3. W3C XML Schema Definition Language

En la especificación XML se hace referencia a las DTD como método para definir vocabularios XML, pero las DTD tienen una serie de limitaciones, que llevaron al W3C a definir una nueva especificación. Esta especificación recibió el nombre de "W3C XML Schema Definition Language" (que popularmente se llama ***XML Schema o XSD***); se creó para sustituir la DTD como método de definición de vocabularios de documentos XML. Se puede encontrar la especificación más reciente en w3.org/XML/Schema.

> Durante la especificación, muchos miembros del grupo de trabajo consideraron que *XSD* era demasiado complejo y desarrollaron alternativas más simples, entre las que destaca ***Relax NG*** (relaxng.org). Al ser posterior a *DTD* y a *XSD*, pudo solucionar algunos de los problemas de éstos. Los esquemas se pueden definir en dos versiones de Relax NG, una basada en XML y otra que no lo está, llamada **Relax NG Compact**, y que está pensada para generar documentos de esquemas más pequeños y compactos.

El éxito de *XSD* ha sido muy grande y actualmente se utiliza para otras tareas aparte de simplemente validar XML. También se utiliza en otras tecnologías XML como XQuery, servicios web, etc.

Las características más importantes que aporta XSD son:

- Está **escrito en XML** y, por tanto, no hay que aprender un lenguaje nuevo para definir esquemas XML.
- Tiene su **propio sistema de datos**, por lo que se podrá comprobar el contenido de los elementos.
- Soporta **espacios de nombres** para permitir mezclar diferentes vocabularios.
- Permite **ser reutilizado** y sigue los modelos de programación como **herencia** de objetos y **sustitución** de tipo.

3.1. Asociar un esquema a un archivo XML

A diferencia de lo que ocurre en otros lenguajes de definición, (como por ejemplo en las DTD, en el que la asociación se debe especificar en el documento XML), para validar un XML con un XSD no es necesario modificar el archivo XML. De todos modos, también es posible hacerlo definiendo el espacio de nombres.

Para asociar un documento XML a un documento de esquema hay que definir en él el espacio de nombres con el atributo ***xmlns***, y por medio de uno de los atributos del lenguaje especificar cuál es el archivo de esquema:

```
<liga  xmlns: XSI = "http://www.w3.org/2001/XMLSchema-instance"
      XSI: noNamespaceSchemaLocation = "lliga.xsd" >
```

Se pueden definir las referencias al archivo de esquema de dos maneras:

Definir referencias a un esquema

atributo	significado
`noNamespaceSchemaLocation`	Se emplea cuando no se utilizarán espacios de nombres en el documento.
`schemaLocation`	Se emplea cuando se utilizan explícitamente los nombres de los espacios de nombres en las etiquetas.

3.2. Definir un archivo de esquema

XSD está basado en XML y, por tanto, debe cumplir con las reglas de XML:

- Aunque no es obligatorio normalmente siempre se empieza el fichero con la declaración XML.
- Sólo hay un elemento raíz, que en este caso es <**schema**>.

Como no se está generando un documento XML libre, sino que se está utilizando un vocabulario concreto y conocido, para poder utilizar los elementos XML siempre se deberá especificar el espacio de nombres de ***XSD:"http://www.w3.org/2001/XMLSchema".***

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

En la declaración de este espacio de nombres se está definiendo que ***xs*** (alias para hacer referencia a todas las etiquetas de este espacio de nombres).

Los **alias** para XSD normalmente son **xs** o **xsd**, pero en realidad no importa cuál se utilice siempre que se utilice el mismo en todo el documento.

XSD define muchas etiquetas y no se verán todas aquí. Puede encontrar todas las etiquetas posibles en la especificación www.W3.Org/TR/xmlschema11-1

La etiqueta ***<schema>*** puede tener diferentes atributos, algunos de los cuales podemos ver en la tabla:

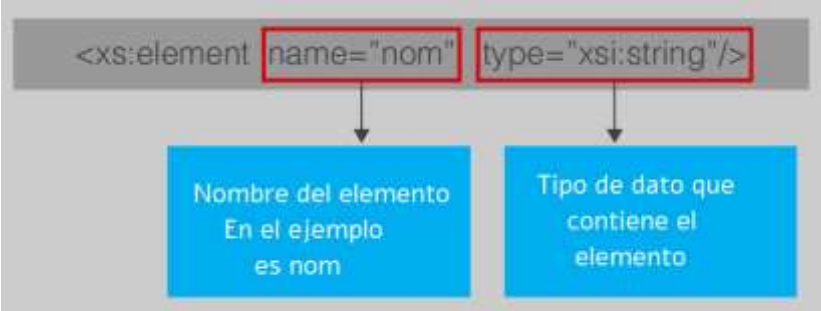
Atributos de la etiqueta `<schema>`

atributo	significado
`attributeFormDefault`	Puede ser `qualified` si hacemos que sea obligatorio poner el espacio de nombres ante los atributos o `unqualified` si no lo hacemos. Por defecto se utiliza `unqualified`.
`elementFormDefault`	Sirve para definir si es necesario el espacio de nombres delante del nombre. Puede tomar los valores `qualified` y `unqualified`.
`version`	Permite definir qué versión del documento de esquemas estamos definiendo (no es la versión de XML Schemas).

