Módulo 3. Validez de Documentos.

Como ya se ha expuesto, un documento XML correcto es aquel que cumple dos condiciones:

- Ser un documento válido.
- Ser un documento bien formado.

En el módulo anterior, ya se explicaron las reglas para formar un documento bien formado y se comentó, brevemente, cómo crear un documento válido. En este módulo se profundizará más en este último aspecto, explicando la Definición de Tipo de Documento (o DTD) y los XML Schemas (o Esquemas XML).

Las diferencias principales entre las DTD y los Schemas son:

- Las DTD tienen una sintaxis específica mientras que los Schema utiliza sintaxis XML.
- Un Schema se puede manipular, como cualquier otro documento XML.
- Hay muchas más herramientas para trabajar con DTD que con Schema.
- Un Schema soporta la integración de los espacios de nombres permitiendo asociar nodos de un documento con declaraciones de tipo de un esquema.
- La DTD sólo permite una asociación entre un documento y su DTD.

Tema 1. Definición de Tipo de Documento (Document Type Definition).

INTRODUCCIÓN

Como también se mencionó en el módulo anterior, es posible asociar un documento XML a una DTD determinado para así organizar la información contenida de forma que nos sea más útil de cara a nuestro objetivo. Sin embargo, esto no es imprescindible ya que puede haber documentos XML cuya estructura no esté definida en ningún DTD.

Por tanto, atendiendo al hecho de que un documento lleve o no asociado una DTD, podemos diferenciar dos tipos de documentos XML:

- Documentos válidos son aquellos que siguen las reglas de una DTD específica.
- Documentos bien formados son aquellos que no tienen asociado una DTD, pero cumplen todas las reglas de XML.

Evidentemente, un documento *válido* estará siempre *bien formado*.

Si queremos tener la seguridad de que un fichero XML pueda ser perfectamente interpretado por cualquier herramienta, podemos incluir una definición de su construcción que preceda a los datos propiamente dichos. Este conjunto de meta-datos es el que recibe el nombre de DTD.

Las DTD son la clave de auto-descripción en documentos XML, es decir, permiten definir al usuario qué significa exactamente cada una de las marcas que va a incluir a continuación para identificar los datos.

En definitiva, la definición de tipo de documento o DTD contiene las reglas sintácticas que deben cumplir los documentos de un tipo determinado

Un documento XML indica al procesador XML que va a utilizar algún tipo de definición de tipo de documento por medio de la línea de declaración de tipo de documento <!DOCTYPE…> con la cual se le indica que lea las definiciones de la DTD y compruebe que el documento XML cumple con las normas estructurales y sintácticas establecidas.

A este proceso se le conoce con el nombre de validación y a los documentos que superan con éxito la prueba de validación se les denomina documentos XML válidos y aquellos que no cumplan las normas establecidas en la DTD son conocidos como documentos XML no válidos.

EJEMPLO DE CÓMO DEFINIR UNA DTD

Una DTD es un documento en el cual se declaran entidades, notaciones, elementos y lista de atributos permitidos en aquellos documentos XML que la utilicen.

Por lo tanto, tal y como venimos reiterando, la DTD es un mecanismo para describir los objetos que pueden aparecer en un documento XML.

Antes de explicar cómo se declara una entidad, una notación, un elemento o lista de atributos mostraremos un ejemplo.

A continuación se muestra la solución XML que describe a una persona.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persona>
        <nombre> Juan </nombre>
        <apellido> Pérez </apellido>
        <profesion> Taxista </profesion>
</persona>
```

Como una DTD sólo describe el tipo general, no la instancia específica, la DTD para este ejemplo sería un elemento persona que contiene unos elementos hijos nombre, apellido y profesion. Cada uno de estos elementos contiene un texto. Por tanto, la DTD sería:

```
<!ELEMENT persona (nombre, apellido, profesion)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT profesion (#PCDATA)>
```

Esta DTD se puede guardar:

- en un archivo independiente del documento XML: de esta forma diferentes ficheros XML pueden referenciarlo si es necesario.

  Dentro del documento XML se haría una referencia al fichero .dtd que contendría las definiciones utilizando una declaración de tipo de documento.

  Una declaración de tipo de documento es:

  ```
  <!DOCTYPE nombre_elemento_raiz SYSTEM referencia_fichero_dtd>
  ```

  Por ejemplo:

  ```
  <!DOCTYPE persona SYSTEM “persona.dtd”>
  ```

  La declaración del tipo de documento se incluye en el prólogo del documento XML, después de la declaración XML, antes del elemento raíz.

  En el ejemplo:

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE persona SYSTEM "persona.dtd">
  <persona>
      <nombre> Juan </nombre>
      <apellido> Pérez </apellido>
      <profesion> Taxista </profesion>
  </persona>
  ```

  Existiendo el fichero “persona.dtd” en el mismo directorio que el documento XML, sino habría que utilizar una ruta relativa hasta llegar a él.

