# Unidad 6. XML Schema

# 1 Esquemas XML

En este punto del curso conocemos ya la importancia de poder validar documentos XML para así garantizar la robustez y homogeneidad de los datos. En el tema anterior aprendimos a usar uno de los mecanismos que se pueden utilizar para validar documentos: las DTD. En este tema veremos otro mecanismo para ello, los esquemas XML (XML schemas).

Las diferencias más significativas entre los dos métodos, DTD y XML Schema, son las siguientes:

- Los XML schemas son documentos XML, emplean la sintaxis propia de XML, y por tanto pueden ser analizados sintácticamente para comprobar su buena formación, pueden ser manipulados por otro documento XML, y pueden utilizarse herramientas XML para elaborarlos y tratarlos.
- Los XML schemas son más potentes que las DTD, es posible describir con mucho más detalle un documento; por ejemplo con los esquemas se pueden definir tipos de datos, o especificar exactamente el número de ocurrencias de un elemento dentro del documento.
- Con los XML Schema no es posible utilizar entidades.
- A la hora de validar, la utilización de XML Schema supone un gran consumo en recursos y tiempo debido a su gran especificación y complejidad en la sintaxis (esquemas más difíciles de leer y escribir).

El lenguaje de los XML Schema también se conoce como XSD (XML Schema Definition).

Los XML schemas aparecieron con posterioridad a las DTD, y son cada vez más utilizados debido a las ventajas que presentan frente a estos: los esquemas son extensibles, más potentes y versátiles, escritos en XML y soportan tipos de datos y espacios de nombres.

En Mayo de 2001 se publicó la primera versión de la recomendación del W3C para XML Schema, y en octubre de 2004 la segunda edición. La recomendación se divide en tres partes:

- **XML Schema parte 0: Primer (Fundamentos) [1]**, es un documento no normativo que proporciona una introducción a las capacidades de XML Schema y a la creación de esquemas usando el lenguaje XML Schema.
- **XML Schema parte 1: Structures (Estructuras) [2]**, especifica el lenguaje de definición de XML Schema, que permite describir la estructura y las restricciones en los contenidos de documentos XML.
- **XML Schema parte 2: Data Types (Tipos de Datos) [3]**, establece las herramientas para definir los tipos de datos que se usan en los XML Schema y en otras especificaciones XML.

Un esquema XML describe la estructura de un documento XML. Un esquema XML define:

- Los elementos que pueden aparecer en el documento
- Los atributos que pueden aparecer en el documento
- Qué elementos son elementos hijos
- El número de elementos hijo
- Si un elemento está vacío o puede contener texto
- Tipos de datos para los elementos y atributos
- Valores fijos y valores por defecto para los elementos y atributos

Un XML schema está formado por una colección de componentes o bloques. Existen 13 tipos de componentes, que se agrupan en 3 categorías:

1) **Componentes primarios:**
   - Definiciones de tipos de datos simples
   - Definiciones de tipos de datos complejos
   - Declaraciones de atributos
   - Declaraciones de elementos

2) **Componentes secundarios**
   - Definiciones de grupos de atributos
   - Definiciones de restricciones de identidad
   - Definiciones de grupos de modelos
   - Declaraciones de notaciones

3) **Componentes de ayuda ("helper")**
   - Anotaciones
   - Grupos de modelos
   - Partículas
   - Comodines (wildcards)
   - Usos de atributos

A lo largo de este tema vamos a ver en detalle los componentes más importantes, pero antes veamos un ejemplo sencillo para hacernos una idea de lo que estamos hablando.

## 2 Primer ejemplo de XML Schema

Sea el siguiente XML Schema, guardado en el archivo libro.xsd:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

Como hemos dicho anteriormente, los XML schemas son documentos XML. Así, todo esquema debe comenzar con una **declaración XML**: esta es la primera línea del documento.

En la segunda línea encontramos la declaración del **elemento esquema**:

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

Como todo documento XML, un schema debe tener un elemento raíz, dentro del cual se encuentran los demás componentes del documento. En los XML schema el elemento raíz es siempre el elemento "schema". La última línea del esquema es la etiqueta de cierre del elemento "schema":

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

El elemento "schema" incluye el atributo **xmlns** (XML NameSpace), que especifica el espacio de nombres para el esquema. El valor de este atributo es la URI que identifica el vocabulario (XML vocabulary) al que se ajusta el documento, en este caso el vocabulario XML Schema. La sintaxis genérica del atributo xmlns es la siguiente:

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

En el ejemplo, el formato `xmlns:xsd` indica que todos los elementos o atributos que lleven el prefijo "xsd:" 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 "xsd:" delante de todos los elementos y atributos del esquema.