A partir de la etiqueta raíz ya se pueden empezar a definir las etiquetas del vocabulario que se quiere crear.

3.2.1. Definición de elementos

Los elementos se definen, tal como se ve en la siguiente figura, con la etiqueta <***element***>; con atributos se especifica como mínimo el nombre y opcionalmente el tipo de datos que contendrá el elemento.



XSD divide los elementos en dos grandes grupos basándose en los datos que contienen:

- Elementos con **contenido de tipo simple**: son elementos sin atributos que sólo contienen datos.
- Elementos con **contenido de tipo complejo**: son elementos que pueden tener atributos, no tener contenido o contener otros elementos.

Según la definición, se puede ver, que casi siempre habrá algún tipo complejo, ya que la raíz normalmente contendrá otros elementos.

- **Elementos con contenido de tipo simple**

Se consideran elementos con contenido de tipo simple aquellos que no contienen otros elementos ni tienen atributos.

La versión 1.1 de XSD define una cincuentena de tipos de datos diferentes, que se pueden encontrar en su definición (www.W3.Org/TR/xmlschema11-2).

Tipo de datos más usados en XSD

tipo	Datos que se pueden almacenar
string	Cadenas de caracteres
decimal	valores numéricos
boolean	Sólo puede contener 'true' o 'false' o (1 o 0)
date	Fechas en forma (AAAA-MM-DD)
anyURI	Referencias a lugares (URL , caminos de disco ...)
base64binary	Datos binarios codificados en *base64*
integer	números enteros

A partir de los tipos básicos, el estándar crea otros, con el objetivo de tener tipos de datos que se puedan adaptar mejor a los objetivos de quien diseña el esquema. En este sentido aparecen los tipos llamados ***positiveInteger***, ***nonNegativeInteger***, ***gYearMonth***, ***unsigned***, ***...***

Los tipos de datos permiten restringir los valores que contendrán las etiquetas XML. Por ejemplo, si se parte de la definición siguiente:

```
<xs:element  name = "posición"  type = "xs: integer"/>
</xs:schema >
```

Sólo se conseguirá validar un elemento, si su contenido es un número entero; el ejemplo siguiente no validará:

```
<posicion > Primero </posicion >
```

Ejemplos de elementos y qué se puede validar

etiqueta	ejemplo
<xs: element name = "día" type = "xs: date" />	<día> 2011-09-15 </día>
<xs: element name = "altura" type = "xs: integer" />	<altura> 220 </altura >
<xs: element name = "nombre" type = "xs: string' />	< nombre> Pere Puig </nombre >
<xs: element name = "tamaño" type = "xs: float" />	<'tamaño> 1.7E2 </tamaño >
<xs: element name = "sitio" type = "xs: anyURI" />	<sitio> http://www.ioc.cat </sitio >

Cuando se define una etiqueta en XSD, se está definiendo que la etiqueta deberá salir en el documento XML una vez. Es bastante habitual que haya etiquetas que se repitan un número determinado de veces. XSD lo ha simplificado usando unos atributos en la etiqueta <element> que determinan la *cardinalidad* de los elementos:

- **minOccurs**: permite definir cuántas veces tiene que salir un elemento como mínimo. Un valor de '0' indica que el elemento puede no salir.
- **maxOccurs**: sirve para definir cuántas veces como máximo puede salir un elemento. *Unbounded* implica que no hay límite en las veces que puede salir.

Utilizando estos atributos, se puede definir que el elemento <nombre> tiene que salir una vez y el elemento <apellido> un máximo de dos veces.

```
<xs:element  name = "nombre"/>
<xs:element  name = "apellido"  maxOccurs = "2"/>
```

También se pueden dar valores a los elementos con los atributos *fixed*, *default* y *nullable*.

El atributo ***fixed*** permite definir un valor obligatorio para un elemento:

```
<xs:element  name = "centro"  type = "xs: string"  fixed = "JA"/>
```

De modo que sólo se podrá definir el contenido con el valor especificado (o sin nada):

```
<centro/>
<centro > JA </centro >
```

Pero nunca un valor distinto del especificado:

```
<centro > Instituto Hermanos Machado </centro >
```

A diferencia de fixed, ***default*** asigna un valor por defecto, pero deja que sea cambiado en el contenido de la etiqueta.

```
<XSI: elemento  name = "centro"  type = "xs: string"  default = "JA"/>
```

La definición permitiría validar con los tres casos siguientes:

```
<centro/>
<centro > JA </centro >
<centro > Instituto Hermanos Machado </centro >
```

El atributo ***nullable*** se utiliza para decir si se permiten contenidos nulos. Por lo tanto, sólo puede tomar los valores “yes” o “no”.

