instancia deben aparecer en el Esquema con un elemento **element,** cuyos atributos son:

```
<xs:element
name="nombre del elemento"
type="tipo global de datos"
ref="declaración del elemento global"
id="identificador"
form="cualificación" <!--qualified o unqualified →
minOccurs="número mínimo de veces"
maxOccurs="máximo número de veces"
default="valor por defecto"
fixed="valor fijo"
>
```

- **maxOccurs** y **minOccurs** : determinan el número máximo y mínimo de veces que el elemento puede aparecer en un documento instancia; obviamente, sólo se aplican a atributos de elementos de tipo real.
- **name** : especifica el nombre usado para referenciar un tipo de elemento tanto en el resto del Esquema como en un documento instancia.
- **ref** : hace referencia a un elemento global ya declarado. Es mutuamente excluyente con name.

```
<xsd:element name=”Nombre” type=”xsd:string”/>

<xsd:element ref="Nombre"/>
```

De esta forma sabemos que este elemento hace referencia al que tiene como name "Nombre".

- **type** : especifica el tipo de dato del elemento, con un valor nombre referido a un tipo global, simple o complejo.

Al menos hay que indicar el nombre; el tipo de datos , será necesario indicarle el resto de atributos sólo si se necesitan. Por ejemplo se puede definir un elemento como:

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

El elemento descripción será de tipo string. Los tipos de datos en XML Schema son muchos y además permiten personalizar sus posibilidades para adaptarles a cualquier necesidad.

A.1 ELEMENTOS SIMPLES

Un **elemento simple** es un elemento que sólo puede contener texto (cualquier tipo de dato o información), pero **NO PUEDE CONTENER otros elementos, ni atributos, ni puede estar vacio.**

<xsd:element name="nombre" type="tipo"/>

<xsd:element name=”Nombre” type=”xsd:string”/>

Ejemplos:

<xsd:element name=“apellido” type="xsd:string"/>
<xsd:element name=“edad" type="xsd:integer"/>
<xsd:element name=“fecNac" type="xsd:date"/>

<xsd:element/> tiene como posibles ATRIBUTOS:

- **name**=” ” Identificador.
- **type**=” ” tipo de dato.
- **ref:** hace referencia a un elemento global ya declarado. **<xsd:element ref="Nombre"/>**
- **maxOccurs** y **minOccurs:** determinan el número máximo (mínimo) de veces que el elemento puede aparecer en un documento instancia -XML-(**unbounded es** sin Especificar límite)

<xsd:element name=”asignatura” type=”xsd:string” maxOccurs=“5” minOccurs=“2”/>
<xsd:element name=“alumno” type=“xsd:string” minOccurs=“1” maxOccurs=“unbounded”/>

Un elemento simple puede tener un valor por defecto y un valor “fijo”. Esto se indica mediante los atributos **default y fixed**

<xsd:element name="color" type="xsd:string" default="red"/>
<xsd:element name="color" type="xsd:string" fixed="red"/>

A.2 ELEMENTOS COMPUESTOS

- Un elemento complejo puede ser de 4 TIPOS:
  1. Elementos vacíos
  2. Elementos no vacíos con atributos
  3. Elementos con elementos hijos (SEQUENCE, CHOICE, ALL)
  4. Elementos con elementos hijos y con “texto” o valor propio (como el contenido mixto de las DTD con #PCDATA).

- **Un elemento que tiene atributos es de tipo complejo.**
- La declaración es la siguiente:

```
<xsd:element name="Libro">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="Título" type="xsd:string"/>
   <xsd:element name="Autores" type="xsd:string" />
  <xsd:element name="Editorial" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="precio"
  type="xsd:double"/>
</xsd:complexType>
</xsd:element>
```

Para el xml:

```
<libro precio="">
        <Título/>
        <Autores/>
        <Editorial/>
</libro>
```

B. EL ELEMENTO ATTRIBUTE

Los atributos son complementos de información que se puedan asignar a un elemento previamente declarado. En un Esquema XML los atributos se declaran por medio del elemento attribute que a su vez tiene una serie de atributos que ponen restricciones a las propiedades de esta declaración, cuya panorámica depende de su ubicación dentro del Esquema (como se sabe un atributo global debe declararse como hijo del elemento schema).

- Los atributos se deben declarar de forma similar a los "elementos simples"
- Si un elemento puede ir acompañado de atributos, el elemento se deberá declarar como un elemento "complejo", ya no es simple.
- Un atributo se declara de la siguiente forma:

  <xsd:attribute name="nombre" type="tipo" default=""/>

- Una vez que se define un elemento o atributo haciendo uso de **name, nos podemos referir a él haciendo uso de ref.**

Ejemplo:

<xsd:attribute name="idioma" type="xsd:string" default="EN"/>

- Los atributos tienen un tipo de dato: xsd:string, xsd:decimal, xsd:integer, xsd:boolean, xsd:date, xsd:time
- <xsd:attribute> Puede ser **global** (Utilizado por cualquier elemento) **o local**
  - **Si es global:**

    <xsd:schema>
    <xsd:attribute name="miAtributo"/>
    <xsd:element name="algunElemento" type="xs:string"/>
    </xsd:schema>

- **Si es local se coloca como hijo del elemento:**

```
<xsd:schema>
<xsd:element name=”algunElemento”>
<xsd:complexType>
        <xsd:attribute name=”miAtributo”/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

- Los atributos pueden tener valores “por defecto” y valores “fijos”. Atributos **default y fixed.**
- Por defecto los atributos son **opcionale**s
- Un atributo específico del elemento attribute es **use,** que da la posibilidad de limitar el uso del atributo en un documento instancia, pudiendo tomar los valores :
  - **optional, prohibited, y required**, cuyos significados respectivos son evidentes.

```
<xsd:attribute name="lang" type="xsd:string" use="required"/>
```

Señalar que **en los atributos como norma se aplica el valor por defecto cuando los atributos no quedan determinados.**

- El atribute use por defecto es optional.
- El atributo use puede tomar el **valor “” si el atributo no es obligatorio.**

EJEMPLO_1 PELI CON ATRIBUTOS PELI.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula  titulo="the tourist" minutos="120"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "pel_atributos.xsd">
</pelicula>
```

Pel_atributos.xsd

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

EJEMPLO_2 :

El elemento raíz se llama libro y tiene tres hijos y un atributo. Los hijos son *Título, Editorial* que deben aparecer una vez y **Autores** que pueden aparecer de una a diez veces. El hecho de que estén agrupados en una secuencia indica que los elementos deben aparecer en orden, es decir, primero el ***Título***, luego los ***Autores*** y por último la ***Editorial***. Los tres elementos son de **tipo string**. El atributo de libro se llama ***precio*** y es de tipo **double**.

