# Tema 04: XSD

Hasta el momento se ha hablado de cómo generar un lenguaje XML con la definición de las reglas que lo componen (DTD). Otra manera de formalizar esas reglas es con un lenguaje llamado XML Shema (también llamado XSD o **X**ML **S**chema **D**efinition). Se puede decir que un XSD, es la evolución natural de un DTD. Permite expresar con mayor potencia, gramáticas más complejas utilizando la misma sintaxis de XML. Nació en el 1998 y se recomendó el uso en el 2001 por el W3C (World Wide Web Consortium). Las características principales de un esquema son las siguientes:

- Define qué elementos pueden aparecer en un documento XML.
- Define qué atributos pueden aparecer en un documento XML.
- Define qué elementos son compuestos, indicando qué elementos hijos deben aparecer y en qué orden.
- Define qué elementos pueden ser vacíos o que pueden incluir texto asociado.
- Define los tipos que pueden utilizarse en cada elemento o atributo.
- Define la obligatoriedad, la optatividad de elementos y/o atributos.

**¿Qué diferencia un DTD de un esquema XSD si sirven para lo mismo?**

La principal ventaja de XSD es que al estar basado en XML, es fácilmente extensible a las futuras modificaciones o necesidades que se identifiquen. Permite definir de manera muy clara los tipos de datos y los espacios de nombres que se soportan.

Ejemplo de documento XSD:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="note">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="to" type="xs:string"/>
       <xs:element name="from" type="xs:string"/>
       <xs:element name="heading" type="xs:string"/>
       <xs:element name="body" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Como podemos ver, el propio documento XSD está también escrito en XML.

Los XML schemas son documentos XML. Así, todo esquema debe comenzar con una declaración XML: esta es la primera línea del documento.

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

En la segunda línea encontramos la declaración del elemento esquema:

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

El elemento "**schema**" incluye el atributo **xmlns** (XML NameSpace), que especifica el espacio de nombres para el esquema. La sintaxis genérica del atributo xmlns es la siguiente:

```
xmlns:prefijo ="URI_DE_UN_ESPACIO_DE_NOMBRE"
```

En el ejemplo, el formato xmlns:xs indica que todos los elementos o atributos que lleven el prefijo "**xs:**" pertenecen al espacio de nombres especificado en la URI (http://www.w3.org/2001/XMLSchema). Los prefijos se utilizan para distinguir entre diferentes espacios de nombres. Se puede utilizar cualquier prefijo, siempre que se especifique el espacio de nombres XML asociado. Si el esquema referencia un único espacio de nombres, por ejemplo, si sólo utiliza elementos y atributos definidos en la especificación XML Schema, no es obligatorio usar el prefijo. La declaración del elemento esquema en este caso podría quedar así:

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

En este caso no sería necesario usar el prefijo "xs:" delante de todos los elementos y atributos del esquema.

**Otro ejemplo:**

```
<?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>
```

Dentro del elemento "schema" se encuentran todas las declaraciones de elementos y atributos que pueden incluirse en los ejemplares XML que utilicen esta definición de esquema, que se denominan documentos instancia. Así, en el ejemplo de arriba, el elemento raíz en las instancias XML para este schema se llama "**Libro**" y tiene tres elementos hijo y un atributo. Los hijos son "**Título**", "**Editorial**", que deben aparecer ambos una vez, y "**Autores**" que puede aparecer de una a

diez veces. El hecho de que estén agrupados en una secuencia (***<xsd:sequence>***)indica que los elementos deben aparecer en ese orden, es decir, primero el “Título”, luego los “Autores” y por último la “Editorial”. Los tres elementos son de tipo string (cadena de caracteres). El atributo de libro se llama “**precio**” y es de tipo “double”.

---

### Instancia XML. 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>
```

Vemos que la referencia al XML schema desde el ejemplar XML se hace desde dentro de la etiqueta de inicio del elemento raíz, donde se especifican dos cosas:

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

# Espacios de nombres

## ¿Qué es el espacio de nombres?.

Un espacio de nombres XML es una recomendación W3C para proporcionar elementos y atributos con nombre único en una instancia XML. Una instancia XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML. Si a cada uno de estos vocabularios se les da un espacio de nombres, se resuelve la ambigüedad existente entre elementos o atributos que se llamen igual. Los nombres de elementos dentro de un espacio de nombres deben ser únicos.

Un espacio de nombres XML no necesita que su vocabulario sea definido, aunque es una buena práctica utilizar un DTD o un esquema XML para definir la estructura de datos en la ubicación URI del espacio de nombres.

## ¿Cómo se declara el uso de un de espacio de nombres?.

Para definir un espacio de nombres al que pertenece un elemento se usa, dentro de la etiqueta de inicio de ese elemento, el atributo XML reservado “xmlns”, cuyo valor debe ser un identificador uniforme de recurso (URI)∗, que es el nombre del espacio de nombres. El formato genérico de una declaración de un espacio de nombres es el siguiente:

```
xmlns(:<prefijo>)?=<nombre_del_espacio_de_nombres(URI)>
```

El prefijo, que es opcional, es un alias que identifica al espacio de nombres. El prefijo de un espacio de nombres es totalmente arbitrario; lo que define e identifica un espacio de nombres es, en realidad, el URI. Hay que destacar que el URI no se interpreta realmente como una dirección; se trata como una cadena de texto por el analizador XML. Por ejemplo:

```
xmlns:xhtml="http://www.w3.org/1999/xhtml"
```

Se declara el espacio de nombres “http://www.w3.org/1999/xhtml”, al que se le asigna el alias “xhtml”. Todos los elementos que incluyan el prefijo “xhtml” antes del nombre pertenecerán a este espacio de nombres. Insistimos en que los URI sólo se utilizan para que el nombre sea único, no son enlaces, ni tienen que contener información. En el ejemplo anterior el propio “http://www.w3.org/1999/xhtml” no contiene código alguno, simplemente describe el espacio de nombres XHTML a lectores humanos. El hecho de usar una URL (tal como "http://www.w3.org/1999/xhtml") para identificar un espacio de nombres, en lugar de una simple cadena (como "xhtml"), reduce la posibilidad de que diferentes espacios de nombres usen identificadores iguales. Los identificadores de los espacios de nombres no necesitan seguir las convenciones de las direcciones de Internet, aunque a menudo lo hagan. Un URI es una cadena corta de caracteres que identifica inequívocamente un recurso, normalmente accesible en una red o sistema. Un URI (Uniform Resource Identifier) se diferencia de un URL en que permite incluir en la dirección una subdirección, para identificar un fragmento del recurso principal.

## ¿Cómo referenciamos un espacio de nombres con prefijo?.

Una vez que se ha declarado un espacio de nombres y se le ha asignado un alias o prefijo, para indicar

que un elemento pertenece a dicho espacio de nombres se antepone dicho alias seguido del carácter dos juntos ":" como prefijo al nombre del elemento. Dicho prefijo debe aparecer tanto en la etiqueta de inicio como de cierre del elemento. El alcance de la declaración de un prefijo de espacio de nombres comprende desde la etiqueta de inicio de un elemento XML, en la que se declara, hasta la etiqueta final de dicho elemento XML. Se considera que la declaración de espacios de nombres es aplicable al elemento en que se especifica y a todos los elementos dentro del contenido de ese elemento, a menos que sea anulado por otra declaración de espacio de nombres. En las etiquetas vacías, correspondientes a elementos sin "hijos", el alcance es la propia etiqueta.

## Declaración de múltiples espacios de nombres.

Se pueden declarar múltiples espacios de nombres para un mismo elemento, usando varias veces el atributo "xmlns", una por cada espacio de nombres que queremos declarar. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<lb:libro xmlns:lb="urn:loc.gov:libros"
          xmlns:isbn="urn:ISBN:0-395-36341-6">
       <lb:titulo>Más barato que una Docena</lb:titulo>
       <isbn:numero>1568491379</isbn:numero>
</lb:libro>
```

En este ejemplo se declaran dos espacios de nombres para el elemento "libro", que son:

- "urn:loc.gov:libros" al que se asocia el prefijo "lb": el elemento "titulo" y el propio elemento "libro" pertenecen a este espacio de nombres.
- "urn:ISBN:0-395-36341-6" asociado al prefijo isbn: a este espacio de nombres pertenece el elemento "numero".