Como a veces puede interesar definir valores para los elementos que no tienen por qué coincidir con los estándares, el XSD permite definir *tipos de datos personales*. Por ejemplo, si se quiere un valor numérico pero que no acepte todos los valores sino un subconjunto de los enteros. Para definir tipos simples personales no se pone el tipo en el elemento y se define un hijo <**simpleType**>.

```
<xs:element  name = "persona" >
   <xs:simpleType >
      ...
   </xs:simpleType >
</xs:elemento >
```

Dentro de *simpleType* se especifica cuál es la modificación que se quiere hacer. Lo más corriente es que las modificaciones sean hechas con ***list***, ***union***, ***restriction*** o ***extension***. A pesar de que se pueden definir listas de valores no se recomienda hacer. La mayoría de los expertos creen que es mejor definir los valores de la lista utilizando repeticiones de etiquetas. Utilizar ***list*** permitirá definir que un elemento pueda contener listas de valores. Por lo tanto, para especificar que un elemento <partidos> puede contener una lista de fechas se definiría:

```
<xs:element  name = "partidos" >
  <xs:simpleType >
     <xs:list  itemType = "xs: date"/>
  </xs:simpleType >
</xs:elemento >
```

La etiqueta validaría con algo como:

```
<partidos > 2011-01-07 2011-01-15 2011-01-21 </partidos >
```

Los elementos *simpleType* también se pueden definir con un nombre fuera de los elementos y posteriormente usarlos como tipo de datos personal.

```
<xs:simpleType name = "días" >
   <xs:list itemType = "xs: date"/>
</xs:simpleType >

<xs:element name = "partidos" type = "días"/>
```

Utilizando los tipos personalizados con su nombre, se pueden crear modificaciones de tipo ***unión***. Los modificadores unión sirven para hacer que se puedan mezclar tipos diferentes en el contenido de un elemento. La definición del elemento <precio> hará que el elemento pueda ser de tipo valor o de tipo símbolo.

```
<xs:element name = "precios">
   <xs:simpleType>
      <xs:union memberTypes = "valor símbolo"/>
   </xs:simpleType >
</xs:elemento >
```

Podríamos asignar valores como estos:

```
<precios > 25 € </precios>
```

Pero sin lugar a dudas el modificador más interesante es el que permite definir restricciones a los tipos base. Con el modificador de ***restriction*** se pueden crear tipos de datos en los que sólo se acepten algunos valores (que los datos cumplan una determinada condición, etc.).

El elemento <nacimiento> sólo podrá tener valores enteros entre 1850 y 2011 si se define de la siguiente manera:

```
<xs:simpleType name = "anio_nacimiento" >
   <xs:restriction base = "xs:integer" >
      <xs:maxInclusive value = "2011"/>
      <xs:minInclusive value = "1850"/>
   </xs:restriction >
</xs:simpleType >
```

```
<xs:element name = "nacimiento" type = "anio_nacimiento "/>
```

Se pueden definir restricciones de muchos tipos por medio de atributos. Normalmente los valores de las restricciones se especifican en el atributo *value*; veamos la tabla siguiente:

Atributos que permiten definir restricciones en XSD

elementos	resultado
`maxInclusive / maxExclusive`	Se utiliza para definir el valor numérico máximo que puede tomar un elemento.
`minInclusive / minExclusive`	Permite definir el valor mínimo del valor de un elemento.
`length`	Con `lenghtrestringimos` la longitud que puede tener un elemento de texto. Podemos utilizar `<xs:minLength>`y `<xs:maxLenght>`para ser más precisos.
`enumeration`	Sólo permite que el elemento tenga alguno de los valores especificados en las diferentes líneas `<enumeration>`.
`totalDigits`	Permite definir el número de dígitos de un valor numérico.
`fractionDigits`	Sirve para especificar el número de decimales que puede tener un valor numérico.
`pattern`	Permite definir una expresión regular en la que el valor del elemento debe adaptarse para poder ser válido.

Por ejemplo, el valor del elemento <respuesta> sólo podrá tener uno de los tres valores "A", "B" o "C":

```
<xs:element name = "respuesta" >
   <xs:simpleType >
      <xs:Enumeration value = "A"/>
      <xs:Enumeration value = "B"/>
      <xs:Enumeration value = "C"/>
   </xs:simpleType >
</xs:element >
```

Una de las restricciones más interesantes son las definidas por el atributo *pattern*, que permite definir restricciones a partir de expresiones regulares. Como norma general, si se especifica un carácter en el patrón, este carácter debe salir en el contenido; las otras posibilidades las podemos ver en la tabla siguiente:

Definición de expresiones regulares

símbolo	equivalencia
.	cualquier carácter
\ d	cualquier dígito
\ D	Cualquier carácter no dígito
\ s	Caracteres no imprimibles: espacios, tabuladores, saltos de línea ...
\ S	Cualquier carácter imprimible
x *	El de ante * lesionado 0 o más veces
x +	El de ante + lesionado 1 o más veces
x?	El de delante? puede salir o no
[Abc]	Tiene que haber algún carácter de los de dentro
[0-9]	Tiene que haber un valor entre el dos especificados, con estos incluidos
x {5}	Tiene que haber 5 veces lo que haya delante de los corchetes
x {5,}	Tiene que haber 5 o más veces el de delante
x {5,8}	Tiene que haber entre 5 y 8 veces el de delante