Un ejemplo sencillo es el siguiente (libro.xsd):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Libro">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Título" type="xsd:string"/>
<xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
<xsd:element name="Editorial" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="precio" type="xsd:double"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Un ejemplo de fichero XML asignado al XML Schema es el siguiente (libro.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<Libro xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" libro.xsd" precio="20">
<Título>Fundamentos de XML Schema</Título>
<Autores>Allen Wyke</Autores>
<Autores>Andrew Watt</Autores>
<Editorial>Wiley</Editorial>
</Libro>
```

• TIPOS SIMPLES DE DATOS

Utilizando sólo DTD no se tienen predefinidos los tipos de dato (sólo PCDATA). Los Esquemas han tenido que solucionar esta carencia.

Para definir la estructura y tipos de datos que soporta documento instancia, se usan los elementos:

- simpleType (contiene sólo datos y ningún elemento hijo) y
- complexType (contiene además elementos hijo y atributos).

elementos “simples”

19

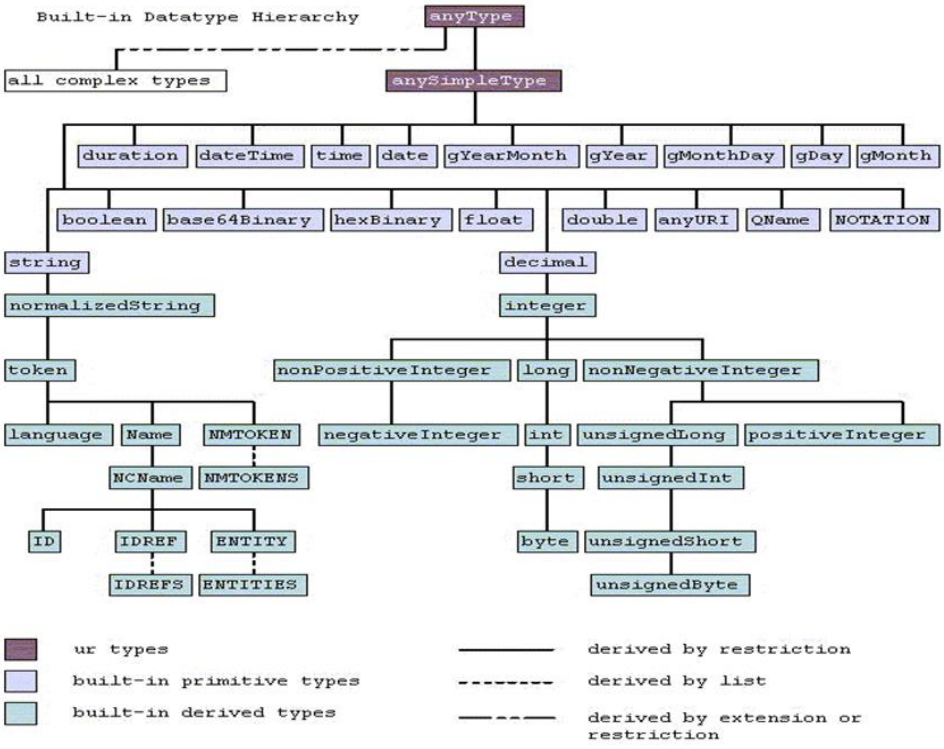
- Para definir un elemento simple, utilizamos las sintaxis siguientes:

```
<xsd:element name="nombre" type="tipo"/>

<xsd:element name="nombre">
  <xsd:simpleType>
    ... Información sobre el tipo simple...
  </xsd:simpleType>
</xsd:element>
```

Un tipo **de dato simple puede** ser:

- **Predefinido o primitivos**: Los tipos más básicos de XML, están ya definidos por el propio lenguaje XML. (incorporado al Espacio de Nombres de los Esquemas).
- **Derivado:** Tipos de datos más complejos creados a partir de los anteriores (definido por el autor del Esquema).



A. PREDEFINIDOS O PRIMITIVOS

Entre los predefinidos podemos distinguir entre:

- Tipos de datos simples primitivos
- Predefinidos derivados

I. DATOS SIMPLES PRIMITIVOS

Son los tipos básicos de XML. Sirven para formar los tipos derivados y tipos más complejos. Para usarlos basta indicarlos en el atributo **type** de una etiqueta **element** o **attribute** (que son las que permiten crear elementos y atributos). De esta forma:

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

Él código remarcado es el que indica que en los documentos XML basados en esta plantilla XMLSchema, habrá un elemento llamado ***descripción*** que contendrá datos de tipo ***string***. El hecho de que se use el prefijo **xs** (es decir ***xs:string***) es para indicar que es un tipo XML y por lo tanto usa el prefijo designado al espacio de nombres de XMLSchema. Los tipos básicos son:

TIPO	EJEMPLO
String	"Hola Mundo" Representan textos con cualquier contenido excepto los símbolos <, > & y las comillas para los que se usará la entidad correspondiente.
Boolean	{true,false,1.0} Sólo puede contener los valores verdadero o falso, escritos como **true** o **false**, o como **1** (verdadero) ó **0** (falso)
Integer	Basado en **number**. Representa números enteros tanto positivos como negativos
Decimal	Representa números decimales en coma fija. Ocupan más internamente pero se representan de forma exacta. No admite los valores **INF**, **NaN** ni tampoco el formato científico.
Float	12.56E3, 12, 12560, 0, -0, INF, -INF, NUM . Permite utilizar números decimales en formato de coma flotante de precisión simple. El separador decimal es el punto. Ejemplos de valores: ***1, 2 3.4, -123,-124.76, 1E+3*** (significa 1000), ***1.45E+4*** (significa 14500), ***1.45E-4*** (significa 0,000145, es decir 1,45 · 10-4), ***INF*** (infinito), ***-INF*** (menos infinito), ***NaN*** (significa ***Not a Number***, no es número, para números no válidos)
Double	12.56E3, 12, 12560, 0, -0, INF, -INF, NUM. Números decimales en formato de coma flotante de precisión doble. Es decir el mismo tipo de números pero con mejor precisión.
Duration	Representa duraciones de tiempo en formato **ISO 8601** (sección 5.3) **P**n**Y**n**M**n**DT**hms Donde todos los signos ***n*** son números. Ejemplos de valores: • ***P1Y*** significa ***Un año***. • ***P1Y2M*** significa ***un año y dos meses.*** • ***P1Y2M3D*** significa ***un año y dos meses y tres días*** • ***P3D*** significa ***tres días*** • ***P1Y2M3DT12H30M40.5S*** significa ***un año, dos meses, tres días, doce horas treinta minutos y cuarenta segundosy medio***

	• ***PT12H30M40.5S doce horas treinta minutos y cuarenta segundosy medio*** Como se observa la **P** es obligatoria y la **T** sirve para separar los valores de fecha de los valores hora.
DateTime	CCYY-MM-DDThh:mm:ss . Representa fechas según el formato **ISO 8601** (sección 5.4). El formato es **yyyy-mm-ddThh:mm:ss** , por ejemplo ***1998-07-12T16:30:00.000*** (12 de julio de 1998 a las 16:30). La **T** es obligatoria para separar la fecha de la hora.
Time	Representa horas en el formato ***hh:mm:ss***
Date	Representa fecha en formato ***yyyy-mm-dd***
gYearMonth	Un mes y año de calendario gregoriano. Representa un mes y un año en formato ***yyyy-mm***
gYear	Representa un año usando cuatro cifras.
gMonthDay	Día y mes del calendario gregoriano formato ***--mm-dd***
gMonth	Mes del calendario gregoriano
gDay	Una fecha del calendario gregoriano(día)
NonPositiveInteger	desde negativo infinito a cero
NegativeInteger	desde negativo infinito a –1
Long	-9223372036854775808 a 9223372036854775807
Int	-2147483648 a 2147483647
Short	-32768 a 32767
Byte	-127 a 128
NonNegativeInteger	De cero a infinito
UnsignedLong	0 a 18446744073709551615
UnsignedInt	0 a 4294967295
UnsignedShort	0 a 65535
UnsignedByte	0 a 255
PositiveInteger	1 a infinito

Recordemos que **un elemento simple puede tener un valor “por defecto” y un “valor fijo” utilizando atributos default y fixed**.

Ejemplos de declaración de elementos usando tipos primitivos son :

- <xs:element name=”temperatura” type=”xs:decimal”/>
  Donde el elemento temperatura tiene un valor decimal; por ejemplo:
  <temperatura>98.6</temperatura>
- <xs:element name=”fiesta” type=”xs:date”/>
  Donde el elemento fiesta tiene un valor CCYY-MM-DD; por ejemplo:
  <fiesta>2004-04-12</fiesta>
- <xs:element name=”pagado” type=”xs:boolean”/>
  <pagado>true</pagado>

II. PREDEFINIDOS DERIVADOS SUBTIPO DEL TIPO DE DATO PRIMITIVO

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

TIPO DERIVADO DE STRING	
NormalizedString	Una cadena sin tabuladores, sangrías, o retornos de carro.
Token	Cadena sin/con tabulaciones, espacios consecutivos, etc.
Language	Cualquier valor xml: lang válido, por ejemplo : EN, FR,...
IDREFS	debe usarse sólo con atributos
ENTITIES	debe usarse sólo con atributos
NMTOKEN	debe usarse sólo con atributos
NMTOKENS	debe usarse sólo con atributos
Name	NCName part (ningún espacio de nombres calificador)
ID	debe usarse sólo con atributos
IDREF	debe usarse sólo con atributos
ENTITY	debe usarse sólo con atributos
anyURI	Una URI estándar de Internet
NOTATION	Declara en laces a contenidos externos no-XML
QName	Cadena, nombre con cualificación

Como ejemplo de uso de datos predefinidos, podemos citar su uso para manejar sucesiones de ceros y unos; por ejemplo:
<xsd:element name=”cero” type=”xsd:unsignedByte” fixed=”0”/>

o el uso de enteros positivos, por ejemplo :

<element name=”codigopostal” type=”positiveInteger”/>

Es el momento de recalcar que un tipo de dato en Esquemas se dice que es **atómico** si es indivisible como tal dato. Por ejemplo, dado el valor de ES para España definido como NMTOKEN (tipo de dato predefinido derivado) en un elemento Estado ninguna parte del mismo, como sería el carácter “S”, tiene ningún significado por sí mismo.

B. DEFINICIÓN Y DERIVACIÓN DE DATOS SIMPLES

A partir de los datos simples existe la posibilidad de obtener nuevos tipos de datos simples de dos formas:

- por **definición** (listas y unión) o
- por **derivación** (por **restricción** a partir de un tipo base utilizando las facetas correspondiente).

Para definir y derivar nuevos datos simples se usa el elemento **simpleType** que constituye la representación de un tipo simple en un documento, identificándolo con un nombre.

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

Los contenidos posibles de este elemento son los elementos: List, Union, y Restriction.

Definir tipos por unión

Se trata de utilizar dentro del tipo de datos una etiqueta llamada **union** que permite unir las definiciones de dos tipos de datos.
Por ejemplo:

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

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

Definir tipos simples por lista

Las listas permiten que un componente tenga como contenido una determinada lista de valores. la construcción de listas indica dos pasos:

(1) Crear un tipo simple de datos cuyo contenido es una etiqueta ***list***, la cual posee el atributo **itemType** para indicar el tipo de elementos de la lista. Simplemente con ello e podría establecer una lista (valores separados por espacios) de valores pertenecientes al tipo indicado.

(2) Crear el tipo ya definitivo de datos que contendrá una etiqueta **restriction** a la cual como tipo base se indica el tipo simple de datos relacionado con la lista. En la restricción se pueden indicar estas etiquetas usando el atributo **value**:

length. Indica que la lista tendrá un número exacto de valores.

minLength. Indica que la lista tendrá un número mínimo de valores

maxLength. Indica que la lista tendrá un número máximo de valores

enumeration. Posibles valores que puede tener la lista

whiteSpace. Gestión de los espacios en blanco en cada elemento de la lista.

pattern. Expresión regular que debe cumplir cada elemento de la lista.

Ejemplo:

```
<xs:simpleType name="listaDecimales">
        <xs:list itemType="xs:decimal" />
</xs:simpleType>
 <xs:simpleType name="listaNotas">
        <xs:restriction base=" listaDecimales">
                <xs:minLength value="3" />
                <xs:maxLength value="6" />
        </xs:restriction>
 </xs:simpleType>
```

Primero se define un tipo de lista (***listaDecimales***) simplemente indicando que será una lista cuyos valores serán números decimales. Después se concreta la lista (***listaNotas***), de modo que ahora se indica que la lista constará de tres a seis números (en el XML cada número irá separado por espacios)

Establecer tipos simples por restricción

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

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

El atributo **base** sirve para indicar en qué tipo nos basamos al definir la restricción (es decir de qué tipo estamos creando este derivado). El apartado restriction puede tener numerosas etiquetas que permiten establecer las restricciones deseadas al tipo.
Las etiquetas interiores a **restriction** disponen de un atributo llamado **fixed** que sólo puede valer verdadero (**true**) o falso (**false**). En caso de que sea verdadero, ningún tipo derivado del definido puede modificar la propiedad establecida; es decir, si establecemos **minLength** (tamaño mínimo) con valor ***ocho*** (propiedad **value**) y **fixed="true"**, ningún tipo derivado del definido podrá definir que el tamaño mínimo sea inferior a 8 caracteres. Es un atributo de uso opcional.

Las posibles restricciones que se pueden establecer son:

- **Tamaños de texto**. Indica tamaños máximos y mínimos que debe de tener el texto. Ejemplo:
  - **minLength**. Indica el mínimo número de caracteres. Eso lo hace mediante el atributo **value,** en el que se indica un número con el tamaño mínimo que deseamos.
  - **maxLength**. Indica un tamaño máximo de caracteres o de dígitos numéricos. Usa el mismo atributo **value**.
  - **length**. Indica un tamaño fijo de caracteres para el tipo. Es decir si indicados ***length*** con ***value="9"*** el texto deberá tener exactamente nueve caracteres.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:doc="http://www.jor.net/doc"
targetNamespace="http://www.jor.net/doc">
<xs:simpleType name="nombresTipo">
        <xs:restriction base="xs:normalizedString">
                <xs:maxLength value="15" />
                <xs:minLength value="4" />
        </xs:restriction>
</xs:simpleType>
<xs:element name="título" type="doc:nombresTipo" />
```

En el ejemplo, se define el tipo persona ***nombresTipo*** que permite textos entre 4 y 15 caracteres. El hecho de declararlo como derivado de **normalizedString** permite restringir que no considere en los textos más de un espacio seguido ni los tabuladores

ni saltos de línea (de otro modo casi siempre se superaría el mínimo de cuatro, quizá incluso fuera mejor derivar de ***NCNames*** que es aún más restrictivo).