## El espacio de nombres por defecto.

Cuando la declaración del espacio de nombres no especifica el carácter dos puntos y el alias del espacio de nombres, se está definiendo un espacio de nombres por defecto. Un espacio de nombres por defecto se aplica al elemento donde está declarado (si ese elemento no posee prefijo de espacio de nombres), y a todos los elementos sin prefijo dentro del contenido del ese elemento. El efecto de dicha declaración es anular cualquier declaración de nivel superior del espacio de nombres por defecto, estableciendo su valor a "nulo".
Los espacios de nombres por defecto no se aplican directamente a los atributos. El espacio de nombres de un atributo sin prefijo es una función del tipo del elemento al cual está adjunto, y al espacio de nombres (si hubiese alguno) de ese elemento. Si la referencia URI de la declaración de un espacio de nombres por defecto está vacía (el valor del atributo "xmlns" es una cadena vacía), entonces se considera que los elementos sin prefijo pertenecientes al ámbito de la declaración no están en ningún espacio de nombres. Esto tiene el mismo efecto, dentro del ámbito de la declaración, que si no hubiera espacio de nombres por defecto. En el siguiente ejemplo, los elementos no prefijados ("libro" y "titulo") pertenecen al espacio de nombres por defecto ("urn:loc.gov:libros").

```
<?xml version="1.0" encoding="UTF-8"?>
<libro xmlns="urn:loc.gov:libros"
       xmlns:isbn="urn:ISBN:0-395-36341-6">
```

```
        <titulo>Más barato que una Docena</titulo>
        <isbn:numero>1568491379</isbn:numero>
</libro>
```

# Declaración de elementos en XML Schema XSD

**Cómo se declaran elementos:**

Todos los elementos que se vayan a usar en el ejemplar XML tienen que declararse en el esquema. Las declaraciones de elementos en XML Schema tienen esta sintaxis:

```
<xsd:element name="nombreElemento"
      type="tipoSimple/tipoComplejo"
      minOccurs="valor"
      maxOccurs="valor"
      fixed=”valor"
      default=”valor”/>
```

- **name**: es el nombre del elemento
- **type**: el tipo de elemento. XML Schema define dos tipos de elementos:

  **Tipos simples**: son elementos que sólo pueden contener datos carácter; no pueden incluir otros elementos ni tampoco atributos.

  Ejemplo:

  ```
  <xsd:element name=”fecha” type=”xsd:date”/>
  ```

  Declaramos un elemento llamado “fecha”, de tipo “date” (el prefijo xsd: indica que este tipo de datos “date” es parte del vocabulario de XML Schema).

  **Tipos complejos**: estos elementos pueden incluir otros elementos y/o atributos.
  El contenido de estos elementos se define entre la etiqueta de inicio

  ```
  <xsd:complexType> y la correspondiente de cierre
  </xsd:complexType>.

  Ejemplo:

      <xsd:element name=”libro”>

        <xsd:complexType>

          <xsd:attribute name=”formato”
  type=”xsd:string” use=”required”/>

        </xsd:complexType>

      </xsd:element>
  ```

  Declaramos el elemento “libro”, que es de tipo complejo. Entre la etiquetas <xsd:complexType> y la de cierre </xsd:complexType> se especifica la definición de este elemento. En este caso se trata de un elemento que incluye un atributo “formato”, que es de tipo string, y que es obligatorio.

- **minOccurs y maxOccurs** (Opcionales) : estos dos atributos indican el mínimo (minOccurs) y máximo (maxOccurs) número de ocurrencias del elemento. El valor por defecto para ambos atributos es 1. Si se quiere indicar que el elemento puede aparecer un número ilimitado de veces, el atributo maxOccurs tomará el valor “unbounded”.
- **fixed** (Opcional): especifica un valor fijo para el elemento.
- **default** (Opcional): especifica un valor por defecto para el elemento.

A diferencia de lo que ocurre con los valores de atributos, los valores “fixed” y “default” de los elementos sólo se añaden al documento XML si el elemento está presente.

**Modelos de contenido para elementos**

El contenido de un elemento se define mediante un modelo de contenido. En XML Schema existen cuatro modelos de contenido para elementos:

1.- **Texto**: el elemento sólo puede contener datos carácter.

Ejemplo: veamos una definición de un elemento que sólo contiene texto (usamos el tipo “string”, que representa una secuencia de caracteres”).

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

2.- **Vacío:** el elemento no puede contener datos carácter ni otros sub-elementos, pero sí puede incluir atributos. En este caso hay que declararlos como tipos complejos. Si no contienen atributos pueden declararse como tipos simples.

Ejemplo: declaramos el elemento “antiguedad”, que es de contenido vacío (no contiene ni texto ni otros sub-elementos), y que es de tipo complejo porque contiene un atributo, “anyosDeServicio”, que es de tipo “positiveInteger” (entero positivo):

```
<xsd:element name=”antiguedad”>
    <xsd:complexType>
        <xsd:attribute name=”anyosDeServicio”
type=”xsd:positiveInteger”/>
    </xsd:complexType>
</xsd:element>
```

3.- **Elementos**: el elemento puede contener dentro sub-elementos. Existen tres formas de especificar los sub-elementos dentro del elemento, mediante tres tipos de elementos predefinidos en XML Schema: "sequence”, “choice” y “all”.

- **El elemento <xsd:sequence>**. Utilizamos este elemento para indicar una secuencia de elementos que tienen que aparecer en el documento XML. Deben aparecer todos, y en el mismo orden en que se especifican.

  Ejemplo:

```
<xsd:element name="camiseta">
   <xsd:complexType>
      <xsd:sequence>
```

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

- **El elemento <xsd:choice>**. Especifica una lista de elementos de los cuales sólo puede aparecer uno en el documento XML.

  Ejemplo:

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

  El elemento “choice” puede incluir opcionalmente los atributos minOccurs y maxOccurs, para especificar el mínimo y máximo número de elementos hijos que pueden incluirse en el documento.

- **El elemento <xsd:all>**. Se comporta igual que el elemento <xsd:sequence>, pero no es obligado que en el documento XML aparezcan todos los elementos especificados, ni en el mismo orden.

  Ejemplo:

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

4.- **Mixto (Mixed)**: el elemento puede contener tanto datos carácter como elementos hijo. Los elementos hijo se definen igual que en el modelo anterior, mediante los elementos “sequence”, “choice” o “all”. Para indicar que el elemento puede además incluir datos carácter se usa el atributo “mixed” con valor igual al “true” en el elemento “complexType”.

Ejemplo:

```
<xsd:element name="confirmacionPedido">
<xsd:complexType mixed=”true”>
<xsd:sequence>
   <xsd:element name=”intro”  type=”xsd:string”/>
```

```
      <xsd:element name=”nombre” type=”xsd:string”/>
      <xsd:element name=”fecha”  type=”xsd:string”/>
      <xsd:element name=”titulo” type=”xsd:string”/>
    <xsd:sequence>
    </xsd:complexType>
    </xsd:element>
```

El elemento definido arriba podría usarse dentro de un documento XML de la siguiente manera:

```
<confimacionPedido>

    <intro>Para:</intro>

    <nombre>Antonio Lara</nombre>Confirmamos que con fecha
      <fecha>24-01-2005</fecha> hemos recibido su pedido de
<titulo>Raices</titulo>. Su título será enviado en 2 días
hábiles desde la recepción del pedido. Gracias,Ediciones
Aranda.

</confimacionPedido>
```

## Referencias a otros elementos

En un schema es posible declarar elementos de forma global y luego hacer referencias a ellos desde otros elementos. La principal ventaja de esto es que permite reutilizar una misma definición de un elemento en varios sitios del schema. Otra ventaja de este mecanismo es que puede ayudar a que los schema estén mejor estructurados y sean más fácilmente legibles. Para referenciar a un elemento se utiliza el atributo **“ref”** cuyo valor es el nombre del elemento referenciado, en lugar del atributo “name” seguido de la definición del elemento. Vamos a ver cómo quedaría el ejemplo utilizando referencias a elementos:

```
<?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 ref="Título"/>
                <xsd:element ref="Autores"/>
                <xsd:element ref="Editorial"/>
            </xsd:sequence>
            <xsd:attribute name="precio" type="xsd:double"/>
          </xsd:complexType>
     </xsd:element>
     <xsd:element name="Título" type="xsd:string"/>
     <xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
     <xsd:element name="Editorial" type="xsd:string"/>
</xsd:schema>
```

Los tipos de datos se dividen en tipos de datos simples (o primitivos) y tipos de datos complejos. Los tipos de datos simples se pueden utilizar en los valores de los atributos y en los elementos que contienen sólo datos carácter. Existe una serie de tipos de datos definidos en el estándar y que por tanto se pueden usar directamente en los esquemas. Además de estos, el usuario puede definir sus propios tipos de datos, tanto simples como complejos.