Utilizando este sistema se pueden definir tipos de datos muy personalizados. Por ejemplo, podemos definir que un dato debe tener la forma de un DNI (8 cifras, un guion y una letra mayúscula) con esta expresión:

```
<xs:simpleType name = "dni" >
   <xs:restricción base = "xs: string" >
      <xs:pattern value = "[0-9] {8} - [AZ]"/>
   </xs:restricción >
</xs:simpleType >
```

Aparte de las restricciones también está ***extension***, que sirve para añadir características extra a los tipos. No tiene mucho sentido que salga en elementos de tipo simple, pero se puede utilizar, por ejemplo, para añadir atributos a los tipos simples (lo que les convierte en tipos complejos).

- **Elementos con contenido de tipo complejo**

Los elementos con contenido de tipo complejo son aquellos que tienen atributos, contienen otros elementos o no tienen contenido.

Los elementos con contenido complejo han recibido muchas críticas porque se consideran demasiado complicados, pero deben utilizarse ya que, en todos los archivos de esquema normalmente habrá un tipo complejo: la raíz del documento.

Se considera que hay cuatro grandes grupos de contenidos de tipo complejo:

- Los que en su contenido **sólo tienen datos**. Por lo tanto, son como los de tipo simples, pero con atributos.
- Los elementos que en el contenido sólo **tienen elementos**.
- Los elementos **vacíos**.
- Los elementos con contenido **mezclado**.

Los elementos con tipo complejo se definen especificando que el tipo de datos del elemento es <**xs:*complexType***>.

```
<xs:element name = "clase" >
   <xs:complexType >
        ....
   </xs:complexType >
</xs:element >
```

Del mismo modo que con los tipos simples, se pueden definir tipos complejos con nombre para reutilizarlos como tipos personalizados.

```
<xs:complexType  name = "curso" >
    ...
</xs:complexType >

<xs:elemento clase type = "curso"/>
```

Una característica básica de XSD es que sólo los elementos de tipo complejo pueden tener atributos. En esencia no hay muchas diferencias entre definir un elemento o un atributo, ya que se hace de la misma manera, pero con la etiqueta ***attribute***.
Los tipos de datos son los mismos y, por tanto, pueden tener tipos básicos como en el ejemplo siguiente:

```
<xs:attribute  name = "número" type = "xs: integer" />
```

Se pueden poner restricciones al igual que en los datos. En este ejemplo el atributo año puede tener valores superiores a 2011:

```
<xs:attribute  name = "año" >
   <xs:simpleType >
      <xs:restriction  base = "xs: integer" >
         <xs:maxInclusive  value = "2011"/>
      </xs:restriction >
   </xs:simpleType >
</xs:attribute >
```

Salvo que se especifique lo contrario, los atributos siempre **son opcionales**.

La etiqueta <*attribute*> tiene una serie de atributos que permiten definir características extra sobre los atributos:

Atributos más importantes de xs: attribute

atributo	uso
`use`	Permite especificar si el atributo es obligatorio (`required`), opcional (`optional`) o bien no se puede utilizar (`prohibited`).
`default`	Define un valor predeterminado.
`fixed`	Se utiliza para definir valores obligatorios para los atributos.
`form`	Permite definir si el atributo tiene que ir con el alias del espacio de nombres (`qualified`) o no (`unqualified`).

Por ejemplo, el atributo año, deberá especificarse obligatoriamente si se define de la siguiente manera:

```
<xs:attribute  name = "año" type = "xs: integer" use = "required" />
```

- **Sólo datos**

Si el elemento sólo contiene texto, el contenido de *complexType* será un ***simpleContent***. Permite definir restricciones o extensiones a elementos que **sólo tienen datos** como contenido.

La diferencia más importante es que en este caso se pueden definir atributos en el elemento. Los atributos se añaden definiendo una extensión al tipo que se ha especificado para el elemento.

En este ejemplo el elemento <Tamaño> tiene contenido de tipo entero y define dos atributos, longitud y anchura, que también son enteros.

```
<xs:complexType  name = "Tamaño" >
   <xs:simpleContent >
      <xs:extension  base = "xs: integer" >
         <xs:atribute  name = "longitud"  type = "xs: integer"/>
         <xs:atribute  name = "anchura" type = "xs: integer"/>
      </xs:extension >
   </xs:simpleContent >
</xs:complexType >
```

Los **elementos que contienen otros elementos** también pueden ser definidos en XSD dentro de un <*complexType*> y pueden ser algunos de los elementos de la siguiente tabla:

Elementos para definir contenido complejo

etiqueta	sirve para
`sequence`	Especificar el contenido como una lista ordenada de elementos.
`choice`	Permite especificar elementos alternativos.
`all`	Definir el contenido como una lista desordenada de elementos.
`complexContent`	Extender o restringir el contenido complejo.