- **Dígitos máximos**. Parecido al anterior pero trabajando con números. Indica las posibles cifras que puede tener el número.
  - **totalDigits**. Número máximo de dígitos del número, incluyendo los decimales. El atributo value indica el número máximo deseado
  - **fractionDigits.** Máximo número de decimales que puede tener el número.

```
<xs:simpleType name="tipo1">
        <xs:restriction base="xs:decimal">
                <xs:totalDigits value="6" />
                <xs:fractionDigits value="2" />
        </xs:restriction>
</xs:simpleType>
 <xs:element name="para" type="tipo1" />
```

En el ejemplo, los elementos que utilicen el ***tipo1*** definido, podrán escribiré números de hasta seis cifras.

- **Máximos y mínimos numéricos**. Restringe valores numéricos asignando topes a los mismos. Sirve para números y para valores de tiempo o duración. Se hace con:
  - **minExclusive**. Establece un valor mínimo. El valor debe ser mayor que el establecido por la etiqueta a través del atributo **value**.
  - **maxExclusive**. Establece un valor máximo. El valor debe ser menor que el establecido por la etiqueta a través del atributo **value**.

EJEMPLO DE TIPO SIMPLE DERIVADO
Definición de **miEntero**, rango de 10000-99999

```
<xsd:simpleType name="miEntero">
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="10000"/>
                 <xsd:maxInclusive value="99999"/>
        </xsd:restriction>
</xsd:simpleType>
```

minInclusive y maxInclusive restringe el min y max

- **Espacios en blanco**. Sirve para indicar la política de manejo de espacios en blanco en los textos. La etiqueta que lo controla es **whiteSpace** y tiene tres posibles valores para el atributo **value**:
  - **preserve**. No modificar espacios en blanco, ni tabuladores ni saltos de línea. Es decir se les tendrá en cuenta.
  - **replace**. Cada doble espacio o tabulador o salto de línea se cambia por un espacio (al estilo del tipo predefinido **normalizedString**)
  - **collapse**. Como el anterior, pero además elimina los espacios a izquierda y derecha. Es muy útil para usar en combinación con las propiedades de tamaño de texto vistas anteriormente.

Ejemplo (archivo ***prueba.xsd***):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="tipo1">
        <xs:restriction base="xs:string">
                <xs:whiteSpace value="collapse" />
        </xs:restriction>
 </xs:simpleType>
<xs:element name="prueba" type="tipo1" />
 </xs:schema>
```

En el archivo anterior, se define un elemento llamado ***prueba*** que tiene un tipo que colapsa los espacios en blanco y así en un documento XML que aplique este esquema, por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
 <prueba xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="prueba.xsd">
Este            es el texto
                que deseo       probar
</prueba>
```

Cuando mostremos el resultado en un programa que aplique el esquema (por ejemplo en las últimas versiones de **Internet Explorer, Firefox, Safari** o **Chrome**), tendremos el contenido de la etiqueta prueba en esta forma:

```
Este es el texto que deseo probar
```

Ejemplo supongamos que se quiere almacenar la dirección de vivienda habitual

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

- **Enumeraciones**. Las realiza una etiqueta llamada **enumeration** que sirve para indicar los posibles valores que puede tomar un componente. Por ejemplo:

```
<xs:simpleType name="diasSemanaTipo">
<xs:restriction base="xs:string">
        <xs:whiteSpace value="collapse" />
        <xs:enumeration value="Lunes" />
         <xs:enumeration value="Martes" />
        <xs:enumeration value="Miércoles" />
        <xs:enumeration value="Jueves" />
        <xs:enumeration value="Viernes" />
```

```
            <xs:enumeration value="Sábado" />
            <xs:enumeration value="Domingo" />
</xs:restriction>
</xs:simpleType>
```

Enumeraciones. **Limitan el contenido a una lista de valores**. Se realizan con una sucesión de etiquetas **enumeration**. Estas etiquetas se pueden combinar con las propiedades anteriores (aunque no tiene sentido) y no poseen atributo **fixed**. De modo que si un tipo deriva de una enumeración, podrá volver a enumerar para indicar valores válidos y estas deberán estar en la enumeración anterior; una vez más podremos restringir más, pero nunca menos.

Ejemplo de enumeración:

```
<xs:simpleType name="sexoTipo">
<xs:restriction base="xs:NCName">
        <xs:enumeration value="Hombre"/>
        <xs:enumeration value="Mujer"/>
</xs:restriction>
</xs:simpleType>
```

Mediante este tipo sólo podremos elegir como valores ***Hombre*** o ***Mujer***.

EJEMPLO DE TIPO SIMPLE DERIVADO

Utilización de la Propiedad Enumeration

```
<xsd:simpleType name="codigoPostal">
<xsd:restriction base="xsd:string">
        <xsd:enumeration value="28400"/>
        <xsd:enumeration value="28001"/>
        <xsd:enumeration value="28020"/>
<!--y así el resto ... -->
</xsd:restriction>
</xsd:simpleType>
```

- **Plantillas** (***pattern***). Permite establecer **expresiones regulares;** es decir, un texto con símbolos especiales que permiten establecer expresiones que ha de cumplir el contenido. Las expresiones regulares son bien conocidas por casi todos los programadores y dan una potencia increíble para establecer restricciones de texto avanzadas, lo que las hace muy utilizadas.

  Las plantillas se manejan con etiquetas **pattern** a las que, en el atributo **value**, se indica un texto con símbolos especiales que especifica la expresión a cumplir. Los símbolos que se pueden utilizar son:

Símbolo	Significado
texto tal cual	Hace que sólo se pueda escribir ese texto. Por ejemplo si se indica "*Hombre*", la restricción será escribir como valor posible exactamente el texto *Hombre*.
[xyz]	Permite elegir entre los caracteres x, y o z
[^xyz]	Prohibe usar cualquiera de los caracteres entre corchetes
[a-z]	Vale cualquier carácter de la *a* a la z.
^	Inicio de línea
$	Final de línea
+	Repite acepta el carácter precedente una o más veces
?	Acepta el carácter precedente 0 o más veces
*	Acepta el carácter precedente una o más veces
{n}	Acepta exactamente *n* repeticiones del carácter precedente.
{n,}	Acepta al menos *n* repeticiones del carácter precedente.
{n,o}	Acepta entre *n* y *n* repeticiones del carácter precedente.
\s	Permite indicar los caracteres especiales. Por ejemplo \^ representa el carácter circunflejo ^ para que sea tomado como texto y no como código especial.