Existen 19 tipos de datos simples predefinidos primitivos, que se pueden agrupar en 4 categorías:

- **Tipos cadena**
  - **string**: secuencia de longitud finita de caracteres*
  - **anyURI**: una uri estándar de Internet
  - **NOTATION**: declara enlaces a contenido externo no-XML
  - **Qname**: una cadena legal Qname (nombre con cualificador)
- **Tipos binario codificado**
  - **boolean**: toma los valores “true” o “false” *
  - **hexBinary**: dato binario codificado como una serie de pares de dígitos hexadecimales
  - **base64Binary**: datos binarios codificados en base 64
- **Tipos numéricos**
  - **decimal**: número decimal de precisión (dígitos significativos) arbitraria *
  - **float**: número de punto flotante de 32 bits de precisión simple *
  - **double**: número de punto flotante de 64 bits de doble precisión *
- **Tipos de fecha/hora**
  - **duration**: duración de tiempo
  - **dateTime**: instante de tiempo específico, usando calendario gregoriano, en formato "YYYYMM-DDThh:mm:ss"
  - **date**: fecha específica del calendario gregoriano, en formato "YYYY-MM-DD” *
  - **time**: una instancia de tiempo que ocurre cada día, en formato "hh:mm:ss"
  - **gYearMonth**: un año y mes del calendario gregoriano
  - **gYear**: año del calendario gregoriano
  - **gMonthDay**: día y mes del calendario gregoriano
  - **gMonth**: un mes del calendario gregoriano
  - **gDay**: una fecha del calendario gregoriano (día)

# Declaración de atributos en XML Schema XSD

### ¿cómo se declaran los atributos?

La sintaxis genérica de declaración de atributos es la siguiente:

```
<xsd:attribute name="nombreAtributo"
               type="tipoSimple"
               use="valor"
               default="valor"
               fixed="valor"/>
```

- **name**: es el nombre del atributo.
- **type**: el tipo del atributo. Los atributos sólo pueden contener tipos simples.
- **use** (Opcional): puede tomar uno de los siguientes valores:
  - required: el atributo debe aparecer en el documento XML.
  - optional: el atributo puede aparecer o no aparecer en el documento XML. Este es el valor por defecto.
  - prohibited: el atributo no debe aparecer en el documento XML.
- **default** (Opcional): si el atributo no aparece en el documento XML, se le asigna el valor especificado en el atributo “default”. Los valores por defecto sólo tienen sentido si el atributo es opcional, de lo contrario tendremos un error.
- **fixed** (Opcional): define un valor fijo para el atributo.
  - si el valor del atributo está presente en la instancia del documento XML, el valor debe ser el mismo que el que indica el atributo “fixed”
  - si el atributo no está presente en el documento XML, se le asigna el valor contenido en el atributo “fixed”

Los valores de los atributos “default” y “fixed” son mutuamente exclusivos, por lo tanto habrá un error si una declaración contiene ambos.
Recordemos que sólo los elementos de tipo compuesto pueden contener atributos. Las declaraciones de atributos para un elemento deben aparecer siempre al final del bloque delimitado por la etiqueta de inicio <complexType> y la de fin </complexType>, después de las especificaciones de todos los demás componentes. Veamos un ejemplo de un elemento que contiene un atributo:

```
<xsd:element name="cliente">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="nombre" type="xsd:string"/>
    <xsd:element name="apellido" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="numCliente" type="xsd:positiveInteger"/>
</xsd:complexType>
</xsd:element>
```

# Declaración de elementos propios

XML Schema permite al usuario la creación de sus propios tipos de datos. Existen varias opciones para hacer esto.

**Crear un tipo complejo con un nombre.**

La forma más sencilla de crear un nuevo tipo es crear un elemento complexType al que se le asigna un nombre mediante el atributo “name”.

Ejemplo:

```
<xsd:complexType name=”precioInfo”>
     <xsd:sequence>
          <xsd:element ref=”precioTipo”/>
          <xsd:element ref=”precioN”/>
     </xsd:sequence>
</xsd:complexType>
<xsd:element name=”precioTipo” type=”xsd:string”/>
<xsd:element name=”precioN” type=”xsd:decimal”/>
```

Con esto creamos un tipo nuevo llamado “precioInfo” que está formado por una secuencia de dos elementos, “precioTipo” y “precioN”, a los que referencia.

Podremos utilizar este tipo que hemos creado para definir elementos, por ejemplo:

```
<xsd:element name=”precio” type=”precioInfo”/>
```

Observamos que en la definición del elemento precio, el nombre del tipo “precioInfo” no lleva el prefijo “xsd”, porque no pertenece al espacio de nombres del estándar XML Schema. La principal ventaja de definir tipos de datos propios es, por un lado, que estos tipos se pueden utilizar donde se quiera, y además, que estos tipos se pueden utilizar como tipos base para definir otros tipos. A continuación vamos a ver los mecanismos que existen para definir tipos derivados de otros.

### Crear un tipo por restricción de un tipo simple predefinido (xsd:restriction)

La forma más sencilla de crear un nuevo tipo a partir de uno ya existente es añadir condiciones a alguno de los tipos predefinidos en el XML Schema. Esto se hace con el elemento <xsd:restriction>.

Por ejemplo:

```
<xsd:simpleType name="monedaUSA">
     <xsd:restriction base="xsd:decimal">
     <xsd:fractionDigits value="2"/>
     </xsd:restriction>
</xsd:simpleType>
```

En este ejemplo creamos un nuevo tipo simple y le asignamos el nombre “monedaUSA”. Mediante el uso del elemento “xsd:restriction” definimos el tipo “monedaUSA” como un subtipo del tipo base “xsd:decimal”, en el que el número de cifras decimales es 2 (xsd:fractionDigits value=”2”). Las restricciones (también llamadas facetas) que pueden aplicarse a los tipos simples son:

- **minInclusive**: mínimo valor que puede tomar el número; por ejemplo, si mininclusive vale 5, el número tiene que ser mayor o igual que 5.

- **minExclusive**: el número debe ser mayor que este valor; por ejemplo, si minExclusive vale 5, el número debe ser mayor que 5.
- **maxInclusive**: máximo valor que puede tomar el número; por ejemplo, si maxinclusive vale 5, el número tiene que ser menor o igual que 5.
- **maxExclusive**: el número debe ser menor que este valor; por ejemplo, si maxExclusive vale 5,el número debe ser menor que 5.
- **totalDigits**: total de cifras en el número, incluyendo las enteras y las decimales.
- **fractionDigits**: número de cifras decimales.
- **length**: número de unidades del valor literal; para la mayoría de los tipos (por ejemplo los tipos “string” y sus derivados, la faceta “length” se refiere a caracteres, pero para listas (que veremos más adelante) se refiere al número de elementos en la lista, y para valores binarios se refiere al número de octetos. No se puede aplicar a los tipos “integer”, “float” o “double” (para estos se puede utilizar “totalDigits”).
- **minLength** y **maxLength**: valor mínimo y máximo respectivamente para la faceta “length”.
- **pattern**: formato que debe tener el valor, especificado mediante una expresión regular tradicional.
- **enumeration**: conjunto de posibles valores que puede tomar el dato.
- **whiteSpace**: controla la forma que tendrá el contenido de este dato una vez haya sido procesado; puede tomar los siguientes valores:
  - “preserve”: los datos no se modifican, quedan tal y como aparecen escritos.
  - “replace”: los tabuladores, saltos de línea y retornos de carro son sustituidos por espacios.
  - “collapse”: hace lo mismo que “replace”, pero además sustituye espacios múltiples por un solo espacio.

Estas facetas se pueden combinar, y se pueden usar tanto en las definiciones de tipos con nombre como en las definiciones de elementos. Veamos algunos ejemplos, en los que las facetas se aplican a definiciones de elementos.
Ejemplo 1: facetas “minInclusive” y “maxInclusive”.

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

Define el elemento “edad” de tipo simple, como un entero en el rango [0-120].