Código fuente de persona.dtd:

```
<!ELEMENT persona (nombre, apellido, profesion)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT profesion (#PCDATA)>
```

- o dentro del propio documento XML, usando la declaración de tipo utilizada anteriormente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE persona[
    <!ELEMENT persona (nombre, apellido, profesion)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT apellido (#PCDATA)>
    <!ELEMENT profesion (#PCDATA)>
]>
<persona>
    <nombre> Juan </nombre>
    <apellido> Pérez </apellido>
    <profesion> Taxista </profesion>
</persona>
```

Una vez expuesto todo esto, en los próximos apartados se detallará como se declaran los posibles objetos utilizados en un documento:

- Declaración de elementos.
- Declaración de lista de atributos.
- Declaración de entidades.

DECLARACIÓN DE ELEMENTOS

Todos los elementos usados en un documento válido, deben declararse en la DTD de un documento con una declaración de elemento.

La sintaxis genérica es:

```
<!ELEMENT nombre_elemento especificación_de_contenido>
```

El *nombre_elemento* se corresponde al nombre de la etiqueta que hemos puesto para dicho elemento.

La *especificación_de_contenido* indica el hijo que puede o debe tener y su orden. Estas especificaciones pueden ser bastante complejas: pueden establecer que un elemento debe tener tres elementos hijos de un tipo determinado o dos hijos de un tipo seguidos por otro elemento de otro tipo, etc.

A continuación, se explicarán las posibilidades existentes en la declaración de elementos.

➲ #PCDATA, EMPTY, ANY

- #PCDATA: Indica que un elemento puede contener sólo datos de caracteres, pero que no puede contener ningún elemento hijo para ningún tipo.

  Por ejemplo, la declaración del elemento nombre indica que puede contener texto pero no puede contener elementos.

```
<!ELEMENT nombre (#PCDATA)>
```

- EMPTY: indica que el elemento es un elemento vacío

  Como ya se explicó, un elemento vacío es un elemento que no tiene asociado contenido.

  Por ejemplo:

```
<imagen src="foto.jpg"/>
```

  La declaración de un elemento de este tipo sería:

```
<!ELEMENT imagen EMPTY>
```

- ANY: indica que el elemento puede contener cualquier otro elemento declarado en la DTD. Raramente se utiliza

➲ Elementos hijos:

- Un único hijo

  Otra opción, es cuando un elemento contiene exactamente un elemento hijo de un tipo determinado.

  En este caso, la especificación de contenido consta del nombre del elemento hijo entre paréntesis.

```
<!ELEMENT nombre_elemento (nombre_elemento_hijo)>
```

- Otras cantidades de hijos

  La mayoría de los elementos contienen datos de caracteres o múltiples elementos hijos. En este último caso, los hijos se especifican separándolos por comas (es decir, utilizando lo que se denominan secuencias). La secuencia indica que los elementos especificados deben aparecer en el orden establecido.

  Por ejemplo: Supongamos que tenemos un elemento nombre_persona definido por tres elementos hijos: nombre, apellido y mote.

```
<nombre_persona>
  <nombre> Francisco </nombre>
  <apellido> Ramirez </apellido>
  <mote> Paquito </mote>
</nombre_persona>
```

La declaración del elemento nombre_persona sería:

```
<!ELEMENT nombre_persona (nombre, apellido, mote)>
```

No todas las instancias de elementos tienen porque tener valor para los mismos elementos hijos.

Podemos fijar uno o tres sufijos a un nombre de elemento en una especificación de contenido para indicar qué cantidad de hijos de dicho elemento se esperan en dicha posición:

Sufijo ?: admite 0 ó 1 hijo de elemento
Sufijo *: admite 0 ó más hijos de elementos.
Sufijo +: admite 1 o más hijos de elementos.

Por ejemplo: La declaración de elemento podría ser que el elemento nombre_persona tiene que contener exactamente un nombre y apellido; y puede o no contener un mote.

```
<!ELEMENT nombre_persona (nombre, apellido, mote?)>
```

➲ Opciones

¿Y qué sucede si un elemento tiene elementos hijo diferentes? Es decir, una instancia de un objeto puede contener un tipo de hijo y otra instancia diferente puede tener otro hijo de diferente tipo.

La solución para estos casos es una lista de nombres de elemento separados por barras verticales.

Por ejemplo: En una tienda en la que se alquilan películas o videojuegos tendríamos:

```
<!ELEMENT producto (pelicula | videojuego)>
```

Es importante no olvidar que el producto será una pelicula o un videojuego, no podrá ser ambas cosas a la vez.

➲ Paréntesis

Todas las opciones anteriores se pueden combinar usando paréntesis.

Por ejemplo: definimos un elemento círculo con un elemento centro y con otro elemento que puede ser el radio o diámetro del círculo.

