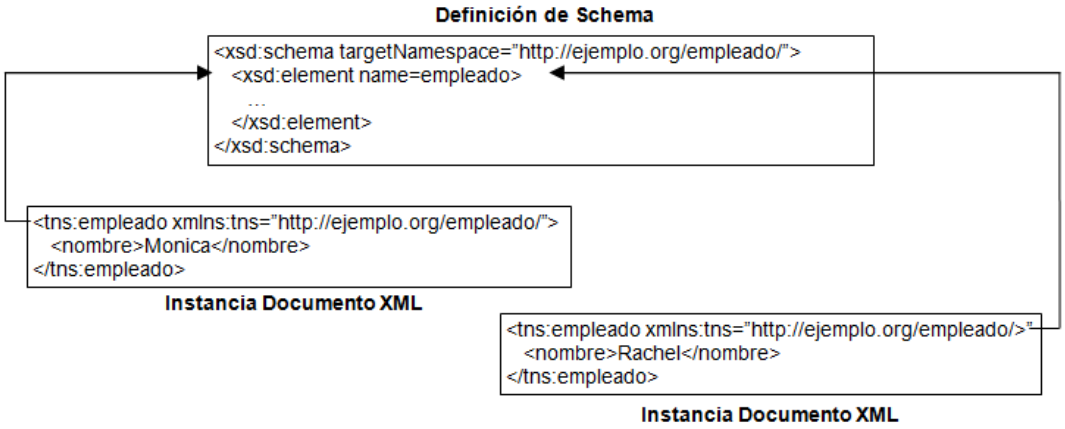
XML-Schema

XML-Schema, a diferencia de DTDs, es un vocabulario basado en XML para describir instancias de documentos XML. Un "schema" describe una clase de documentos, del cual puede haber varias instancias. Esta relación es parecida a la que hay entre *clases* y *objetos* en los sistemas orientados a objetos. Una clase está relacionada con un objeto de la misma forma que un schema a un documento XML. Por ello, al trabajar con un XML Schema, estaremos trabajando con más de un documento.



Los elementos utilizados en una definición de Schema provienen del namespace *http://www.w3org/2001/XMLSchema*. El siguiente código muestra una plantilla básica para Schemas:

```
<xsd:schema xmln:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ejemplo.org/empleado">
  <!-- aquí van las definiciones -->
</xsd:schema>
```

Las definiciones de Schema deben tener un elemento raíz (root) *xsd:schema*. Dentro de este elemento van anidados una variedad de elementos (p.e.: xsd:element, xsd:attribute, xsd:complexType).

El hecho de que un XML Schema es en sí un documento XML resuelve la primera limitante que tenían los DTDs. Las definiciones de Schema pueden ser procesados con una variedad de herramientas XML 1.0 estándar, tales como DOM, SAX, XPath y XSLT. Por esta simplicidad, han aparecido muchas herramientas para el manejo de schemas.

Por tanto, las ventajas de los esquemas son:

- **La sintaxis es XML**, por lo que son analizables como cualquier otro documento XML.
- Soportan íntegramente los espacios de nombres.
- Permiten **validaciones de datos avanzadas**.
- Proporcionan una mayor facilidad **para crear validaciones complejas y reutilizables**.
- Soportan conceptos avanzados como herencia y sustitución de tipos.

Desventajas de los esquemas:

- Son más complejas de entender que las DTD.
- Presentan más incompatibilidades con software que las DTD.
- No permiten definir entidades.

1. Estructura de los esquemas

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**.