Ejemplo 2: facetas “minLength” y “maxLength”.

```
<xs:element name="contraseña">
  <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
          <xs:minLength value="5"/>
          <xs:maxLength value="8"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

Define el elemento “contraseña” como una cadena de entre 5 y 8 caracteres.

### Crear un tipo enumerado (xsd:enumeration)

Una forma muy útil de utilizar facetas es crear tipos enumerados para limitar los valores que puede tomar un elemento o atributo.
Ejemplo:

```
<xsd:attribute name="demanda">
  <xsd:simpleType>
    <xs:restriction base="xsd:string">
      <xs:enumeration value="bajo"/>
      <xs:enumeration value="medio"/>
      <xs:enumeration value="alto"/>
    </xs:restriction>
  </xs:simpleType>
</xsd:attribute>
```

Definimos el atributo “demanda” como una restricción del tipo base “xsd:string”, donde sólo existen tres valores posibles: “alto”, “medio” y “bajo”. El atributo “demanda” debe tomar uno de estos tres valores.

### Listas (xsd:list)

Las listas son similares a la faceta “enumeration”, pero permiten incluir valores múltiples, separados mediante espacios. Las listas permiten elegir el tipo de datos de los valores en la lista, mediante el atributo "itemType”, incluyendo tipos definidos por el usuario. También se pueden aplicar otras facetas, como “length”, etc.
Ejemplo:

```
<xsd:simpleType name="posiblesTiendas">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="oBoy"/>
        <xsd:enumeration value="Yoohoo!"/>
        <xsd:enumeration value="ConJunction"/>
        <xsd:enumeration value="Anazone"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="listaTiendas">
    <xsd:list itemType="posiblesTiendas"/>
</xsd:simpleType>
<xsd:simpleType name="tiendas">
    <xsd:restriction base="listaTiendas">
    <xsd:maxLength value="3"/>
    </xsd:restriction>
</xsd:simpleType>
```

Se define el tipo “tiendas” como una restricción del tipo “listaTiendas” donde “legnth” no puede ser mayor que 3. Es decir, en “tiendas” puede haber hasta 3 valores de la lista. El tipo “listaTiendas” se define como una lista donde cada elemento de la lista es del tipo “posiblesTiendas”. Este último se define como un “enumeration”. Podríamos por ejemplo definir elemento llamado “vendedores” de la siguiente manera:

```
<xsd:element name="vendedores" type="tiendas"/>
```

Este elemento puede tomar hasta tres valores de los que se han especificado en el tipo “posiblesTiendas”. Por ejemplo, en un documento instancia XML podríamos encontrarnos algo como:

```
<tiendas>oBoy Yoohoo!</tiendas>
```

## Añadir atributos a tipos simples por extensión (xsd:extensión).

Sabemos que los elementos de tipos simples son los que sólo pueden contener datos carácter, pero no atributos ni elementos hijo. Por otro lado, los elementos de tipo complejo pueden tener tanto atributos como elementos hijo. ¿Qué pasa si queremos que un elemento pueda tener atributos, pero no elementos hijo? Existe una forma de hacer esto, utilizando los elementos “xsd:simpleContent” y “xsd:extension”.
Veámoslo con un ejemplo:

```
<xsd:element name="nombreOriginal">
<xsd:complexType>
    <xsd:simpleContent>
        <xsd:extension base="xsd:string">
            <xsd:attribute name="confirmado" default="no"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
</xsd:element>
```

En este ejemplo definimos el elemento “nombreOriginal” de tipo complejo, pero al usar el elemento <xsd:simpleContent> estamos diciendo que el contenido de “nombreOriginal” tiene que ser sólo datos carácter. Sin embargo, este elemento sí tiene un atributo. Esto se especifica definiendo el elemento “xsd:simpleContent” como una extensión del tipo base “xsd:string” a la que se le añade el atributo “confirmado”, cuyo valor por defecto es “no”. También es posible definir tipos derivados de los tipos complejos definidos por el usuario, utilizando los mismos mecanismos que hemos visto en esta sección.

# Métodos de diseño XSD XML Schema

Hay varias maneras de abordar el diseño de un XML schema. Usaremos una u otra, o una combinación de varias, dependiendo de factores tales como la complejidad, extensión y el tipo de documentos que estamos definiendo (por ejemplo, sin son documentos donde predomina una colección de datos estructurados, o son documentos con mucho texto libre). A continuación se describen tres formas de diseñar XML schemas.

- **Diseño Anidado o de muñecas rusas.**. Se llama así porque se anidan declaraciones de elementos unas dentro de otras. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Consiste en, partiendo de un documento XML, seguir la estructura del mismo e ir definiendo los elementos que aparecen en el mismo de forma secuencial, incluyendo la definición completa de cada elemento en el mismo orden en el que aparecen en el documento instancia. Este método de diseño es muy sencillo, pero puede dar lugar a esquemas XML difíciles de leer y mantener cuando los documentos son complejos. Además produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones. Es el método de diseño menos recomendable.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http;//www.w3.org/2001/XMLSchema">
    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="titulo" type="xs:string"/>
                <xs:element name="autor" type="xs:string"/>
                <xs:element name= "personaje" minOccurs= "O"  maxOccurs =" unbounded" >
                   <xs:complexType>
                        <xs:sequence>
                            <xs:element name="nombre" type="xs:string"/>
                            <xs:element name="amigoDe" type= "xs : string" minOccurs="O"
maxOccurs="unbounded"/>
                            <xs:element name="desde" type="xs:date"/>
                            <xs:element name="calificación" type="xs:string"/>
                        </xs:sequence>
                   </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- **Diseño Plano o de uso de referencias a elementos y atributos.** En este método primero se definen los diferentes elementos y atributos, para después referenciarlos utilizando el atributo “ref”. La declaración y su definición están en diferentes sitios. Esta técnica es la que más se parece al diseño con DTDs. En el ejemplo que mostramos a continuación se comienza a definir los elementos más simples para terminar con los más complejos, pero puede hacerse al revés (como en los DTDs) comenzando por definir los elementos de mayor nivel y, descendiendo, ir definiendo los elementos de menor nivel hasta llegar a los elementos simples.

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

    <!-- definición de elementos de tipo simple-->
```

```
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="autor" type="xs:string"/>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="amigoDe" type="xs:string"/>
    <xs:element name="desde" type= "xs:date"/>
    <xs:element name="calificacion" type="xs:string"/>

    <!-- definición de atributos -->
    <xs:attribute name="isbn" type="xs:string"/>

    <!-- definición de elementos de tipo complejo -->
    <xs:element name="personaje">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nombre"/>
                <xs:element ref="amigoDe" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="desde"/>
                <xs:element ref="calificación"/>
                <!-- los elementos de tipo simple se referencian ref -->
                <!-- la definición de cardinalidad en elemento referenciado-->
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="titulo"/>
                <xs:element ref="autor"/>
                <xs:element ref="personaje" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="isbn"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- **Diseño con tipos con nombre.** Con este método se definen clases o tipos utilizando los elementos de XML Schema “simpleType” y “complexType” con un nombre. Estos tipos definidos se pueden utilizar mediante el tributo “type” de los elementos. Este mecanismo permite reutilizar de definiciones de tipos en diferentes puntos del documento. Es el método más aconsejado ya que permite la reutilización. En el ejemplo, al igual que en el caso anterior, se ha seguido un diseño ascendente (definiendo primero el de menor nivel) pero se puede hacer un diseño descendente.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- definición de tipos simples-->
<xs:simpleType name="TipoNombre" >
    <xs:restriction base="xs:string">
        <xs:raaxLength value="32"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TipoDesde">
    <xs:restriction base="xs:date"/>
</xs:simpleType>
<xs:simpleType name= "TipoDescripcion" >
```

```
            <xs:restriction base="xs:string"/>
        </xs:simpleType >
        <xs:simpleType name="TipoISBN">
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{13}"/>
            </xs:restriction>
        </xs:simpleType>

        <!-- definición of tipos complejos -->
        <xs:complexType name="TipoPersonaje">
            <xs:sequence>
                <xs:element name="nombre" type="TipoNombre"/>
                <xs:element name="amigoDe" type="TipoNombre" minOccurs = "0"
      maxOccurs="unbounded"/>
                <xs:element name="desde" type="TipoDesde"/>
                <xs:element name="calificacion" type="TipoDescripcion"/>
            </xs:sequence>

        </xs:complexType>
        <xs:complexType name="TipoLibro">
            <xs:sequence>
                <xs:element name="titulo" type="TipoNombre"/>
                    <xs:element name="autor" type"TipoNombre"/>
                <xs:element name="personaje" type="TipoPersonaje" minOccurs="O"/>
            </xs:sequence >
            <xs:attribute name="isbn" type="TipoISBN" use="required"/>
        </xs:complexType>

        <!-- definición del elemento raíz libro usando el tipo complejo TipoLibro-->
        <xs:element name="libro" type="TipoLibro"/>
</xs:schema>
```