El elemento ***<sequence>*** es una de las maneras con las que el lenguaje XSD permite especificar los elementos que deben formar parte del contenido de un elemento. Incluso en el caso en que sólo haya una sola etiqueta, se puede definir como una secuencia.
La condición más importante que tienen, es que los elementos del documento XML a validar deben aparecer en el mismo orden en el que se definen en la secuencia.

```
<xs:element  name = "persona" >
   <xs:complexContent >
      <xs:sequence >
         <xs:element  name = "nombre"  type = "xs: string"/>
         <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
         <xs:element  name = "tipo"  type = "xs: string" />
      </xs:sequence >
   </xs:complexContent >
</xs:elemento >
```

En el ejemplo anterior se define que antes de la aparición de <tipo> pueden aparecer uno o dos apellidos.

```
<persona >
   <nombre > Marcelino </nombre >
   <apellido > Paredes </apellido >
   <apellido > Lozano </apellido >
   <tipo > Profesor </tipo >
</persona >
```

No validará ningún contenido si algún elemento no está exactamente en el mismo orden.

```
<persona >
   <tipo > Profesor </tipo >
   <apellido > Paredes </apellido >
   <nombre > Marcelino </nombre >
</persona >
```

Las secuencias pueden contener en su interior otras secuencias de elementos.

```
<xs:schema  xmlns: xs = "http://www.w3.org/2001/XMLSchema"  elementFormDefault = "qualified" >
   <xs:element  name = "persona" >
      <xs:complexType >
         <xs:sequence >
            <xs:element  name = "nomcomplet" >
               <xs:complexType >
                  <xs:sequence >
                     <xs:element  name = "nombre"  type = "xs: string"/>
                     <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
                  </xs:sequence >
               </xs:complexType >
            </xs:elemento >
            <xs:element  name = "profesión"  type = "xs: string"/>
         </xs:sequence >
      </xs:complexType >
   </xs:elemento >
</xs:schema >
```

El elemento ***<choice>*** sirve para hacer que se tenga que elegir una de las alternativas de las que se presentan en su interior.

En este ejemplo el elemento persona podrá contener o bien la etiqueta <nomApe> o bien <dni>, pero no ambas.

```
<xs:complexType  nombre = "persona" >
 <xs:choice >
    <xs:element  name = "nomApe"  type = "xs: string"/>
    <xs:element  name = "dni"  type = "xs: string"/>
 </xs:choice >
```

Entre las alternativas puede haber secuencias u otros elementos <*choice*>. La definición siguiente es un ejemplo más elaborado que el anterior y permite que se pueda elegir entre los elementos <nombre> y <apellido> o <dni>.

```
<xs:choice >
   <xs:sequence >
      <xs:element  name = "nombre"  type = "xs: string"/>
      <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
   </xs:sequence >
   <xs:element  name = "dni"  type = "xs: string"/>
</xs:choice >
```

La diferencia más importante entre el elemento ***<all>*** y ***<sequence>*** es el orden. El elemento <***all***> permite especificar una secuencia de elementos, pero permite que se especifiquen en cualquier orden. Por lo tanto, si definimos el elemento <persona> de la siguiente forma:

```
<xs:element  name = "persona" >
   <xs:complexType >
      <xs:all >
         <xs:element  name = "nombre"/>
         <xs:element  name = "apellido"/>
      </xs:all >
   </xs:complexType >
</xs:elemento >
```

Nos servirá para validar tanto este documento:

```
<persona >
   <nombre > Pedro </nombre >
   <apellido > García </nombre >
```

como este:

```
<persona >
   <apellido > García </nombre >
   <nombre > Pedro </nombre >
```

Pero siempre se deben tener en cuenta las limitaciones de este elemento (no estaban presentes en las secuencias ordenadas **<*sequence*>**):

- Dentro sólo puede haber elementos. No puede haber ni secuencias, ni alternativas.
- No se puede usar la cardinalidad en los elementos que contenga, ya que provocaría un problema de no-determinismo.

Por lo tanto, el ejemplo siguiente no es correcto, ya que se pide que <apellido> pueda salir dos veces.

```
<xs:all >
   <xs:element  name = "nombre"  type = "xs: string"/>
   <xs:element  name = "apellido"  maxOccurs = "2"  type = "xs: string"/>
</xs:all >
```

Una forma de permitir que se puedan especificar el nombre y los dos apellidos en cualquier orden sería hacer lo siguiente:

```
<xs:complexType >
   <xs:choice >
      <xs:sequence >
         <xs:element  name = "nombre"  type = "xs: string"/>
         <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
      </xs:sequence >
      <xs:sequence >
         <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
         <xs:element  name = "nombre"  type = "xs: string"/>
      </xs:sequence >
   </xs:choice >
</xs:complexType >
```

La etiqueta ***complexContent*** permite definir extensiones o restricciones a un tipo complejo que contenga contenido mezclado o sólo elementos. Esto hace que con una extensión se pueda ampliar un contenido complejo ya existente o restringir los contenidos.