\d: *cualquier dígito.*
\D: *cualquier no-dígito.*
\s: *espacio en blanco, retorno de carro, línea nueva, o intro.*
\S: *cualquier carácter distinto a espacio en blanco.*
(ab)*: *cualquier cosa entre paréntesis puede aparecer cero o más veces.*
(ab)?: *cualquier cosa entre paréntesis puede aparecer una o más veces.*
a{n}*: "a" puede aparecer en una cadena n veces.*

Por ejemplo la validación para un dato tipo DNI (8 cifras y un número), sería:

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

EJEMPLO DE TIPO SIMPLE DERIVADO

Definición del Tipo Simple "SKU"

```
<xsd:simpleType name="SKU">
<xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
 </xsd:restriction>
</xsd:simpleType>
```

Tres dígitos seguidos de un guion seguido de dos caracteres ASCII en mayúsculas

EJEMPLOS:

Valores mínimo y máxima que pueden tomar. Es un tipo simple que se aplican restricciones. Restringe un entero (integer).

facetas (ej. 1)

26

```
<xsd:element name="age">
   <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="0"/>
                <xsd:maxInclusive value="100"/>
        </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

Resolver el supuesto siguiente: Imaginemos que una persona para solicitar el carnet joven debe tener entre 16 y 24 años. Aplicaremos el supuesto sobre un element edad.

Enumeraciones. Restricción a un string (cadena de caracteres)

facetas (ej. 2)

27

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

Enumeraciones de otra forma:

facetas (ej. 2, alt.)

28

```
<xsd:element name="car" type="carType"/>

<xsd:simpleType name="carType">
   <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Audi"/>
        <xsd:enumeration value="Golf"/>
        <xsd:enumeration value="BMW"/>
   </xsd:restriction>
</xsd:simpleType>
```

Patrones-Rango. En este ejemplo, el elemento “letter” debe tener como valor una letra minuscula. Se debe adaptar al patron (pattern value[a-z])

facetas (ej. 3)

29

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

Patrones-varios. En este ejemplo, el elemento “initials” debe tomar como valor 3 letras mayúsculas o minúscula (sólo 3). El patrón deben ser combinaciones diferentes de letras.

facetas (ej. 4)

30

```
<xsd:element name="initials">
   <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
        </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Patrones-Rango. En este ejemplo, el elemento “choice” debe tomar como valor una de estas letras: x, y o z. Se debe elegir entre x,y,z.

facetas (ej. 5)

31

```
<xsd:element name="choice">
   <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:pattern value="[xyz]"/>
        </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

El patron son 5 digitos entre 0-9

facetas (ej. 6)

32

```
<xsd:element name="prodid">
   <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
        <xsd:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
        </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

El patron son cualquier combinación de letras incluyendo vacio.

facetas (ej. 7)

33

```
<xsd:element name="letter">
   <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:pattern value="([a-z])*"/>
        </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

En este ejemplo , el valor del campo “password” debe ser de 8 caracteres.

facetas (ej. 8)

34

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

Patrones longitud de caracteres. Los elementos length, minLengthy maxLengthpermiten indicar el número exacto, mínimo y máximo de caracteres que puede tener un valor de un elemento.

facetas (ej. 9)

35

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

Las restricciones se pueden aplicar también a atributos. Ejemplo suponiendo que un alumno puede pertenecer a un curso y dentro de un curso puede haber diferentes letras (A,B,C…). Podemos aplicar una restricción sobre el atributo “letra”, para que tome uno de los valores anteriores.

<xs:attribute name=”letra”>

```
        <xs:simpleType>
        <xs:restriction base=”xs:string”/>
                <xs:enumeration value=”A”/>
                <xs:enumeration value=”B”/>
                <xs:enumeration value=”C”/>
                <xs:enumeration value=”D”/>
        </xs:restriction>
        </xs:simpleType>
</xs:attribute>
```

Otra solución válida podría ser:

```
<xs:attribute name=”letra” type =”letraCurso”>
        <xs:simpleType name =”letraCurso”/>
        <xs:restriction base=”xs:string”/>
                <xs:pattern value=”[A-D]”/>
        </xs:restriction>
        </xs:simpleType>
</xs:attribute>
```

En este último caso se ha definido un tipo simple llamado letraCurso que podría ser reutilizado en otras partes del esquema. Hay que destacar que en lugar de utilizar una enumeración, se ha definido un patrón en forma de expresión regular, lo que permite tener mayor potencia para declarar valores predeterminados. También hubiera sido válido cualquiera de las siguientes expresiones:

```
<xs:pattern value=”[ABCD]”/>
<xs:pattern value=”[A|B|C|D]”/>
```

TIPOS COMPLEJOS

Los tipos de datos del apartado anterior sólo sirven para indicar contenidos simples (que contienen sólo información en el interior) de elementos o bien para indicar posibles valores a los atributos. Los tipos compuestos permiten definir contenidos más complejos. Puesto que lo normal es que los elementos de un documento XML puedan contener otros elementos y por supuesto atributos, es lógico que la mayoría de elementos indiquen mediante tipos compuestos su contenido. Los datos simples son apropiados para indicar el tipo de contenido de los atributos o bien para indicar el contenido de los elementos simples (lo que en DTD serían elementos sólo con contenido **# PCDATA**).

Al igual que los datos simples, los compuestos pueden ser globales o locales. En el caso de ser locales no se indica un nombre (atributo **name**) y entonces sólo se podrán utilizar para el elemento en el que se definieron. Los globales se pueden utilizar para distintos elementos y por lo tanto requieren que se indique su nombre.
Los tipos compuestos se definen con la etiqueta **complexType**. En esa etiqueta podemos utilizar diferentes modelos de definición de contenidos.

- Para definir elementos complejos se utiliza la siguiente sintaxis:

```
<xsd:element name="nombre_elemento">
  <xsd:complexType>
    ... Información sobre el tipo complejo ...
  </xsd:complexType>
</xsd:element>
```

  - El tipo complejo es local al elemento y no puede reutilizarse.

```
<xsd:element name="nombre_elemento"
             type="nombre_tipo">

<xsd:complexType name="nombre_tipo">
    ... Información sobre el tipo complejo ...
</xsd:complexType>
```

  - El tipo complejo es global y puede ser utilizado en la definición de cualquier elemento.

```
<xsd:complexType name="nombre_tipo">
  <xsd:complexContent>
    <xsd:extension base="nombre_tipo_base">
      ... Información adicional sobre el tipo base...
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- Es posible utilizar un mecanismo de "herencia" para reutilizar o extender tipos definidos previamente.

- Definiciones locales y globales:
  - Los elementos declarados en el primer nivel del un schema están disponibles para su uso en todo el esquema.
  - Los elementos declarados dentro de un tipo complejo (complexType), son locales al tipo y no pueden reutilizarse fuera de él.
  - El orden de las declaraciones en el primer nivel de un schema no especifican el orden de los elementos dentro del documento XML.

En realidad los elementos XML desde el punto de vista de la sintaxis XML Schema pueden tener cuatro tipos de contenido:

- **Contenido simple** (***Simple Content***). Sólo admite texto en su interior y no otros elementos (son los elementos definidos en DTD como **(#PCDATA)**)
- **Vacíos (*Empty*)**. No pueden contener ni más elementos dentro ni texto.
- **Contenido compuesto** (***Complex Content***). Pueden contener otros elementos, pero no **PCDATA**.
- **Contenido mixto (*Mixed Content*)**. Pueden contener tanto texto como más elementos.

La sintaxis de la etiqueta **complexType** admite señalar contenidos simples y compuestos. Para los vacíos y mixtos hay que utilizar atributos especiales en la etiqueta **element**. Sin indicar nada especial, **compexType** parte de que estamos definiendo contenidos complejos (es decir, elementos que contienen más elementos).
Los elementos pueden contener atributos, más adelante se indica la forma de incorporarlos.