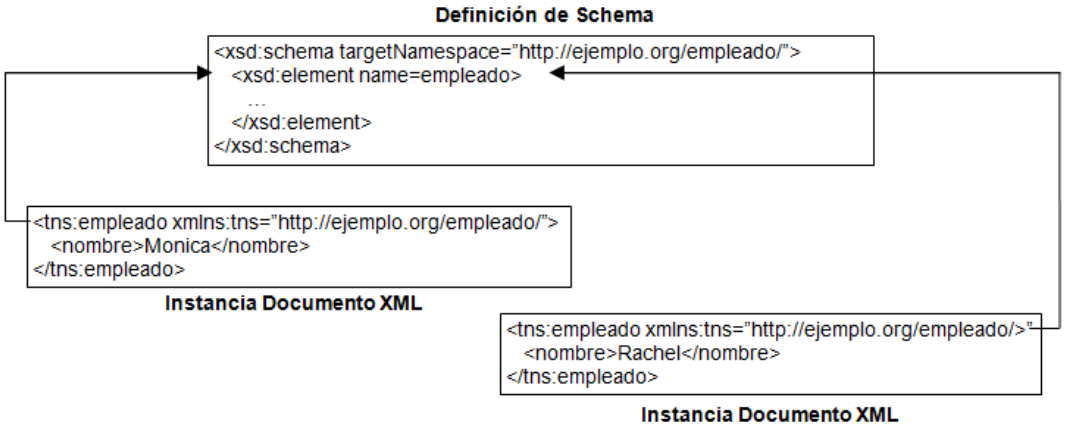
Etiqueta schema

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">
```

Atributo targetNamespace

Las definiciones dentro del xsd:schema están automáticamente asociadas con el *namespace* especificado en el atributo *targetNamespace*. En el ejemplo siguiente, las definiciones del schema estarían asociadas con el *namespace* http://ejemplo.org/empleado/.



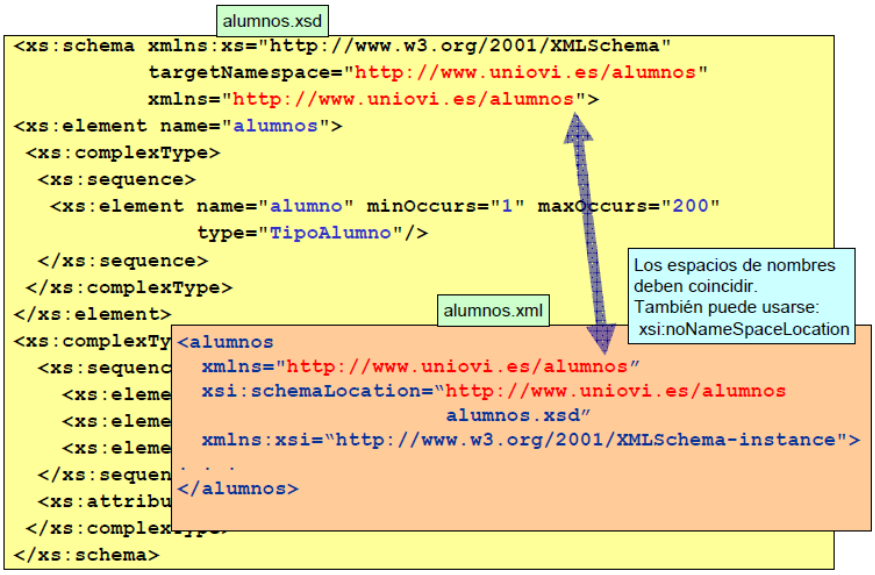
En la instancia (documento XML), declararemos lo siguiente:

<tns:empleado xmlns:tns="http://ejemplo.org/empleado/" />

Donde decimos que el identificador de namespace (en el ejemplo, **tns**), es la clave que vincula la instancia de documento xml (en el ejemplo, **empleado**) con la definición de schema correspondiente (en el ejemplo, el definido en el namespace **http://ejemplo.org/empleado/**). Esto NO indica que haya un esquema en esa dirección ni nada por el estilo, simplemente se usan URIs para definir namespaces para asegurar de que no colisionan nombres de namespaces de documentos distintos.

Sólo recalcar que: el namespace **(xmlns)** del elemento del empleado **("http://ejemplo.org/empleado/")** tiene que ser el mismo que el definido en el *targetNamespace* de la definición del Schema.

Para el siguiente ejemplo,
En el XML-Schema: *targetNamespace="http://www.uniovi.es/alumnos"*
En el XML: *xmlns:"http://www.uniovi.es/alumnos"*



También podemos hacer uso de los atributos:

- **schemaLocation**: se usará para indicar el espacio de nombres destino, seguido del documento de esquema.

XSD: libreria.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.libros.org"
        xmlns="http://www.libros.org"
        elementFormDefault="qualified">
  <xs:element name="Libreria" type="TipoLibreria"/>
  <xs:complexType name="TipoLibreria">
   <xs:sequence>
    <xs:element name="Libro" type="TipoLibro" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TipoLibro">
   <xs:sequence>
    <xs:element name="Titulo" type="xs:string"/>
    <xs:element name="Autor" type="xs:string" maxOccurs="3"/>
    <xs:element name="Fecha" type="xs:string"/>
    <xs:element name="ISBN" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
</xs:schema>
```