# Comienzo de un esquema simple.

Un esquema es un documento XML que sólo contiene texto y aparece con la extensión .**xsd**. Comienza con una declaración XML estándar, seguida de una declaración del espacio de nombre de XML esquema. Por ejemplo:

```
<?xml version=”1.0 “ encoding="UTF-8"?>
<xsd:schema xmnls:xsd= http://www.w3.org/2001/XMLSchema>
        ( aquí irán el conjunto de reglas del esquema)
</xsd:schema>
```

Para comenzar un esquema, los pasos a realizar son:

1. Escribe la declaración XML, similar a <?xml version=”1.0 “?>, al comienzo del documento.
2. Escribe <xsd:schema.
3. Escribe xmnls:xsd= “http://www.w3.org/2001/XMLSchema” para declarar el espacio de nombre del “esquema de esquemas”. A partir de ahora cualquier documento o escritura que aparezca con el prefijo xsd: delante será reconocido como si procediera de este espacio de nombre.
4. Escribe > para completar el elemento de apertura.
5. Tras la definición del propio esquema se completa el documento con </xsd:schema>.

Se guarda el esquema como archivo de sólo texto y con extensión .xsd.

## Ejemplo Guía.

Para continuar con la explicación, y para que resulte más sencillo, vamos a ir viendo un ejemplo. Consideremos un documento XML en un fichero llamado **po.xml**.

Describe una hoja de pedido generada por un pedido de productos para el hogar y una aplicación de facturación:

Hoja de Pedido, po.xml

```
<?xml version="1.0"? encoding="UTF-8">
<hojaPedido fechaPedido="2015-10-26">

    <enviarA pais="EEUU">

        <nombre>Alice Smith</nombre>
        <calle>123 Maple Street</calle>
        <ciudad>Mill Valley</ciudad>
        <estado>CA</estado>
        <zip>90952</zip>

    </enviarA>

   <facturarA pais="EEUU">
```

```
            <nombre>Robert Smith</nombre>
            <calle>8 Oak Avenue</calle>
            <ciudad>Old Town</ciudad>
            <estado>PA</estado>
            <zip>95819</zip>

        </facturarA>
        <comentario>¡Deprisa, mi césped parece una selva! </comentario>

        <elementos>

            <elemento numProducto="872-AA">
              <nombreProducto>Cortacesped</nombreProducto>
              <cantidad>1</cantidad>
              <precioEEUU>148.95</precioEEUU>
              <comentario>Confirmar que es eléctrico</comentario>
            </elemento>

            <elemento numProducto="926-AA">
              <nombreProducto>Monitor para bebes </nombreProducto>
              <cantidad>1</cantidad>
              <precioEEUU>39.98</precioEEUU>
              <fechaEnvio>1999-05-21</fechaEnvio>
            </elemento>

        </elementos>
    </hojaPedido>
```

La hoja de pedido consiste de un elemento principal, **hojaPedido**, y los subelementos **enviarA**, **facturarA**, **comentario**, y **elementos**. Estos subelementos (excepto comentario) además contienen otros subelementos, y así, hasta que un subelemento como por ejemplo **precioEEUU** contiene un número en lugar de más subelementos.

## Resumen

Los elementos que contienen subelementos o tienen atributos son denominados tipos complejos, mientras que los elementos que contienen números (o cadenas de caracteres, o fechas, etc.) pero no contienen subelementos ni atributos se denominan tipos simples.

Algunos elementos tienen atributos; los atributos siempre son tipos simples.

Los tipos complejos en el documento instancia, y algunos de los tipos simples, están definidos en el esquema para hojas de pedido. Los demás tipos simples están definidos como parte del repertorio de tipos simples predefinidos del Esquema XML.

El esquema de la hoja de pedido está contenido en el archivo po.xsd:

**El Esquema de la Hoja de Pedido , po.xsd**

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:annotation>
    <xsd:documentation xml:lang="es">
        Esquema de hoja de pedido para Example.com.
   </xsd:documentation>
</xsd:annotation>
<xsd:element name="hojaPedido" type="TipoHojaPedido"/>
<xsd:element name="comentario" type="xsd:string"/>
<xsd:complexType name="TipoHojaPedido">

      <xsd:sequence>
         <xsd:element name="enviarA" type="direccionEEUU"/>
         <xsd:element name="facturarA" type="direccionEEUU"/>
         <xsd:element ref="comentario" minOccurs="0"/>
         <xsd:element name="elementos" type="Elementos"/>
      </xsd:sequence>
      <xsd:attribute name="fechaPedido" type="xsd:date"/>

</xsd:complexType>
<xsd:complexType name="direccionEEUU">

      <xsd:sequence>
        <xsd:element name="nombre" type="xsd:string"/>
        <xsd:element name="calle" type="xsd:string"/>
        <xsd:element name="ciudad" type="xsd:string"/>
        <xsd:element name="estado" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:decimal"/>
      </xsd:sequence>
      <xsd:attribute name="pais" type="xsd:NMTOKEN" fixed="EEUU"/>

</xsd:complexType>
<xsd:complexType name="Elementos">

      <xsd:sequence>
        <xsd:element name="elemento" minOccurs="0" maxOccurs="unbounded">

            <xsd:complexType>

                <xsd:sequence>
                  <xsd:element name="nombreProducto" type="xsd:string"/>
                  <xsd:element name="cantidad">
                  <xsd:simpleType>
                       <xsd:restriction base="xsd:positiveInteger">
                          <xsd:maxExclusive value="100"/>
                      </xsd:restriction>
                 </xsd:simpleType>
                 </xsd:element>
```

```
                <xsd:element name="precioEEUU" type="xsd:decimal"/>
                <xsd:element ref="comentario" minOccurs="0"/>
                <xsd:element name="fechaEnvio" type="xsd:date" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="numProducto" type="SKU" use="required"/>

            </xsd:complexType>

        </xsd:element>
        </xsd:sequence>
        </xsd:complexType>
        <!-- Stock Keeping Unit [Código de Almacenaje], -->
        <!-- un código para identificar productos -->
        <xsd:simpleType name="SKU">
        <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
        </xsd:restriction>
        </xsd:simpleType>

</xsd:schema>
```

## Resumen

El esquema consiste de un elemento schema y una variedad de subelementos, los más notables, element (elemento), complexType (tipo complejo), y simpleType (tipo simple) que determinan la aparición de elementos y su contenido en los documentos instancia (los documentos XML).

Cada uno de los elementos del esquema tiene el prefijo "xsd:" que está asociado con el espacio de nombres de XML Schema. El prefijo xsd: es usado por convención para denotar el espacio de nombres del XML Schema, aunque puede utilizarse cualquier prefijo. El mismo prefijo, y por tanto la misma asociación, también aparece en los nombres de los tipos simples predefinidos, por ejemplo xsd:string. El propósito de la asociación es identificar los elementos y tipos simples como pertenecientes al vocabulario del lenguaje XML Schema y no al vocabulario del autor del esquema.

# Uniendo todo con XML Schemas

En este punto desarrollaremos un ejemplo de principio a fin con todas las explicaciones necesarias para la compresión de los elementos utilizados. El escenario propuesto intenta crear una aplicación de reserva de habitaciones en un hotel. Las reservas vienen por la red, y permiten indicar un conjunto de habitaciones y las personas que se alojarán en éstas; asimismo, se indica el mecanismo mediante el cual se hará el pago de éstas.

### Ejemplo de Hotel