Elementos vacíos

Para indicar que un elemento es vacío basta con no indicar valores e indicar el nombre del elemento sin indicar tipo de datos alguno. Ejemplo:

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

En este caso el elemento caso sólo dispone de un atributo llamado valor, no será posible meter ningún contenido en la etiqueta de casado. Si no deseamos atributos (aunque es muy extraño), entonces simplemente no habrá etiquetas **attribute** (pero sí todas las demás).

EJEMPLO

- Para declarar un elemento vacío con atributos, se utilizará la siguiente sintaxis:

```
<xsd:element name="producto">
  <xsd:complexType>
   <xsd:attribute name="prodid" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

- `<producto prodid="1345" />`

Definición de contenidos con texto

Se trata de usar la etiqueta **<xs:element** al estilo que se ha usado en los ejemplos. Es decir se indica el nombre y el tipo de datos e incluso se pueden indicar atributos dentro del apartado **complexType**, pero nunca se ponen elementos dentro de este elemento y así sólo se admitirá por contenido el texto (tipo **PCDATA**).
Ejemplo:

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

Definición de contenidos con texto y atributos

En este caso se indica que el elemento posee contenido simple en la etiqueta **complexType** dentro de la cual se indican los atributos, mientras que es la etiqueta del contenido simple (**simpleContent**) la que poseerá el tipo de datos para el contenido del elemento.
Hay dos formas de indicar contenido simple: por extensión (mediante etiqueta **extension**) y por restricción (etiqueta **restriction**).
Los atributos se deben indicar en el apartado **extension** que es el encargado de indicar los tipos .

Ejemplo:

```
<xs:element name="documento">
<xs:complexType>
        <xs:simpleContent>
                <xs:extension base="xs:string">
                <xs:attribute name="idioma" type="xs:string" />
                </xs:extension>
        </xs:simpleContent>
</xs:complexType>
</xs:element>
```

En el ejemplo, documento es un elemento de tipo string (texto) que contiene un atributo llamado idioma (también string). Es un poco enrevesado, pero es necesario hacerlo así

EJEMPLO

- Para declarar un elemento no vacío con atributos, y sin elementos hijos, se utilizará la siguiente sintaxis:

```
<xsd:element name="medida">
  <xsd:complexType>
        <xsd:simpleContent>
                <xsd:extension base="xsd:integer">
                        <xsd:attribute name="metrica" type="xsd:string" />
                </xsd:extension>
        </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Definición de contenidos compuestos

Como se ha comentado antes, los contenidos compuestos se refieren a los elementos que contienen otros elementos (pero nunca texto libre). Hay tres posibles tipos de elementos a contener: **secuencias**, **elecciones (choice)** y contenidos libres (**all**). Además se pueden incorporar atributos.

Secuencias

Dentro de un elemento es habitual indicar su contenido como una secuencia de elementos. Esto se permite con la etiqueta **sequence**, dentro de la cual se añaden etiquetas **element** para indicar los elementos que entran en la secuencia.

Proporciona una representación de un conjunto ordenado de elementos, de forma que en el documento instancia correspondiente, los elementos tengan el mismo orden de aparición que en el Esquema; nótese que puede haber cero o muchos elementos para cada tipo, dependiendo de los valores **minOccurs y maxOccurs** de cada elemento.

- secuence
  - Indican una secuencia ordenada de elementos, dónde los elementos sólo pueden ocurrir una vez.

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

Ejemplo:

```
<xs:simpleType name="emailT">
        <xs:restriction base="xs:string">
                <xs:pattern value="[A-Za-z]{3,}@.{3,}"></xs:pattern>
        </xs:restriction>
</xs:simpleType>
        <xs:element name="email">
        <xs:complexType >
                <xs:sequence>
                        <xs:element name="remite" type="emailT" />
                        <xs:element name="para" type="emailT" minOccurs="1"
                        maxOccurs="unbounded" />
                        <xs:element name="CC" type="emailT" minOccurs="0"
                        maxOccurs="unbounded" />

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

En el ejemplo, el elemento email está compuesto de cuatro elementos. El ***remite*** (que tiene obligatoriamente que aparecer una vez), el ***para*** que aparecerá al menos una vez y que puede aparecer tantas veces como se desee y los apartados opcionales ***CC*** y ***CCO*** que pueden aparecer repetidos.

La etiqueta **sequence** posee los atributos **minOccurs** y **maxOccurs** para indicar que el bloque de la secuencia se puede repetir.

EJEMPLO:PELI.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pelicula xmlns:xsi="http://www.w3.org/2001/XMLschema-instance"
        xsi:noNamespaceSchemaLocation="peli.xsd">
 <titulo>Brasil</titulo>
</pelicula>
```

PELI.XSD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="pelicula">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="titulo" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

EJERCICIO a PARTIR DEL documento peli.xml propuesto a continuación, realizar peli.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula  titulo="brasil" minutos="120"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="pel_con_repeticion.xsd">
<director>Terry Gilliam</director>
<reparto>
        <interprete>Robert de Niro</interprete>
        <interprete>Bob Hoskins</interprete>
        <interprete>Ian Holm</interprete>
</reparto>
</pelicula>
```

EJERCICIO A PARTIR DEL DOCUMENTO XML SIGUIENTE REALIZAR EL XSD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persona xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ejercicio.xsd">
        <nombre/>
        <nacimiento/>
        <direccion>
                <calle>Caoba 1</calle>
                <poblacion>Madrid</poblacion>
                <provincia>Madrid</provincia>
                <CPostal>28005</CPostal>
        </direccion>
        <varon/>
</persona>
```

Elecciones (choice)

Sirven para permitir elegir uno de entre varios elementos. Su funcionamiento es el mismo que en las secuencias, pero en este caso se utiliza una etiqueta llamada **choice**.

Permite indicar un único elemento de una secuencia de elementos válidos. Especifica que de entre los elementos hijos solo pueda aparecer uno u otro. Los atributos **maxOccurs y minOccurs** permiten que el documento instancia seleccione el número de elementos (por ejemplo, entre dos y cuatro).

Veamos cómo expresar distintas alternativas.

EJEMPLO

Una selección entre varias:

```
<xsd:element name=”transporte”>
<xsd:complexType>
        <xsd:choice>
                <xsd:element name=”tren” type=”xsd:string”/>
                <xsd:element name=”avion” type=”xsd:string”/>
                <xsd:element name=”automovil” type=”xsd:string”/>
        </xsd:choice>
</xsd:complexType>
</xsd:element>
```

De forma que al ser choice equivalente a un “o exclusivo”, el elemento transporte sólo puede contener uno de los tres elementos tren, avión o automóvil.

EJEMPLO

Dar la posibilidad de repetir la opción. Un ejemplo es una cadena de bits:

```
<xsd:element name=”cadena-binaria”>
<xsd:complexType>
        <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>
                <xsd:element name=”cero” type=”xsd:unsignedByte” fixed=”0”/>
                <xsd:element name=”uno” type=”xsd:unsignedByte” fixed=”1”/>
        </xsd:choice>
</xsd:complexType>
</xsd:element>
```

EJEMPLO:

- choice
  - Permiten indicar un único elemento de una secuencia elementos válidos.

```
<xsd:element   name=”iidAlumno">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element  name=”NIF"  type="xs:string" />
         <xsd:element  name=”PASAPORTE"  type="xs:string" />
      </xsd:choice>
   </xsd:complexType>
 </xsd:element>
```

EJEMPLO

```
<xs:element name="identificación">
<xs:complexType>
        <xs:choice>
                <xs:element name="firma" type="xs:NCName"/>
                <xs:element name="código" type="xs:NCName" />
        </xs:choice>
</xs:complexType>
```

En el ejemplo, el elemento ***identificación***, consta de dos posibles elementos ***firma*** y ***código*** de los que sólo se podrá incluir uno.