XML: libreria.xml

```
<?xml version="1.0"?>
<Libreria xmlns="http://www.libros.org"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.libros.org libreria.xsd">
 ...
</Libreria>
```

XML: libreria.xml

```
<?xml version="1.0"?>
<Libreria xmlns="http://www.libros.org"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.libros.org libreria.xsd">
 <Libro>
  <Titulo>My Life and Times</Titulo>
  <Autor>Paul McCartney</Autor>
  <Autor>Phil McCartney</Autor>
  <Fecha>July, 1998</Fecha>
  <ISBN>94303-12021-43892</ISBN>
 </Libro>
 ...
</Libreria>
```

- **noNamespaceSchemaLocation**: se usará cuando el esquema esté en un ***fichero local***, y no se especifique espacio de nombres para el esquema *(es decir, cuando no se especifique el targetNameSpace en el XML-Schema y por tanto oblique a especificar xmlns en el documento XML).*

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:element name="helado">
                <xs:complexType>
                …
                </xs:complexType>
        </xs:element>
</xs:schema>
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
<helado
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="helado.xsd">
…
</helado>
```

XSD: esquema1.xsd

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

XML

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

- **elementFormDefault/attributeFormDefault**: sirven para exigir que al usar el esquema desde un documento, los elementos/atributos se asocien con el targetNamespace del esquema, es decir, tienen que estar calificados por el prefijo del namespace al que pertenecen:

XSD (personSchema.xsd)

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

XML: los elementos locales del namespace, deben estar calificados con el prefijo del namespace

```
<ivan:person xmlns:ivan="http://www.ivan.com/schemas"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ivan.com/schemas personSchema.xsd"
hasDog=”true”>
<ivan:firstName>Steven</ivan:firstName>
<ivan:lastName>Segal</ivan:lastName>
<ivan:age>39</ivan:age>
</ivan:person>
```

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 colocan 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. 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.ejemplo.net/doc"
targetNamespace="http://www.ejemplo.net/doc">

<xs:element …> <!--Definición global

    <xs:simpleType …> <!--Definición local,
    …
    </xs:simpleType>
…
</xs:element>
```

```
</xs:schema>
```

2. Elementos

➢ SINTAXIS COMPLETA

```
<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" >
```

Un ejemplo:

```
<?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>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="heladeria">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="helado" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="helado">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="sabor"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="sabor">
        <xs:complexType mixed="true"/>
</xs:element>
</xs:schema>
```

Ejemplo de uso de type y ref:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">

 <xsd:element name="persona" type="tipoPersona"/>
 <xsd:element name="comentario" type="xsd:string"/>

 <xsd:complexType name="tipoPersona">
          <xsd:sequence>
                   <xsd:element name="datos" type="info"/>
                    <xsd:element ref="comentario" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="nacimiento" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="info">
          <xsd:sequence>
                   <xsd:element name="nombre"    type="xsd:string"/>
                   <xsd:element name="apellidos" type="xsd:string"/>
                   <xsd:element name="dni"       type="xsd:string"/>
          </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

- **TIPOS SIMPLES DE DATOS**

Son los tipos básicos de XML. Ejemplo de uso:

```
<?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>
```

Los tipos básicos son:

- **string**: representan textos con cualquier contenido excepto los símbolos <, > & y las comillas para los que se usará la entidad correspondiente.
- **boolean**: Sólo puede contener los valores verdadero o falso, escritos como true o false, o como 1 (verdadero) ó 0 (falso).
- **integer**
- **float**
- **double**
- **decimal**
- **duration**: representa duraciones de tiempo en formato ISO 8601 (sección 5.3) PnYnMnDThms 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 segundos y medio*
- *PT12H30M40.5S doce horas treinta minutos y cuarenta segundos y medio*

Como se observa la P es obligatoria y la T sirve para separar los valores de fecha de los valores hora.

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

- **TIPOS SIMPLES PERSONALES**

Sintaxis para la definición de tipos global:

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

Sintaxis para la definición de tipos local (sólo aplicaría la definición al "elemento1"):

```
<xs:element name= "elemento1">
        <xs:simpleType>
        ...definición del tipo....
        </xs:simpleType>
</xs:element>
```

- **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.