```
<?xml version="1.0" encoding="UTF-8"?>
<reserva version="1">
<habitaciones>
<habitacion numero="202">
  <desde>2015-11-22</desde>
  <hasta>2015-11-24</hasta>
  <residentes>
        <residente responsable="true">
            <nombres>Pedro</nombres>
            <apellido>Camacho</apellido>
           <pasaporte>552321</pasaporte>
           <nacionalidad>PE</nacionalidad>
        </residente>
        <residente>
          <nombres>Julia</nombres>
          <apellido>Urquidi</apellido>
          <pasaporte>664423</pasaporte>
          <nacionalidad>BO</nacionalidad>
        </residente>
   </residentes>
</habitacion>
</habitaciones>
<mediopago>
    <tarjeta_credito>
      <tipo>VISA</tipo>
      <numero>38822050023042</numero>
    </tarjeta_credito>
</mediopago>
</reserva>
```

Asimismo, el sistema del hotel responderá satisfactoriamente así:

### Repuesta satisfactoria del hotel

```
<?xml version="1.0" encoding="UTF-8"?>
<confirmacion version="1">
```

```
    <satisfactorio>true</satisfactorio>
    <comentario>todo conforme</comentario>
</confirmacion>
```

o en caso de no ser satisfactorio el requerimiento, la respuesta será algo como esto:

### Respuesta del hotel errónea

```
<?xml version="1.0" encoding="UTF-8"?>
<confirmacion version="1">
    <satisfactorio>false</satisfactorio>
    <comentario>la habitacion 202 es para una sola persona</comentario>
</confirmacion>
```

Pero estos movimientos ¿están adecuadamente configurados? validaremos los movimientos a través de XML-Schemas

Primero podríamos hacer una validación XML para asegurarnos de que los documentos están bien formados. Ahora construiremos un schema para este archivo. En primer lugar, crearemos "confirmacion.xsd":

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

    ...
</xsd:schema>
```

Los esquemas normalmente se construyen a partir de un conjunto de tags definidos en el "espacio de nombres de esquemas" identificado por el URI que se indicó (el uso de los URIs es una convención, análoga a los nombres de paquete Java). Las herramientas que procesan esquemas ya tienen precargados los tags especiales de este espacio de nombres (en otras palabras, asociados a este URI), por lo que la directiva simplemente los pone a nuestra disposición asociándoles el prefijo "xsd". Esto también es convencional; se podría haber usado: xmlns:otroprefijo="http://www.w3.org..." pero esto sería confuso para quien leyera el schema. Este prefijo debe utilizarse para todos los tags "especiales" del espacio de nombres de esquemas, empezando por el mismo tag "schema", por lo que la declaración con "otroprefijo" tendría que haber sido:

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

En lo que sigue usaremos "xsd" para los tags del espacio de nombres de esquemas.

Ahora bien, el elemento raíz es "**confirmacion**", el cual está constituido por dos elementos ("satisfactorio" y "comentario".) En tanto que "confirmacion" tiene elementos "hijos", entonces se dice que es "complejo", mientras que los otros al no tenerlos, son "simples".

Los elementos simples se especificarán mediante:

```
<xs:element name="satisfactorio" type="xsd:boolean"/>
<xs:element name="comentario" type="xsd:string"/>
```

Como se aprecia, se ha utilizado los tipos "boolean" y "string" del espacio de nombres de esquemas. Ahora bien, se suele exigir que los elementos aparezcan en una secuencia ordenada (aunque no es obligatorio), por lo que podríamos envolverlos en una "secuencia":

```
<xsd:sequence>
  <xsd:element name="satisfactorio" type="xsd:boolean"/>
  <xsd:element name="comentario" type="xsd:string"/>
</xsd:sequence>
```

Todo esto representa el contenido que deberíamos asociar al elemento "confirmación"; en otras palabras, lo anterior define el "tipo" del elemento complejo "confirmación". Los "tipos" pueden tener un nombre, tal como "conf_tipo", y ser definidos así:

```
<xsd:complexType name="conf_tipo">

    <xsd:sequence>
      <xsd:element name="satisfactorio" type="xsd:boolean"/>
      <xsd:element name="comentario" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:integer"/>

</xsd:complexType>
```

Aquí hemos agregado el atributo "**version**" de tipo entero, que es parte del elemento "confirmacion". Con todo esto, "confirmacion" quedará totalmente definido mediante:

```
<xsd:element name="confirmacion" type="conf_tipo"/>
```

## Validación de las confirmaciones. confirmación.xsd

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

<xsd:complexType name="conf_tipo">

    <xsd:sequence>

        <xsd:element name="satisfactorio" type="xsd:boolean"/>

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

    </xsd:sequence>

    <xsd:attribute name="version" type="xsd:integer"/>

</xsd:complexType>

<xsd:element name="confirmacion" type="conf_tipo"/>
```

```
</xsd:schema>
```

Para validar el documento XML, éste requiere hacer referencia al esquema.

---

Esto lo logramos así, documento de confirmación. **confirmacion.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<confirmacion version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="confirmacion.xsd">

    <satisfactorio>true</satisfactorio>
    <comentario>todo conforme</comentario>

</confirmacion>
```

En el elemento principal hemos utilizado el atributo "**noNamespaceSchemaLocation**" para especificar el nombre de nuestro esquema (que deberá ubicarse en el mismo directorio.) Luego se explicará la razón del nombre tan extraño de este atributo.

Es importante anotar que este atributo también pertenece a otro espacio de nombres estandarizado denominado "instancias de esquemas", indentificado con el URI que se aprecia, y asociado a los documentos típicamente con el prefijo "xsi". Nótese que el "xsi" sólo se emplea en los atributos del elemento raíz con este propósito, y raramente aparece en el resto del documento.

Ahora bien, para este esquema concreto podríamos ahorrar algo de texto definiendo el tipo del elemento complejo "confirmacion" al interior de la declaración del mismo de este modo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="confirmacion">
  <xsd:complexType>
    <xsd:sequence>
        <xsd:element name="satisfactorio" type="xsd:boolean"/>
        <xsd:element name="comentario" type="xsd:string"/>
    </xsd:sequence>
   <xsd:attribute name="version" type="xsd:integer"/>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Esto es más eficiente para este caso dado que el tipo "conf_tipo" sólo se utiliza una única vez.

Ahora pasemos a la solicitud de reserva. El elemento raíz es "**reserva**" y contiene los elementos "**habitaciones**" y "**mediopago**" (en dicho orden):

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

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

    <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="habitaciones">
    ...
    </xsd:element>
    <xsd:element name="mediopago">
    ...
    </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:integer"/>
    </xsd:complexType>

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

Si bien nuestro ejemplo sólo introduce una "habitacion", es de esperarse que se puedan reservar varias de una vez (pero al menos una.) Por lo tanto, nuestro elemento "habitaciones" contiene una secuencia de elementos "habitacion" que admite repeticiones; esto se indica agregando el atributo maxOccurs="unbounded" al tag xsd:element:

```
<xsd:element name="habitaciones">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="habitacion" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="desde" type="xsd:date"/>
<xsd:element name="hasta" type="xsd:date"/>
<xsd:element name="residentes">
...
</xsd:element>
</xsd:sequence>
<xsd:attribute name="numero" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Nótese que "habitacion" también resulta ser un elemento complejo conteniendo los elementos "desde", "hasta", "residentes" y el atributo "numero" (de habitación.)

Dado que el elemento "residentes" también es complejo, a fin de no hacer tantos anidamientos, crearemos un tipo explícitamente:

```
<xsd:complexType name="tipo_residentes">
```

```
  <xsd:sequence>
    <xsd:element name="residente" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
          <xsd:element name="nombres" type="xsd:string"/>
        <xsd:element name="apellido" type="xsd:string"/>
        <xsd:element name="pasaporte" type="xsd:string"/>
        <xsd:element name="nacionalidad" type="xsd:string"/>
      </xsd:sequence>
     <xsd:attribute name="responsable" type="xsd:boolean"/>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

Con esto el elemento "residentes" queda así:

```
<xsd:element name="residentes" type="tipo_residentes"/>
```

**Selección excluyente.**

Para el caso de "mediopago" hemos indicado el caso de "tarjeta_credito", pero obviamente podrían haber otros más. Asumamos que también puede haber "efectivo", el cual tiene como atributo la moneda. Normalmente el medio de pago es uno u otro, pero no ambos, por lo que la "secuencia" de elementos no tiene mucho sentido aquí.