Etiqueta all

Se trata de una posibilidad similar a **choice** y **sequence** que se utiliza de la misma forma y que tiene como diferencia principal que los elementos que contiene pueden aparecer cero o una vez y además en el orden que quieran.
Indica una secuencia **no ordenada de elementos**, donde los elementos sólo pueden ocurrir una vez.

EJEMPLO

```
<xsd:element name=”Pais” maxOccurs=”unbounded”>
<xsd:complexType>
        <xsd:all>
                <xsd:element name=”Nombre” type=”xsd:string”/>
                <xsd:element name=”Capital” type=”xsd:string”/>
                <xsd:element name=”Poblacion” type=”xsd:int”/>
                <xsd:element name=”Extension” type=”xsd:int”/>
        <xsd:all>
<xsd:complexType>
<xsd:element>
```

se permite que los elementos hijo del elemento Pais puedan dares en cualquier orden en los documentos instancia.

EJEMPLO:

```
<xs:element name="identificación">
<xs:complexType>
 <xs:all>
        <xs:element name="firma" type="xs:NCName"/>
         <xs:element name="código" type="xs:NCName" />
</xs:all>
</xs:complexType>
```

En este caso la ***firma*** y el ***código*** pueden aparecer o no, aparecer los dos e incluso el orden será indiferente. Es una etiqueta muy potente que ahorra mucho trabajo.
Esta etiqueta tiene los atributos **minOccurs** y **maxOccurs**, pero sólo se puede indicar como valores cero o uno.

EJEMPLO: a PARTIR DEL DOCUMENTO PELI.XML REALIZAR PELI.XSD UTILIZANDO ALL

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="pel_tit_min_all.xsd">
        <titulo>brasil</titulo>
        <minutos>120</minutos>
</pelicula>
```

Mezcla de elementos

A veces los contenidos de un documento XML son extremadamente complejos y por eso se permite en los esquema colocar etiquetas **choice** dentro de etiquetas **sequence** y viceversa. Y

lo mismo ocurre con las etiquetas **all**. Estas posibilidades permiten crear cualquier tipo de esquema por complejo que resulte.

Ejemplo:

```
<xs:element name="correo">
<xs:complexType>
<xs:sequence>
        <xs:element name="remite" type="xs:string"/>
        <xs:element name="para" type="xs:string" minOccurs="1"
        maxOccurs="unbounded"/>
 <xs:choice>
         <xs:element name="cc" type="xs:string" minOccurs="1" maxOccurs="unbounded"
/>
 <xs:element name="cco" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
</xs:choice>
</xs:sequence>
</xs:complexType>
 </xs:element>
```

En el ejemplo, el elemento ***correo*** consta de tres elementos: ***remite, para*** y un tercero que puede ser ***cc*** o ***cco***.

Añadir atributos

En los apartados **complexType**, los atributos del elemento se definen al final del apartado **complexType** (justo antes de cerrarse).

Ejemplo:

```
<xs:element name="persona">
<xs:complexType>
<xs:sequence>
        <xs:element name="nombre" type="xs:string" />
        <xs:element name="apellidos" type="xs:string" />
</xs:sequence>
<xs:attribute name="sexo">
 <xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:enumeration value="Hombre" />
               <xs:enumeration value="Mujer" />
        </xs:restriction>
 </xs:simpleType>
</xs:attribute>
        <xs:attribute name="fechaNacimiento" type="xs:date" use="required" />
</xs:complexType>
 </xs:element>
```

Las personas contienen dos elementos en secuencia (***nombre*** y ***apellidos***) y dos atributos: uno opcional (***sexo***) que sólo pueden tomar los valores ***Hombre*** o ***Mujer*** y uno obligatorio para la fecha de nacimiento.

Contenidos mixtos

Es el caso más complejo. Se trata de elementos que contienen otros elementos y además texto (e incluso atributos). Para permitir esta posibilidad hay que marcar el atributo **mixed** de la etiqueta **complexType** a **true**.

Ejemplo:

```
<xs:element name="documento">
 <xs:complexType mixed="true">
          <xs:choice>
                    <xs:element name="negrita" type="xs:string" minOccurs="0"
                    maxOccurs="unbounded"/>
          </xs:choice>
           <xs:attribute name="lenguaje" type="xs:language" />
</xs:complexType>
 </xs:element>
```

EJEMPLO

- Para declarar un elemento con contenido "mixto", basta con añadir un atributo "mixed" al elemento *xsd:complexType*:

```
<xsd:element name="faxenvio">
  <xsd:complexType mixed="true">
        <xsd:sequence>
          <xsd:element name="cliente" type="xsd:string"/>
          <xsd:element name="idenvio" type="xsd:positiveInteger"/>
          <xsd:element name="fechaenvio" type="xsd:date"/>
        </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- La declaración anterior permitiría un texto como el siguiente:

```
<faxenvio>
  Estimado Sr. <cliente>Juan Pérez</cliente>: Su pedido con
  número de envío <idenvio>1032</idenvio> se distribuirá el
  día <fechaenvio>2011-07-13</fechaenvio>.
</faxenvio>
```

Uso de elementos y atributos globales

Se trata de definir un elemento o un atributo para ser reutilizado en diferentes partes del documento. La forma de utilizarlos es:

- Definirlos en la zona global (es decir, no definirlos dentro de ningún otro componente)
- En el caso de los elementos no se pueden indicar los atributos **minOccurs** y **maxOccurs** para indicar la cardinalidad, porque sólo tiene sentido dentro de otro elemento.
- En el caso de los atributos no se puede utilizar el atributo **use**, que indica la obligatoriedad de uso del atributo.
- Una vez definidos, donde se quieran reutilizar se define el elemento o atributo dentro del componente en el que se quiere colocar y no se le da nombre, sino que se usa el atributo **ref** para indicar el nombre del elemento o atributo global.

EJEMPLO

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="email">
 <xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:pattern value="[A-Za-z]{3,}@.{3,}">
        </xs:pattern>
        </xs:restriction>
 </xs:simpleType>
</xs:element>
<xs:element name="trabajador">
<xs:complexType>
        <xs:sequence>
                <xs:element name="nombre" type="xs:string" />
                <xs:element name="apellidos" type="xs:string" />
               <xs:element ref="email" minOccurs="1"
               maxOccurs="unbounded" />
         </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

GRUPOS DE ELEMENTOS

La etiqueta **group** permite realizar grupos de elementos y eso permite organizarse mejor a la hora de crear un esquema. Dentro de cada grupo podemos utilizar etiquetas **sequence, choice** y **all** de la misma forma que la vista anteriormente y así después utilizar el grupo en la forma deseada. Muchas veces los grupos se definen de forma global y así se pueden utilizar en distintos elementos; pero es posible definirles localmente (suele tener menos interés hacerlo en local).