- **Tipos simples por restricción:**

```
<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.

```
<xsd:simpleType name="volumen">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0.25"/>
    <xsd:maxInclusive value="5.00"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tamaño">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="individual"/>
    <xsd:enumeration value="regular"/>
    <xsd:enumeration value="familiar"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="refresco">
  <xsd:simpleType>
    <xsd:union memberTypes="volumen tamaño" />
  </xsd:simpleType>
</xsd:element>
```

Posibles restricciones que se pueden establecer:

- Sobre los textos:
  - **length**: establece una longitud fija
  - **minLength**: establece un mínimo en la longitud
  - **maxLength**: establece un máximo en la longitud

- Sobre los dígitos:
  - **totalDigits**: número exacto de cifras permitidas en el elemento o atributo. Debe ser mayor que 0.
  - **fractionDigits**: máximo número de decimales que puede tener el número. Debe ser mayor ó igual a 0.

- **maxExclusive:** establece un valor máximo. El valor debe ser mayor que el establecido por la etiqueta a través del atributo *value*.
- **minExclusive:** establece un valor mínimo. El valor debe ser menor que el establecido por la etiqueta a través del atributo *value*.
- **maxInclusive:** establece un valor máximo. El valor debe ser mayor ó igual que el establecido por la etiqueta a través del atributo *value*.
- **minInclusive:** establece un valor mínimo. El valor debe ser menor ó igual que el establecido por la etiqueta a través del atributo *value*.

- Sobre los espacios en blanco: la etiqueta que lo controla es **whitespace**, y tiene 3 posibles valores para el atributo value:
  - **preserve**: mantiene los espacios en blanco, tabuladores, saltos de línea y retornos de carro.
  - **collapse**: borra todos los espacios en blanco, tabuladores, saltos de línea y retornos de carro.
  - **replace**: sustituye todos los espacios en blanco, tabuladores, saltos de línea y retornos de carro por un único espacio en blanco.

```
<?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>
```

- Enumeraciones: limitan el contenido a una lista de valores:

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

- Plantillas: permite establecer expresiones regulares. 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.

Símbolo	*Significado*
. (punto)	Hace referencia a cualquier carácter/número/…, pero sólo uno. Valores posibles, “a1z”, “axz”. Valores no posibles, “a12z”, “a1vz”.
a|b|c	Valores posibles, “a”, “b”, “c”. Valores no posibles “ab”, “bc”.
PB&J	Valor posible PB&J
1+2	Valores posibles, “112”, “1112”.
1\+2	Valor posible “1+2”

```
<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>
<xsd:element name="letra">

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

```
</xsd:element>
```

```
<xsd:element name="idioma">

<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="ingles|español"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:element>
```

```
<xsd:element name="temperatura">

<xsd:simpleType>
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="-10"/>
    <xsd:maxInclusive value="120"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:element>
```

- **ATRIBUTOS**

Sintaxis:

```
<xs:attribute
name="nombre del elemento"
type="tipo global de datos"
ref="declaración del elemento global"
form="cualificación" <!--qualified o unqualified  -->
id="identificador"
default="valor por defecto"
fixed="valor fijo"
use="uso" <!-- prohibited, optional o required - >
>
```

Fixed en atributos expresa que si no se define, se utiliza éste. Si se define, ha de tener el valor que se indique en éste.

- **TIPOS COMPUESTOS**

Los elementos complejos son todos aquellos que se componen de otros elementos, y/o que puedan contener atributos propios. Podemos identificar como elementos complejos los siguientes:

- Elementos que contienen otros elementos en su interior.
- Elementos que contienen atributos en su interior.
- Elementos que contienen otros elementos y atributos en su interior.