Por ejemplo, si ya hay definido un tipo de datos *nomComplet* en la que están los elementos <nombre> y <apellido> se puede reutilizar la definición para definir un nuevo tipo de datos, *agenda* en el que se añada la dirección de correo electrónico.

```
<xs:complexType  name = "nomComplet" >
 <xs:sequence>
    <xs:element  name = "nombre"  type = "xs: string"/>
    <xs:element  name = "apellido"  type = "xs: string"  maxOccurs = "2"/>
 </xs:sequence >
</xs:complexType >

<xs:complexType  name = "agenda" >
   <xs:complexContent >
      <xs:extension  base = "nomComplet" >
         <xs:sequence >
            <xs:element  name = "email"  type = "xs: string" />
         </xs:sequence >
      </xs:extension >
   </xs:complexContent >
</xs:complexType >
```

De esta manera se podrá definir un elemento de tipo agenda:

```
<xs:element  name = "persona"  type = "agenda"/>
```

que deberá tener los tres elementos <nombre>, <apellido> y <email>:

```
<persona >
   <nombre > Pedro </nombre >
   <apellido > García </apellido >
   <email > pgarcia@ioc.cat </email >
</persona >
```

- **Elementos sin contenido**

Para XSD los elementos sin contenido son siempre **de tipo complejo**. En la definición simplemente no se especifica ningún contenido y tendremos un elemento vacío.

```
<xs:element  name = "delegado" >
   <xs:complexType />
</xs:elemento >
```

La especificación permite definir el elemento de esta manera:

```
<delegado />
```

Si el elemento necesita atributos simplemente se especifican dentro del *complexType*.

```
<xs:element  name = "delegado" >
   <xs:complexType >
      <xs:attribute  name = "año"  use = "required"  type = "xs: gYear"/>
   </xs:complexType >
</xs:elemento >
```

Y ya se podrá definir el atributo en el elemento vacío:

```
<delegado año = "2012"/>
```

- **Contenido mezclado**

Los elementos con contenido mezclado son los elementos que tienen de contenido **tanto elementos como texto**. Se pensó para poder incluir elementos en medio de un texto narrativo. En XSD el contenido mezclado se define poniendo el atributo ***mixed="true"*** en la definición del elemento *<complexType>.*

```
<xs:element  name = "carta" >
   <xs:complexType  mixed = "true" >
      <xs:sequence >
         <xs:element  name = "nombre"  type = "xs: string"/>
         <xs:element  name = "día"  type = "xs: gDay"/>
      </xs:sequence >
   </xs:complexType >
</xs:elemento >
```

Esto nos permitiría validar un contenido como este:

```
<carta > Estimado señor <nombre > Pedro <nombre >:
 Le envío esta carta para recordarle que hemos quedado para
encontrarnos el día <día > 12 </día >
</carta >
```

3.3. Ejemplo de creación de un XSD

Se pueden crear definiciones de vocabularios XSD a partir de lo que queremos que contengan los datos o bien a partir de un archivo XML de muestra. En un centro en el que sólo se imparten los ciclos formativos de SMR y ASIR, cuando entregan las notas a los alumnos lo hacen creando un archivo XML como el siguiente:

```
<? xml  version = "1.0"  ?>
<clase modulo = “3” nommodul = “Programación básica” >
   <curso numero = "1” especialidad = "ASIR" >
      <profesor >
         <nombre > Marcelino </nombre >
         <apellido > Pardo </apellido >
      </profesor >
      <alumnos >
         <alumno >
            <nombre > Filomeno </nombre >
            <apellido > García </apellido >
            <nota > 5 </nota >
         </alumno >
         <alumno delegado = "si" >
            <nombre > Frederic </nombre >
            <apellido > Paz </apellido >
            <nota > 3 </nota >
         </alumno >
         <alumno >
            <nombre > Manel </nombre >
            <apellido > Parrales </apellido >
            <nota > 8 </nota >
         </alumno >
      </alumnos >
   </curso >
</clase >
```