Los espacios de nombres se estudiarán en detalle más adelante.

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".

Un ejemplo de documento XML asociado a este XML Schema es el siguiente (libro.xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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.

# 3 Espacios de nombres

Para entender bien el funcionamiento de los XML schemas es necesario conocer y saber manejar los espacios de nombres XML (XML name spaces).

Según Wikipedia:

"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 le 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".

La recomendación del W3C sobre espacios de nombres es [7].

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.

## *3.1 Declarar 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

### 3.2 Referenciar 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 puntos ":" 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.

### 3.3 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="ISO-8859-1"?>
<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".

### 3.4 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=”ISO-8859-1”?>
<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>
```

### *3.5 Uso de los espacios de nombres*

Los espacios de nombres se usan para combinar vocabularios y facilitan la incorporación de elementos no previstos inicialmente.

Los espacios de nombres se crearon para que no existieran colisiones entre los diferentes módulos de software que eran capaces de reconocer las marcaciones (etiquetas y atributos) del lenguaje XML ya que los diferentes documentos que contienen marcaciones de distintas fuentes independientes entre sí, solían tener problemas de reconocimiento cuando habían sido creados para otros programas de software que, sin embargo, utilizaban el mismo tipo de elemento o nombre de atributo.

Una de las motivaciones para esta modularidad es que, si existe un conjunto de marcaciones disponibles, que son entendibles y para las que existe software útil disponible, es mejor la reutilización de estas marcaciones que el hecho de reinventar unas nuevas. Así, se consideró que las construcciones de documentos debían tener nombres universales, cuyo ámbito se extendiera más allá del documento que las contuviera. La especificación Namespaces XML describe un mecanismo, los espacios de nombres XML, que lleva a cabo esta misión.

## 4 Espacios de nombres y XML schemas

Existe una relación estrecha entre espacios de nombres y XML schemas, en varios sentidos que veremos a continuación.

**a) Dentro del XML schema**

Al crear un XML schema hacemos uso de los elementos y atributos especificados en el estándar de XML Schema. Para que esto sea posible debemos incluir en el elemento raíz del esquema (el elemento “schema”) una referencia al espacio de nombres **“http://www.w3.org/2001/XMLSchema”**. Esto se hace incluyendo el atributo “xmlns” en el elemento “schema”:

- **xmlns:** identifica el espacio de nombres al que pertenecen los componentes incluidos en el esquema, asignando opcionalmente un prefijo (este prefijo suele ser “xs” o “xsd”. Así, la sentencia

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

  indica que los elementos y tipos de datos utilizados en el esquema provienen del espacio de nombres “`http://www.w3.org/2001/XMLSchema`”, al que se le asigna el prefijo “xs”.

Además, podemos hacer que el esquema que estamos creando tenga asociado un espacio de nombres propio (target namespace). Para ello se puede utilizar el atributo **targetNamespace** del elemento "schema":

- **targetNamespace:** crea un espacio de nombres al que pertenecen los elementos que se definen en el esquema. Por ejemplo:

```
targetNamespace="http://www.iesluisbraille.es/esquema1"
```

También se puede especificar si los elementos y los atributos declarados en él deben estar certificados por un espacio de nombres, ya sea explícitamente mediante un prefijo o implícitamente de forma predeterminada, cuando se utilicen en un documento instancia XML. Para ello se pueden utilizar los atributos **elementFormDefault** y **attributeFormDefault** del elemento **<xsd:schema>**. Los posibles valores de estos atributos son:

- "**qualified**": indica que, en los documentos instancia XML que referencien a este esquema, los elementos (en el caso de elementFormDefault)/atributos (en el caso de attributeFormDefault) del "target namespace" deben estar cualificados con un prefijo.

- "**unqualified**": Este es el valor por defecto. Indica que los elementos/atributos no necesitan estar prefijados en el documento instancia.

**b) En los documentos instancia XML**

En los documentos instancia XML, la referencia a un esquema XML se hace mediante atributos en el elemento raíz del documento instancia XML. Estos atributos, especificados en el estándar, se encuentran definidos en el espacio de nombres **"http://www.w3.org/2001/XMLSchema-instance"**. Por lo tanto para poder usarlos hay que hacer referencia a dicho espacio de nombres en el documento instancia XML, mediante el atributo "xmlns". Normalmente a este espacio de nombres se le asigna el prefijo "xsi" (aunque se puede usar cualquier prefijo), con lo que la referencia al espacio de nombres quedaría así:

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

Con esto podemos utilizar los atributos "noNameSpaceLoction" y "schemaLocation", que nos permiten asociar un documento instancia XML con un esquema:

- **noNamespaceSchemaLocation**: identifica un documento de schema que no tiene un espacio de nombres de destino (no incluye el atributo "targetNamespace") y lo asocia al documento XML instancia.

- **schemaLocation:** Asocia un documento de esquema que tiene un espacio de nombres de destino (targetNamespace) con un documento de instancia. Este atributo tendrá dos valores, separados por un espacio en blanco. El primer valor coincide con el del "targetNamespace" especificado en el schema. El segundo es la ubicación donde se encuentra definido el XML schema.

Por ejemplo, supongamos que tenemos el archivo "apuntes.xsd" que contiene un XML schema. Si en dicho schema se definió