Para los casos de opciones excluyentes se emplea "**xsd:choice**", y su uso es análogo a sequence:

```
<xsd:element name="mediopago">
<xsd:complexType>
<xsd:choice>

    <xsd:element name="tarjeta_credito">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tipo" type="xsd:string"/>
      <xsd:element name="numero" type="xsd:string"/>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

    <xsd:element name="efectivo">
    <xsd:complexType>
      <xsd:attribute name="moneda" type="xsd:string"/>
    </xsd:complexType>
    </xsd:element>

</xsd:choice>
</xsd:complexType>
```

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

No olvidemos modificar el XML para que haga referencia a nuestro esquema:

```
<reserva version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="solicitud.xsd">
<habitaciones>
...
```

También si alteramos el medio de pago:

```
<mediopago>
   <efectivo moneda="EUR"/>
</mediopago>
```

## No cualquier Moneda!

Típicamente los establecimientos sólo aceptan ciertas monedas para las operaciones. Por ejemplo, asumamos que sólo se aceptarán euros (EUR) y dólares (USD), podemos utilizar para el efectivo:

```
<xsd:attribute name="moneda">
<xsd:simpleType>

    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="EUR"/>
      <xsd:enumeration value="USD"/>
    </xsd:restriction>

</xsd:simpleType>
</xsd:attribute>
```

Si tenemos :

```
<mediopago>
    <efectivo moneda="PEK"/>
</mediopago>
```

Y validamos, tendremos errores:

- sol_reserva_ef_schema.xml:26: element efectivo: Schemas validity error : Element 'efectivo', attribute 'moneda': [facet 'enumeration'] The value 'PEk' is not an element of the set {'PEN', 'USD'}.
- sol_reserva_ef_schema.xml:26: element efectivo: Schemas validity error : Element 'efectivo', attribute 'moneda': 'PEk' is not a valid value of the local atomic type.
- sol_reserva_ef_schema.xml - invalid

## Atributos Default

Entre las personas que se alojan por lo general hay un responsable de hacer las coordinaciones y servir

de intermediario con el hotel: la "cabeza de grupo". Esto lo hemos señalado mediante el atributo "responsable" del elemento "residente". Sin embargo, es interesante notar que este atributo no fue fijado para todos los residentes, solo para el primero de ellos.

Esto es un concepto general: los atributos no son necesariamente obligatorios (aunque no se pueden agregar arbitrariamente si no están declarados en el esquema.)

En ciertas aplicaciones, es conveniente que los atributos sí sean obligatorios. Esto lo podríamos haber señalado así en nuestro esquema:

```
<xsd:attribute name="responsable" type="xsd:boolean" use="required"/>
```

Con esto la validación fallaría:

- $ xmlstarlet val -e -s solicitud_atob.xsd sol_reserva_ef_schema.xml
- sol_reserva_ef_schema.xml:16: element residente: Schemas validity error : Element 'residente': The attribute 'responsable' is required but missing.
- sol_reserva_ef_schema.xml - invalid

Para efectos de la aplicación que procesa el documento, es más conveniente recibir siempre un atributo "responsable" (en vez de considerar separadamente el caso en que hay o no hay tal atributo.) Del mismo modo, puede ser conveniente para quien redacta el documento el no tener que suministrarlo; para esta situación el atributo "default" proporciona un compromiso:

```
<xsd:attribute name="responsable" type="xsd:boolean" default="false"/>
```

Los valores default también pueden aplicarse a elementos. Por ejemplo, asumiendo que la mayoría de residentes es español, se podría definir el elemento nacionalidad así:

```
<xsd:element name="nacionalidad" type="xsd:string" default="ES"/>
```

Con esto, el documento se podría simplificar así:

```
<nombres>Pedro</nombres>
<apellido>Camacho</apellido>
<pasaporte>552321</pasaporte>
<nacionalidad/>
```

Sin embargo, si se elimina el elemento fallaría:

```
<nombres>Pedro</nombres>
<apellido>Camacho</apellido>
<pasaporte>552321</pasaporte>
```

- sol_reserva_ef_schema.xml:10: element residente: Schemas validity error : Element 'residente': Missing child element(s). Expected is ( nacionalidad ).
- sol_reserva_ef_schema.xml - invalid

Es decir, el valor "default" de un elemento aplica sólo para aquellos que están vacíos, pero que sí existen.

### Elementos Opcionales

Suponiendo que no se dispone del pasaporte, pero igual deseamos hacer la reserva:

```
<nombres>Julia</nombres>
<apellido>Urquidi</apellido>
<!--<pasaporte>664423</pasaporte>-->
<nacionalidad>BO</nacionalidad>
```

Para que esto sea válido podemos modificar el esquema así:

```
<xsd:element name="pasaporte" type="xsd:string" minOccurs="0"/>
```

Por omisión minOccurs="1" y maxOccurs="1" por lo que normalmente todo elemento debe aparecer exactamente una vez.

## Espacios de Nombres

En casos más complejos es frecuente la reutilización de distintos esquemas para validar un mismo documento. Asimismo, los esquemas se pueden construir reutilizando tipos definidos en otros esquemas. Esto puede dar lugar a conflictos de nombres en los que distintos creadores de esquemas eligieron el mismo tag para designar algún elemento.

A fin de evitar estos conflictos, es conveniente que los tipos definidos en los esquemas tengan un "espacio de nombres" propio. Por ejemplo, para nuestro esquema de reservas podríamos definir el espacio de nombres: "http://www.hotel-esquematico.com/reservas" (convencionalmente se emplea un URI), y empleamos el atributo targetNamespace:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas">
<xsd:complexType name="tipo_residentes">

...

<xsd:sequence>
  <xsd:element name="desde" type="xsd:date"/>
  <xsd:element name="hasta" type="xsd:date"/>
  <xsd:element name="residentes" type="nuevo:tipo_residentes"/>
</xsd:sequence>
```

Nótese que nuestro esquema hace uso del tipo "tipo_residentes" el cual ya no se puede referenciar por mediante type="tipo_residentes" dado que los tipos pertenecen al espacio de nombres que estamos creando, por lo tanto debemos hacer referencia al mismo usando un prefijo; por este motivo hemos asociado un prefijo ("nuevo") con el mismo espacio de nombres con la directiva:

```
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
```

Así podemos definir el elemento "residentes" satisfactoriamente.

A continuación, el documento XML sufre estos cambios:

```
<res:reserva version="1" encoding="UTF-8"
xmlns:res="http://www.hotel-esquematico.com/reservas"
```

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**
**xsi:schemaLocation="solicitud_ns.xsd">**
**...**

**</res:reserva>**

Esto es, se ha definido un prefijo (que hemos denominado "**res**") asociado al espacio de nombres que se creó en el esquema, y con este prefijo estamos declarando al elemento "reserva" (es decir, "res:reserva".)

## Calificación de elementos

Como se vio en la sección anterior, el documento sólo requirió prefijar al elemento "reserva". De acuerdo al esquema este es el único elemento "global", es decir, que es declarado como hijo inmediato de <schema>. El resto de elementos del esquema (por ejemplo, "habitaciones", etc.) se denominan "locales".

Por omisión, el prefijo es necesario sólo para los elementos globales.

Si se desea forzar el prefijo a los elementos locales (además de los globales), se debe emplear la "qualificación" de elementos:

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas"
**elementFormDefault="qualified"**>

Con esto nuestro documento tendrá esta forma:

<?xml version="1.0" encoding="UTF-8"?>
<res:reserva version="1"
xmlns:res="http://www.hotel-esquematico.com/reservas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="solicitud_ns.xsd">
<res:habitaciones>
<res:habitacion numero="202">
<res:desde>2009-11-22</res:desde>
<res:hasta>2009-11-24</res:hasta>
<res:residentes>
<res:residente responsable="true">
<res:nombres>Pedro</res:nombres>

...

Análogamente se puede exigir que los atributos sea "qualificados" mediante el atributo de "schema": attributeFormDefault="qualified"

Cuando se tiene un documento XML en el que todos o la mayoría de elementos comparten un mismo prefijo, puede ser conveniente utilizar el "default namespace", que corresponde al espacio de nombre

asociado a los elementos sin prefijo. Así, el documento anterior equivale a este:

```
<?xml version="1.0" encoding="UTF-8"?>
<reserva version="1"
xmlns="http://www.hotel-esquematico.com/reservas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="solicitud_ns.xsd">
<habitaciones>
<habitacion numero="202">
<desde>2009-11-22</desde>
<hasta>2009-11-24</hasta>
<residentes>
<residente responsable="true">
<nombres>Pedro</nombres>
```

Notar el uso de xmlns="http://www.hotel-esquematico.com/reservas", esto es, el default namespace asumido para todos los elementos sin prefijo.

## Dividiendo el esquema en partes menores

El esquema ahora será dividido en dos partes. Una de ellas contendrá la definición del tipo "tipo_residente":

**solicitud_nsq_p2.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nuevo="http://www.hotel-esquematico.com/reservas"
targetNamespace="http://www.hotel-esquematico.com/reservas"
elementFormDefault="qualified">
<xsd:complexType name="tipo_residentes">
<xsd:sequence>
<xsd:element name="residente" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nombres" type="xsd:string"/>
<xsd:element name="apellido" type="xsd:string"/>
<xsd:element name="pasaporte" type="xsd:string"/>
<xsd:element name="nacionalidad" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="responsable" type="xsd:boolean"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

La otra parte incluirá la anterior:

**solicitud_nsq_p1.xsd**

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

<xsd:include schemaLocation="solicitud_nsq_p2.xsd"/>

<xsd:element name="reserva">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="habitaciones">
...
```

Aquí el elemento clave es "xsd:include", que permite incluir otro archivo de esquema. Con esto, todas las definiciones del primer esquema se suman al del segundo. El documento XML sólo requiere apuntar al documento "top".