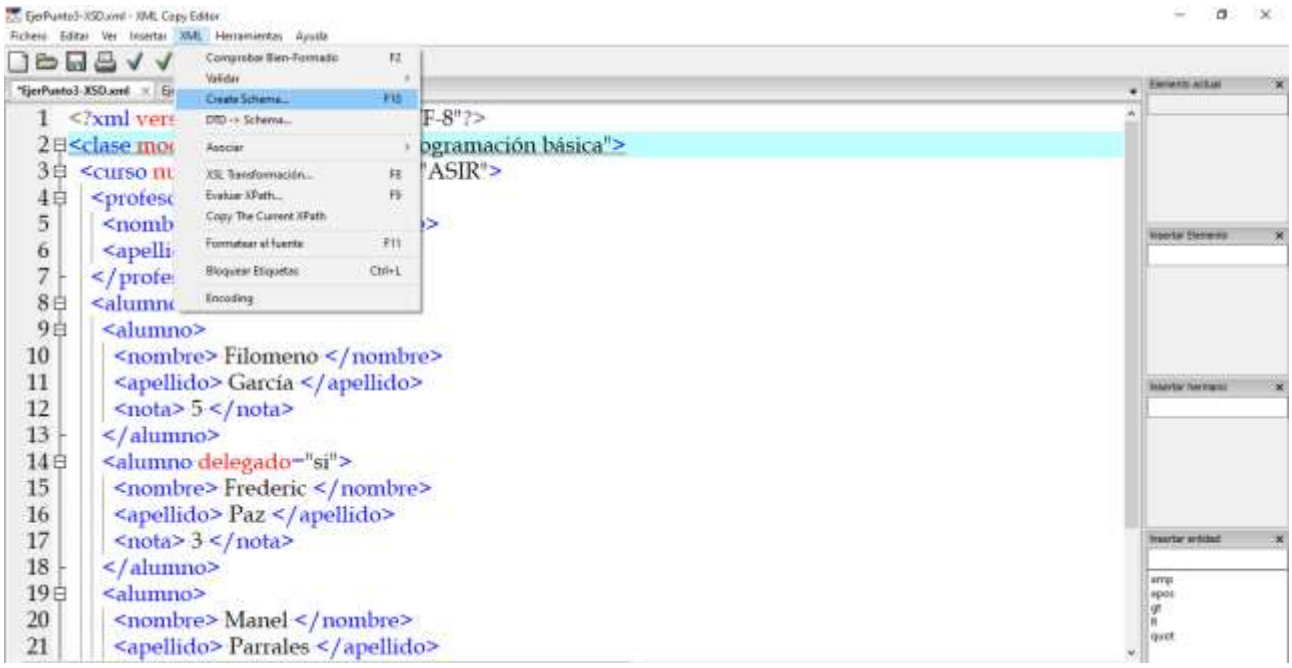
En la secretaría necesitan que se genere la definición del vocabulario en XSD para comprobar que los archivos que reciben son correctos.

3.3.1 Resolución

A la hora de diseñar un esquema desde un XML, siempre se debe partir de un fichero XML que tenga todas las opciones que pueden salir en cada elemento; en caso contrario, el resultado no será válido para todos los archivos.

Una de las cosas que se deben tener en cuenta a la hora de hacer un ejercicio como éste, es que **no hay una manera única de hacerlo**. Se puede hacer con tipos personalizados, sin tipos personalizados, a veces se puede resolver combinando unos elementos, pero también con otros, etc. Por lo tanto, la solución que se ofrecerá aquí es sólo una de las posibles soluciones.

Otra cosa que se debe tener en cuenta es que los editores XML suelen ofrecer una manera de crear XSD, como se puede ver en la figura.



3.3.2. Análisis

La primera fase normalmente consiste en analizar el fichero para determinar si hay partes que pueden ser susceptibles de formar un tipo de datos. En el ejercicio se puede ver, que tanto para el profesor como para los alumnos, los datos que contienen son similares (los alumnos añaden la nota), y por tanto se puede crear un tipo de datos que hará más simple el diseño final.

Por tanto, en la raíz podemos crear el tipo complejo persona, que contendrá el nombre y el apellido.

```
<xs:complexType  name = "persona" >
   <xs:sequence >
      <xs:element  name = "nombre"  type = "xs: string"/>
      <xs:element  name = "apellido"  type = "xs: string"/>
   </xs:sequence >
</xs:complexType >
```

Como los alumnos son iguales, pero con un elemento extra, se puede aprovechar el tipo persona extendiéndolo.

```
<xs:complexType  name = "estudiante" >
   <xs:complexContent >
      <xs:extension  base = "persona" >
         <xs:sequence >
            <xs:element  name = "nota" >
               ...
            </xs:elemento >
         </xs:sequence >
      </xs:extension >
   </xs:complexContent >
</xs:complexType >
```

Las notas no pueden tener todos los valores (sólo de 1 a 10), o sea, que se puede añadir una restricción al tipo entero.

```
<xs:element  name = "nota" >
   <xs:simpleType >
      <xs:restricción  base = "xs: integer" >
            <xs:maxInclusive  value = "10"/>
            <xs:minInclusive  value = "1"/>
      </xs:restricción >
   </xs:simpleType >
</xs:elemento >
```

3.3.3. Desarrollo

La raíz siempre será de tipo complejo porque contiene atributos y tres elementos, de manera que se puede comenzar la declaración como una secuencia de elementos.

```
<xs:element  name = "clase" >
   <xs:complexType >
      <xs:sequence >
         <xs:element  name = "curso" >
            ...
         </xs:elemento >
         <xs:element  name = "profesor"  type = "persona"/>
         <xs:element  name = "alumnos" >
            ...
         </xs:elemento >
      </xs:sequence >
   </xs:complexType >
</xs:elemento >
```

Se han dejado los elementos <curso>y <alumnos> para desarrollarlos, mientras que el elemento <profesor> ya se puede cerrar porque se ha definido el tipo persona.