```
targetNamespace:"http://www.iesluisbraille.es/esquema1"
```

En un documento instancia que queremos asociar a este esquema tendremos:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.iesluisbraille.es/esquema1 apuntes.xsd"
```

Si por el contrario en el documento “apuntes.xsd” no se especifica Un “targetNamespace”, en el documento instancia XML tendremos:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=”apuntes.xsd”
```

# 5 Declaraciones de elementos

## *5.1 Cómo se declaran elementos: tipos simples y tipos complejos*

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. En el caso de los elementos con un valor “fixed”, el elemento puede estar vacío, o si no lo está, su contenido debe coincidir con el especificado en el atributo “fixed”.

### *5.2 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 “aniguedad”, que es de contenido vacío (no contiene ni texto ni otros subelementos), 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”.**

   a. **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>
```

**b. 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.

**c. 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>
```

### *5.3 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 inicial del tema (del apartado 2) utilizando referencias a elementos:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

## 6 Declaraciones de 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 excluyentes, 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>
```

## 7 Tipos de datos simples predefinidos (Built-in Data Types)

Las definiciones de tipos de datos se recogen en la parte 2 de la recomendación [3]. 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, como veremos más adelante.

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**

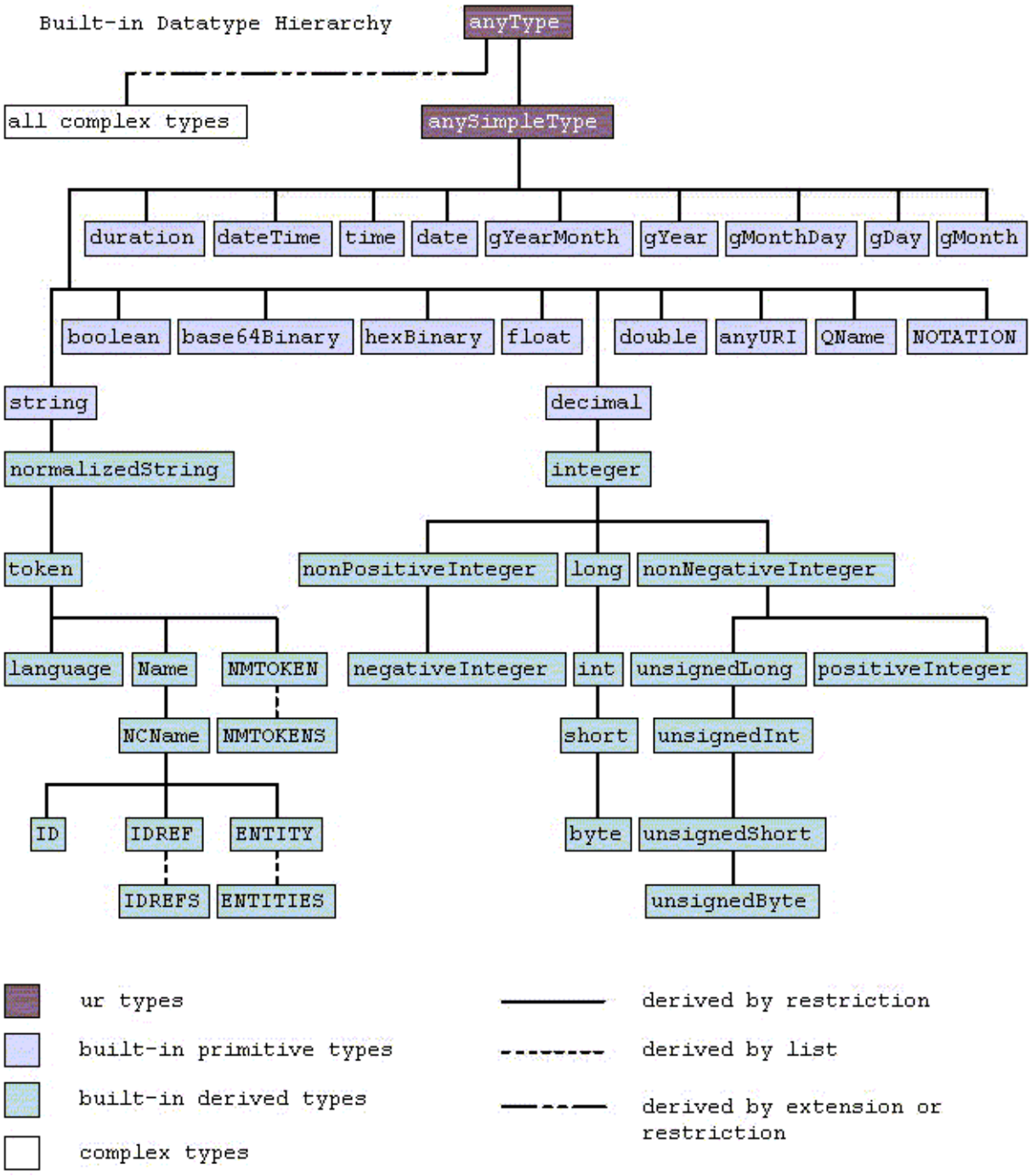
- **decimal:** número decimal de precisión (dígitos significativos) arbitraria *
- **float:** número de punto flotante de 32 bits de precisión simple *
- **double:** número de punto flotante de 64 bits de doble precisión *