Esto solo es válido si el documento incluido define elementos en el mismo espacio de nombres del inclusor (es decir, ambos comparten el mismo targetNamespace.)

# Ejercicios

### Ejercicio 1

Definir un elemento "edadLaboral" que será de tipo entero no negativo, debe tener un valor mínimo de 16 (incluido) y máximo de 70 (no incluido), es decir, [16, 70).

### Ejercicio 2

Definir el tipo de datos "TipoEdadLaboral" de cara a poder utilizarlo en repetidas ocasiones. Declarar el elemento "edadLaboral" de tipo "TipoEdadLaboral"

### Ejercicio 3

Definir un tipo de dato simple"TipoEstaciones", basado en el tipo predefinido xs:token que solamente permita como valores los nombres de las cuatro estaciones.

### Ejercicio 4

Definir un tipo de dato simple "TipoCantidad", basado en el tipo predefinido xs:decimal que permita números con 2 dígitos decimales y 11 dígitos en total. El rango de valores implícito sería desde -999.999.999,99 hasta 999.999.999,99.

### Ejercicio 5

Definir un tipo de dato simple "TipoDireccionFormateada", basado en el tipo predefinido xs:string de manera que todos los caracteres de espaciado se colapsen y la definición de este tipo coincida con el tipo predefinido xs:token.

### Ejercicio 6

Definir un tipo de datos basado en xs:string que se caracterice por cadenas de tres letras mayúsculas.

### Ejercicio 7

Definir una expresión regular para una cadena de caracteres que contenga las letras "a" o "b" o "+".

### Ejercicio 8

Define un tipo simple para números de teléfono, debe estar representado por 3 dígitos, un guión, otros tres dígitos, otro guión y tres dígitos más. (Ejemplo: 923-678-341)

### Ejercicio 9

Define tipos simples a partir de las siguientes especificaciones:

Una cadena de texto que represente las matrículas españolas anteriores a la normalización del año 2000. El formato es: una o dos letras mayúsculas + un guión + cuatro dígitos + un guión + una o dos letras mayúsculas (Ejemplo: ZA-2317-AB)

Una cadena de texto que represente las cuatro posibles formas de pago: con tarjeta VISA, MasterCard, American Express y en Efectivo.

A continuación, crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos.
Escribe un documento instancia XML que sea válido con respecto a ese esquema.

### Ejercicio 10

Define diferentes tipos simples a partir de las siguientes especificaciones:

- Un número real con tres decimales que represente las temperaturas posibles de la tierra, suponiendo que van desde -75 a 75 grados, ambas incluidas.
- Un xs:token que sólo pueda valer las siglas de los países vecinos de España, incluyendo a España: ES, PR, FR, AN.
- Un número real que represente salarios, con 5 dígitos y 2 decimales.

- Un mensaje de la red social Tweeter constituido por una cadena de caracteres de una longitud máxima de 140 caracteres.

A continuación crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos. Por último, escribe un documento instancia XML que sea válido con respecto a ese esquema.

## Ejercicio 11

Se quiere reflejar que las tallas de ropa se pueden identificar por números o por letras (S, M, L) de manera que cualquiera de los fragmentos XML <talla>42</talla> o <talla>S</talla> sea válido.

## Ejercicio 12

Definir un tipo TipoListaNumerosBingo como una lista de valores enteros positivos que empiezan por 1 y acaban en 90.

## Ejercicio 13

Dado un tipo simple "TipoColoresSemaforo" basado en xs:string y que sólo pueda tomar los valores Rojo, Amarillo y Verde, define un nuevo tipo "TipoListaColoresSemaforo" que sea una lista cuyos elementos sean del tipo "TipoColoresSemaforo".

Por último, crea otro tipo que sea una lista de sólo tres colores de semáforo, "Tipo3ColoresSemaforo"

Crea un esquema que contenga estos tipos y declara un elemento del tipo recién construido "Tipo3ColoresSemaforo"

## Ejercicio 14

Definir el tipo <mensaje> con los elementos descendientes <emisor><receptor> y <contenido>

## Ejercicio 15

Definir los tipos correspondientes a un documento novela en la que puede haber un elemento

<prologo>, o un elemento <prefacio> o un elemento <introduccion>, sólo uno de los tres, todos textuales.

### Ejercicio 16

Definir un elemento <letrasgriegas> en el que los elementos <alfa> y <omega> y aparecen como sus descendientes pero sin importar su orden.

### Ejercicio 17

Definir un elemento <persona> que contenga una secuencia de elementos descendientes <primerApellido>, <segundoApellido> y <fechaNacimiento>.

### Ejercicio 18

Basándote en el ejercicio anterior definir un tipo complejo "TipoPersona" y declarar un elemento de ese tipo.

### Ejercicio 19 : alumno.xml

**Definir un esquema para que los datos de alumno contenidos en el siguiente fichero sean validados correctamente.**

El archivo alumno.xml sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<alumno dni="111" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="alumno.xsd">
<nombre>Juan Garcia</nombre>
<direccion>
<calle>Avenida de la Fuente</calle>
<numero>6</numero>
<ciudad>Zafra</ciudad>
<provincia>Badajoz</provincia>
</direccion>
<telefono>924555555</telefono>
</alumno>
```

## Ejercicio 20alumno2.xml

**Vamos a ampliar el ejercicio anterior haciendo que permita una relación de alumnos.**

```
<?xml version="1.0" encoding="UTF-8"?>
<alumnos  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="alumnos.xsd">
<alumno dni="111">
<nombre>Juan Garcia</nombre>
<direccion>
<calle>Avenida de la Fuente</calle>
<numero>6</numero>
<ciudad>Zafra</ciudad>
<provincia>Badajoz</provincia>
</direccion>
<telefono>924555555</telefono>
</alumno>
<alumno dni="222">
<nombre>Jose Sanchez</nombre>
<direccion>
<calle>Calle Ancha</calle>
<numero>3</numero>
<ciudad>Zafra</ciudad>
<provincia>Badajoz</provincia>
</direccion>
<telefono>924550000</telefono>
</alumno>
</alumnos>
```

## Actividad OBLIGATORIA

Construir el documento XSD para el siguiente documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<libro isbn="786-78-5645-567-1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                xsi:noNamespaceSchemaLocation="Ejercicio_28.xsd">

    <titulo>Un mundo feliz</titulo>
    <autor>Aldoux Huxley</autor>
    <personaje codigo="A1">

        <nombre>Bernard Marx</nombre>
        <amigoDe>A2</amigoDe>
        <amigoDe>A4</amigoDe>
        <amigoDe>A3</amigoDe>
        <desde>1983-12-12</desde>
        <calificación>Bueno</calificación>
```

```
        </personaje>
        <personaje codigo="A2">

            <nombre>Lenina Crowne</nombre>
            <desde>1984-06-07</desde>
            <calificación>Regular</calificación>

        </personaje>
        <personaje codigo="A3">

            <nombre>Helmhotz Watson</nombre>
            <desde>1984-06-07</desde>
            <calificación>Regular</calificación>

        </personaje>
        <personaje codigo="A4">

            <nombre>Benito Hoover</nombre>
            <amigoDe>A1</amigoDe>
            <desde>1984-06-07</desde>
            <calificación>Regular</calificación>

        </personaje>

    </libro>
```