El elemento <curso> no tiene contenido y, por tanto, será de tipo complejo. Además, se deben definir dos atributos.

```
<xs:element  name = "curso" >
   <xs:complexType >
      <xs:attribute  name = "numero"  use = "required" >
         ...
      </xs:attribute >
      <xs:attribute  name = "especialidad"  use = "required" >
         ...
      </xs:attribute >
   </xs:complexType >
</xs:elemento >
```

Los atributos deben desarrollarse porque no pueden tener todos los valores:

- numero sólo puede ser "1" o "2".
- especialidad será "SMR" o "ASIR".

La manera más adecuada para hacerlo será restringir los valores con una enumeración.

```
<xs:attribute  name = "numero"  use = "required" >
   <xs:simpleType >
      <xs:restricción  base = "xs: integer" >
         <xs:Enumeration  value = "1"/>
         <xs:Enumeration  value = "2"/>
      </xs:restricción >
   </xs:simpleType >
</xs:attribute >
<xs:attribute  name = "especialidad"  use = "required" >
   <xs:simpleType >
      <xs:restricción  base = "xs: string" >
         <xs:Enumeration  value = "ASIR"/>
         <xs:Enumeration  value = "SMR"/>
      </xs:restricción >
   </xs:simpleType >
</xs:attribute >
```

Ya sólo queda por desarrollar <alumnos>, que tiene una repetición de elementos <alumno>, del que ya hay un tipo.

```
<xs:element  name = "alumno" >
   <xs:complexType >
      <xs:sequence >
         <xs:element  name = "alumno"  type = "estudiante"
                  maxOccurs = "Unbounded" minOccurs = "1"/>
      </xs:sequence >
   </xs:complexType >
</xs:elemento >
```

El resultado final será:

```
<? xml  version = "1.0"  encoding = "UTF-8" ?>
<xs:schema  xmlns: xs = "http://www.w3.org/2001/XMLSchema" >
 <xs:complexType  name = "persona" >
   <xs:sequence >
     <xs:element  name = "nombre"  type = "xs: string"/>
     <xs:element  name = "apellido"  type = "xs: string"/>
   </xs:sequence >
</xs:complexType >
 <xs:complexType  name = "estudiante" >
   <xs:complexContent >
     <xs:extension  base = "persona" >
       <xs:sequence >
         <xs:element  name = "nota" >
           <xs:simpleType >
             <xs:restricción  base = "xs: integer" >
               <xs:maxInclusive  value = "10"/>
               <xs:minInclusive  value = "1"/>
             </xs:restricción >
           </xs:simpleType >
         </xs:elemento >
       </xs:sequence >
       <xs:attribute  name = "delegado"  use = "optional" >
         <xs:simpleType >
           <xs:restricción  base = "xs: string" >
             <xs:Enumeration  value = "si"/>
           </xs:restricción >
         </xs:simpleType >
       </xs:attribute >
     </xs:extension >
   </xs:complexContent >
</xs:complexType >
 <xs:element  name = "clase" >
   <xs:complexType >
     <xs:sequence >
       <xs:element  name = "curso" >
         <xs:complexType >
           <xs:sequence >
             <xs:element  name = "profesor"  type = "persona"/>
```

```
            <xs:element  name = "alumnos" >
              <xs:complexType >
                <xs:sequence >
                  <xs:element  name = "alumno"  type = "estudiante"  maxOccurs = "Unbounded" />
                </xs:sequence >
              </xs:complexType >
            </xs:elemento >
          </xs:sequence >
          <xs:attribute  name = "numero"  use = "required" >
            <xs:simpleType >
              <xs:restricción  base = "xs: integer" >
                <xs:Enumeration  value = "1"/>
                <xs:Enumeration  value = "2"/>
              </xs:restricción >
            </xs:simpleType >
          </xs:attribute >
          <xs:attribute  name = "especialidad"  use = "required" >
            <xs:simpleType >
              <xs:restricción  base = "xs: string" >
                <xs:Enumeration  value = "ASIR"/>
                <xs:Enumeration  value = "SMR"/>
              </xs:restricción >
            </xs:simpleType >
          </xs:attribute >
        </xs:complexType >
      </xs:elemento >
    </xs:sequence >
    <xs:attribute  name = "módulo"  use = "required"  type = "xs: integer" />
    <xs:attribute  name = "nommodul"  use = "required"  type = "xs: string" />
  </xs:complexType >
</xs:elemento >
</xs:schema >
```

En el archivo XML hay que asociar el esquema para validar:

```
<clase modulo= "3" nommodul= "Programación básica"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation= "file:/D:/Inma/1Dam_MARCAS/EjerPunto3-XSD.xsd">
```

3.4. Conversión entre esquemas

A la hora de hacer esquemas, hay que tener en cuenta, que ya existen herramientas que permiten convertir los lenguajes de definición de esquemas de un formato a otro.

A pesar de ello siempre se deberán revisar los resultados para comprobar que no se ha perdido ningún significado importante al hacer la conversión, ya que los sistemas automáticos no son perfectos.

La mayoría de los editores XML llevan alguna herramienta para hacerlo, pero también hay herramientas específicas para hacer conversiones, como el programa de código abierto.