Ejemplos:

```
<curso letra=”A”>1</curso>
```

```
<xs:element name="curso" >
        <xs:complexType>
            <xs:attribute name="letra" type="xs:string" />
        </xs:complexType>
</xs:element>
```

“type” y “ref” también se podrían usar para los atributos, así, el siguiente ejemplo sería válido:

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

 <xs:element name="size" type="SizeType"/>

 <xs:complexType name="SizeType">
  <xs:attribute name="system" type="xs:string" use="required"/>
  <xs:attribute ref="dim"/>
  <xs:attribute name="value" default="10">
   <xs:simpleType>
    <xs:restriction base="xs:integer">
     <xs:minInclusive value="2"/>
     <xs:maxInclusive value="20"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>

 <xs:attribute name="dim" type="xs:integer" fixed="1"/>

</xs:schema>
```

➢ DEFINICIÓN DE CONTENIDOS COMPUESTOS

Los contenidos compuestos se refieren a los elementos que contienen otros elementos (pero nunca texto libre). Además, se pueden incorporar atributos. Hay tres posibles tipos de elementos a contener:

- **Secuencias (<xs:sequence>)**:permite indicar el orden específico en el que deben aparecer los elementos hijo.

```
<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>
```

- **Elecciones (<xs:choice>)**: especifica que, de entre los elementos hijo, sólo puede aparecer uno. También tiene los atributos minOccurs/maxOccurs.

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

- **Contenidos libres (<xs:all>)**: especifica que todos los elementos hijo pueden aparecer en cualquier orden determinado, siempre que sólo aparezcan una vez. También tiene los atributos minOccurs/maxOccurs, pero sólo podrán tener los valores 0 ó 1.

```
<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>
```

Se pueden definir **mezcla de elementos** de la siguiente forma:

```
<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>
```

Los **atributos** se definen al final del complexType. Para el siguiente ejemplo, el elemento persona tendrá los atributos *sexo* y *fechaNacimiento*:

```
<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>
```

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

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

ANEXOS

Restricciones en los tipos de datos:

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

Tipos de datos derivados de STRING

Name	Description
ENTITIES	
ENTITY	
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
IDREFS	
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NCName	
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
NMTOKENS	
normalizedString	A string that does not contain line feeds, carriage returns, or tabs
QName	
string	A string
token	A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

Uso de *extensión* (<xsd:extension> , <xsd:simpleContent> y <xsd:complexContent>)

Mediante el elemento <xsd:extension> es posible crear un tipo nuevo de secuencia a partir de un elemento <xsd:simpleType> o <xsd:complexType> existente. Lo anterior permite ajustar ciertas secuencias para casos específicos sin la necesidad de crear un nuevo elemento simpleType o complexType.

Los elementos <xsd:simpleContent> y <xsd:complexContent> son utilizados en conjunción de <xsd:extension> al momento de generar la nueva secuencia.

```
<xsd:simpleType name="formatoclasificacion">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Trabajo"/>
    <xsd:enumeration value="Familia"/>
    <xsd:enumeration value="Personal"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="formatonuevo">
  <xsd:simpleContent>

    <xsd:extension base="formatoclasificacion">
      <xsd:attribute name="clasificado" type="xsd:date"/>
    </xsd:extension>

  </xsd:simpleContent>
</xsd:complexType>
```

En la declaración anterior una vez descrito el tipo de dato (simpleType) llamado *formatoclasificacion* se declara un tipo de dato (complexType) llamado *formatonuevo*. Éste último extiende el comportamiento del tipo de dato *formatoclasificacion* agregando un atributo adicional a su estructura. Nótese el uso de los elementos <xsd:simpleContent> y <xsd:extension>.

```
    <xsd:complexType name="residencia">
       <xsd:choice>
         <xsd:element ref="ciudad"/>
         <xsd:element ref="municipio"/>
       </xsd:choice>

    </xsd:complexType>


<xsd:complexType name="residenciacompleta">
  <xsd:complexContent>

    <xsd:extension base="residencia">
      <xsd:sequence>
        <xsd:element name="direccion" type="xsd:string"/>
        <xsd:element name="pais" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>