Los grupos cuando se definen de forma global requieren indicar un nombre para ellos (si se definen de forma local no). Cuando un elemento desea incorporar un grupo global, utiliza la etiqueta **group** y con el atributo **ref** indicaría el nombre del elemento global definido anteriormente. Pueden contener los atributos **minOccurs**, **maxOccurs** para indicar las veces que puede repetirse el grupo en el elemento que le contiene.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:group name="seccionesCorreo">
          <xs:sequence>
                    <xs:element name="remite" type="xs:string"/>
                    <xs:element name="para" type="xs:string" minOccurs="1"
                   maxOccurs="unbounded"/>
          <xs:choice>
         <xs:element name="cc" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
         <xs:element name="cco" type="xs:string" minOccurs="1" maxOccurs="unbounded"
         />
          </xs:choice>
         </xs:sequence>
 </xs:group>

 <xs:element name="correo">
 <xs:complexType>
          <xs:sequence>
          <xs:group ref="seccionesCorreo" />
                    <xs:element name="contenido" type="xs:string" />
         </xs:sequence>
</xs:complexType>
 </xs:element>
 </xs:schema>
```

Ejemplo Supongamos que queremos representar las características de un coche. Nos gustaría almacenar: marca, modelo, caballos. Cualquier automóvil tiene esas características, la definición del esquema podría ser la siguiente:

```
<xs:group name="grupoCoche">
  <xs:sequence>
        <xs:element name="marca" type="xs:string"/>
        <xs:element name="modelo" type="xs:string"/>
        <xs:element name="caballos" type="xs:integer"/>
 <xs:/sequence>
<xs:group>
<xs:element name="coche" type="tipoCoche"/>
 <xs:complexType name="tipoCoche">
        <xs:sequence>
                  <xs:element name="codigo" type="xs:integer"/>
                  <xs:group ref="grupoCoche"/>
```

```
                <xs:element name=”combustible” type=”xs:string”/>
        </xs:sequence>
</xs:complexType>
……..
```

ANOTACIONES

ELEMENTO ANNOTATION. Proporciona un mecanismo para documentar o anotar los componentes de un Esquema, haciendo una función similar a los comentarios, aunque dando una información más completa, ya que annotation no se reduce a la forma: <!-- texto -->, sino que puede proporcionar documentación tanto para quien maneje el documento como para el procesado automático del mismo.

ELEMENTO DOCUMENTATION. Consiste bien en un texto, legible por un humano, bien en una referencia a un texto dado por un URI, siempre dentro de un elemento annotation. Su contenido en un documento instancia no tiene límites, de forma que un elemento annotation puede contener muchos elementos documentation y cada uno puede presentar información más o menos redundante, como sería el caso del uso de lenguajes distintos. Sus dos atributos opcionales son source (una URI que el procesador no valida, que representa cualquier documento considerado relevante) y xml:lang (que especifica el lenguaje de la documentación).

ELEMENTO APPINFO. Proporciona información a otras aplicaciones, lo que supone un procesamiento externo al propio documento y que sólo un elemento annotation puede contener.

- XML-Schema proporciona 2 elementos de comentarios:
  - Documentation (explicación al lector)
  - AppInfo (explicación al lector o a un programa)

```
<xsd:complexType nombre=“elemento”>
<xsd:annotation”>
<xsd:documentation>este elemento es .. </xsd:documentation>
</xsd:annotation”>
</xsd:complexType>
```

EJEMPLO :PELI.XML ANOTACION

```
<?xml version="1.0" encoding="UTF-8"?>
<pelicula  titulo="brasil" minutos="120"
```

```
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="pel_con_repeticion_anotacion.xsd">
        <director>Terry Gilliam</director>
        <reparto>
                <interprete>Robert de Niro</interprete>
                <interprete>Bob Hoskins</interprete>
                <interprete>Ian Holm</interprete>
        </reparto>
</pelicula>
```

PELI.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<element name="pelicula">
        <complexType>
        <sequence>
            <element name="director"/>
           <element name="reparto">
             <complexType>
               <sequence>
                  <element name="interprete" maxOccurs="unbounded">
                    <annotation>

                          <documentation>incluye todos los actores que quieras
                          </documentation>
                    </annotation>
                  </element>
                  </sequence>
                  </complexType>
                          </element>
                          </sequence>
```

```
                <attribute name="titulo" type="string" use="required"/>
                <attribute name="minutos" type="integer" use="optional"/>
            </complexType>
        </element>
</schema>
```

MODELOS DE DISEÑO DE ESQUEMAS XML

Existen varios modelos de estructuración de las declaraciones al construir los esquemas, pero recordaremos que el orden en que aparecen los componentes en un esquema no es representativo.

- Diseño Anidado o de muñecas rusas: Se llama así porque se anidan declaraciones unas dentro de otras, como las muñecas MatriosKas. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Esto produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones.

EJEMPLO: **Realizar el documento xsd que valide el siguiente documento.**

matricula.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<matricula xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="matricula_rusas.xsd">
        <personal>
            <dni>99223366M</dni>
            <nombre>Pepito grillo</nombre>
            <titulacion>Ciclo ASIR_DAM</titulacion>
            <curso_academico>2012/2013</curso_academico>
            <domicilios>
                <domicilio>
                    <nombre>C/ Principal nº1</nombre>
                </domicilio>
                <domicilio>
```

```
                                <nombre>C/ Secundaria nº2</nombre>
                        </domicilio>
                </domicilios>
        </personal>
        <pago>
                <tipo_matricula>Matrícula Ordinaria</tipo_matricula>
        </pago>
</matricula>
```

- Diseño con tipos con nombres reutilizables: Se definen tipos de datos simples o complejos a los que se identifica con un nombre (son plantillas, como las clases en la programación orientada a objetos). Al declarar elementos y atributos, se indica que son de alguno de los tipos con nombres previamente definidos.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="matricula" type = "tmatricula">

        <xs:complexType name="tmatricula">
            <xs:sequence>
                <xs:element name="personal" type ="tpersonal"/>
                <xs:element name="pago" type="tpago"/>
            </xs:sequence>
        </xs:complexType>

        <xs:complexType name="tpersonal">
                        <xs:all>
                            <xs:element name="dni" type="xs:string"/>
                            <xs:element name="nombre" type="xs:string"/>
                            <xs:element name="titulacion" type="xs:string"/>
                            <xs:element name="curso_academico" type="xs:string"/>
                            <xs:element name="domicilios"type="tdomicilios"/>
                        </xs:all>
        </xs:complexType>

        <xs:complexType name ="tpago">
                        <xs:all>
                            <xs:element name="tipo_matricula" type="xs:string"/>
                        </xs:all>
        </xs:complexType>

        <xs:complexType name="tdomicilios">
                <xs:sequence>
                    <xs:element name="domicilio" maxOccurs="ubounded">
                            <xs:complexType>
                                <xs:all>
                                        <xs:element name="nombre" type="xs:string"/>
                                </xs:all>
                            </xs:complexType>
                    </xs:element>
                </xs:sequence>
        </xs:complexType>

    </xs:element>
```

- Diseño plano. Se declaran los elementos y atributos y se indica una referencia a su definición.

Herramientas on-line:

http://www.freeformatter.com/xsd-generator.html

Entrada: Documento XML
Salida: Esquema XML XSD