**Tipos de fecha/hora**

- **duration:** duración de tiempo
- **dateTime:** instante de tiempo específico, usando calendario gregoriano, en formato "YYYY-MM-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)

De cada uno de estos tipos primitivos se pueden obtener tipos **derivados**, como se muestra en el siguiente diagrama, sacado de la recomendación [3]:

**Nota**: En este curso utilizaremos los tipos marcados con "*", y algunos de sus tipos derivados.




Es posible definir tipos de datos simples a partir de estos tipos predefinidos utilizando las llamadas *facetas* (lo veremos más adelante).

# 8 Definiciones de tipos de datos

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

## *8.1 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.

## *8.2 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 (lo veremos en el siguiente apartado)
- **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.

### *8.3 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.

### *8.4 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="listaTiendass">
   <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 eltipo “posiblesTiendas”. Por ejemplo, en un documento instancia XML podríamos encontrarnos algo como:

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

### *8.5 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.

## 9 Métodos de diseño de 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.

### 9.1 Diseño plano

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.

### 9.2 Usar referencias a elementos y atributos

En este método primero se definen los diferentes elementos y atributos, para después reverenciarlos utilizando el atributo “ref”.

### 9.3 Usar 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 atributo “type” de los elementos. Este mecanismo permite reutilizar definiciones de tipos en diferentes puntos del documento.

## 10 Referencias


[1] XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-0/

[2] XML Schema Part 1: Structures Second Edition. W3C Recommendation 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/structures.html

[3] XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

[4] Manual imprescindible de XML. Juan Diego Gutiérrez Gallardo. Ed. Anaya Multimedia.

[5] XML Primer Plus. Nicholas Chase. Sams Publishing.

[6] http://es.wikipedia.org/wiki/Espacio_de_nombres_XML

[7] Namespaces in XML 1.0 (Third Edition). W3C Recommendation 8 December 2009. http://www.w3.org/TR/REC-xml-names/

[8] "Hipertexto, el nuevo concepto de documento en la cultura de la imagen". María Jesús Lamarca Lapuente. Tesis doctoral. Universidad Complutense de Madrid.

http://www.hipertexto.info

[9] Tutorial sobre XML Schema de W3Schools. http://www.w3schools.com/Schema

[10] Tutorial de WebTaller.com sobre espacios de nombres en XML. http://www.webtaller.com/construccion/lenguajes/xml/lecciones/espacios-nombres-xml.php