  </xsd:complexContent>

</xsd:complexType>
```

El fragmento anterior define un tipo de dato (complexType) llamado *residencia*, para posteriormente extender su funcionamiento a través del tipo de dato *residenciacompleta* agregando una secuencia adicional (nótese el uso de los elementos <xsd:complexContent> y <xsd:extension>).

PATRONES

Otras posibles expresiones regulares

\d → Dígito

\D → Cualquier carácter no numérico

\s → Carácter blanco: espacio, tabulador, salto de línea…

\p{Expr Unicode} → Expr puede ser Lu, IsGreek…

\P{Expr Unicode} → Negación de Expr Unicode

A?B → AB | B

[abc]x → ax | bx | cx

\w → Letra o dígito

\W → Cualquier carácter diferente de letra o dígito

. → Cualquier caracter

(Fuente: http://www.datypic.com/books/defxmlschema/chapter10.html)

Table 10-1. Using normal characters

Regular expression	Matching strings	Non-matching strings
a	a	b
a|b|c	a, b, c	abc

Table 10-3. Using character entity references

Regular expression	Matching strings	Non-matching strings
a z	a z	az
a *z	az, a z, a z, a z	a *z
PB&J	PB&J	PBJ

Table 10-5. Using single-character escapes

Regular expression	Matching strings	Non-matching strings
1+2	12, 112, 1112	1+2
1\+2	1+2	12, 1\+2
1\++2	1+2, 1++2, 1+++2	12

Table 10-7. Using multi-character escapes

Regular expression	Matching strings	Non-matching strings
a\dz	a0z, a1z	az, adz, a12z
a.z	a1z, axz	az, axxz

Table 10-9. Using category escapes

Regular expression	Matching strings	Non-matching strings
\p{Lu}	A, B, C	a, b, c, 1, 2, 3
\P{Lu}	a, b, c, 1, 2, 3	A, B, C
\p{Nd}	1, 2, 3	a, b, c, A, B, C
\P{Nd}	a, b, c, A, B, C	1, 2, 3

Table 10-11. Using block escapes

Regular expression	Matching strings	Non-matching strings
\p{IsBasicLatin}	a, b, c	â, ß, ç
\P{IsBasicLatin}	â, ß, ç	a, b, c

Table 10-12. Specifying a list of characters

Regular expression	Matching strings	Non-matching strings
[abc]z	az, bz, cz	abz, z, abcz, abc
[\p{Lu}\d]z	Az, Bz, 1z, 2z	az, bz, cz, A1z

Table 10-13. Specifying a range

Regular expression	Matching strings	Non-matching strings
[a-f]z	az, fz	z, abz, gz, hz
[0-9a-fA-F]z	1z, az, Bz	z, gz, Gz, 1aBz
[-&#x007F;]z	az, bz, cz	âz

Table 10-14. Combining characters and ranges

Regular expression	Matching strings	Non-matching strings
[0-9pqr]z	1z, 2z, pz, rz	cz,dz, 0sz
[p-r\d]z	1z, 2z, pz, rz	cz,dz, 0sz

Table 10-15. Negating a character class expression

Regular expression	Matching strings	Non-matching strings
[^ab]z	cz, dz, 1z	az, bz
[^\d]z	az, bz, cz	1z, 2z, 3z
[^1-3a-c]z	dz, 4z	1z, az

Table 10-16. Subtracting from a character class expression

Regular expression	Matching strings	Non-matching strings
[a-z-[c]]z	az, dz, ez, zz	cz
[a-z-[cd]]z	az, ez, zz	cz, dz
[a-z-[c-e]]z	az, zz	cz, dz, ez, 1z
[^a-z-[123]]z	4z	az, 3z, zz

Table 10-17. Using parenthesized regular expressions

Regular expression	Matching strings	Non-matching strings
(a|b)z	az, bz	abz, z
(ab)*z	z, abz, ababz	az, bz, aabbz
(a+b)*z	z, abz, aabz, abaabz	az, abbz
([a-f]x)*z	z, axz, bxfxfxz	gxz, xz
(\db)*z	z, 1bz, 1b2bz	1z, bz

Regular expression	Matching strings	Non-matching strings
abz	abz	az, abbz
ab?z	az, abz	abbz
ab*z	az, abz, abbz, abbbz, ...	a1z
ab+z	abz, abbz, abbbz, ...	az
ab{2}z	abbz	abz, abbbz
ab{2,}z	abbz, abbbz, abbbbz, ...	az, abz
ab{2,3}z	abbz, abbbz	az, abz, abbbbz
a[b-d]+z	abz, abdz, addbccdddz	az, aez, abez
a\p{Nd}+z	a1z, a11z, a234z	az, abcz
a(bc)+z	abcz, abcbcz, abcbcbcz, ...	az, abz

ANEXO

Especificar elementos vacíos

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

<Libro xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ej3_clase.xsd"/>
```