```
<!ELEMENT circulo (centro , ( radio| diametro))>
```

DECLARACIÓN DE LISTA DE ATRIBUTOS

Un documento válido también debe declarar todos los atributos de los elementos. Para ello, se harán declaraciones *ATTLIST*.

Una sola declaración *ATTLIST* puede declarar múltiples atributos para un solo tipo de elemento. Sin embargo, si el mismo atributo se repite en múltiples elementos, debe declararse independientemente para cada elemento donde aparece.

La sintaxis general para declarar los atributos es:

```
<!ATTLIST nombre_elemento nombre_atributo tipo_atributo propiedades>
```

En *nombre_elemento* se especificará el nombre de la etiqueta correspondiente al elemento.

En *nombre_atributo* se especificará el nombre definido para el atributo en cuestión.

En tipo de atributo puede ser:

- CDATA (Character DATA): un valor de este tipo puede contener cualquier cadena de texto aceptable en una buena estructura XML.

  Ejemplo:
  ```
  <!ATTLIST imagen texto_alternativo CDATA #IMPLIED>
  ```

- NMTOKEN (NaMe Token): es un testigo de nombre XML. Un testigo de nombre es un nombre XML con la diferencia en que ninguno de los caracteres permitidos puede ser el primer carácter en el testigo del nombre, mientras que sólo las letras y el signo subrayado pueden ser el primer carácter de n nombre XML. Por lo tanto 34 y .cshrc son testigos de nombre XML válidos, aunque no son nombres de XML válidos.

  Ejemplo:

  ```
  <!ATTLIST dia año NMTOKEN #REQUIRED>
  ```

- Enumeración: es el único tipo de atributo que no es una palabra clave XML, sino una lista con los valores posibles para dicho atributo.

  Ejemplo:

  ```
  <!ATTLIST fecha mes ( 1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED>
  ```

- ENTITY: contiene el nombre de una entidad sin analizar, declarada en alguna parte en la DTD.

  Ejemplo:

  ```
  <!ATTLIST pelicula origen ENTITY #REQUIRED>
  <pelicula origen=”presentacion.avi” />
  ```

- ENTITIES: contiene nombres de una o más entidades sin analizar, declaradas en cualquier parte en la DTD, separados por un espacio en blanco.

- ID: contiene un nombre XML, único dentro del documento XML. Ningún otro atributo de tipo ID puede tener el mismo valor en el documento.

  Ejemplo:

  ```
  <!ATTLIST persona dni ID  #REQUIRED>
  ```

- IDREF: se refiere al atributo de tipo ID de algún elemento en el documento. Por lo tanto, debe ser un nombre XML. Normalmente, se usan para establecer relaciones entre los elementos.
- IDREFS: contiene una lista de nombres XML separados por un espacio en blanco, cada uno de los cuales debe tener un ID de un elemento del documento.
- NOTATION: contiene el nombre de una notación declarada en la DTD del documento. Tal vez sea el atributo menos usado.

Las *propiedades* pueden ser:

- #IMPLIED: significa que el atributo es opcional. Es decir, cada instancia de dicho elemento puede o no tener un valor para el atributo.

- #REQUIRED: significa que el atributo es necesario. Es decir, cada instancia debe proporcionar un valor para el atributo de ese elemento.

- #FIXED: significa que el valor del atributo es constante.

- Literal: el valor predeterminado real se proporciona como cadena entre paréntesis.

DECLARACIÓN DE ENTIDADES

XML, tal y como se explicó en el módulo anterior, dispone de cinco entidades predefinidas:

- & para el &
- < para el <
- > para el >
- ' para el '
- " para el "

A través, de la declaración de entidades podremos asociar al nombre de la entidad que establezcamos el texto por el que lo reemplaza. Es decir, podemos definir:

<!ENTITY firm “superguaydelamuerte”>

Esta declaración de entidad define &firm; como una abreviatura de *superguaydelamuerte*.

Una vez definida la entidad, podemos usar &firm; sin necesidad de tener que escribir *superguaydelamuerte*, que es más largo y “engorroso”.

Tema 2. Espacios de Nombre.

En XML se llama espacio de nombres a una colección de nombres que proporciona un mecanismo por el que los nombres de elementos y atributos pueden asignarse para diferentes usos dependiendo de cada caso, utilizando prefijos los adecuados.

En XML, los espacios de nombre tienen dos propósitos:

- Distinguir entre elementos y atributos con distintos significados, pero que tienen el mismo nombre.
- Agrupar todos los elementos y atributos relacionados de una sola aplicación XML para que el software pueda reconocerlos con facilidad.

Los espacios de nombres se implantaron mediante la incorporación de un prefijo a cada elemento y atributo.

Entres sus características se encuentran:

- Son ampliamente utilizados para combinar vocabularios.
- Facilitan la incorporación de elementos no previstos inicialmente.
- Su sintaxis puede resultar extraña cuando se empiezan a utilizar.
- Uso de prefijos.
- URIs como elemento diferenciador.
- Difícil combinación con DTDs

Para que resulte fácil su aplicación vamos a imaginar un supuesto en el que tenemos:

```
<pais nombre="España">
<capital>Madrid</capital>
</pais>
```

```
<inversion>
<capital>700000€</capital>
</inversion>
```

¿Cómo se combinarían los ejemplos anteriores en un mismo documento? ¿Cómo distinguiríamos el capital cuando nos referimos a la capital de un país determinado del capital de una inversión económica?

```
<inversiones>
        <país nombre="España">
            <capital>Madrid</capital>
            <capital>700000€</capital>
        </pais>
. . .
</inversiones>
```

Sería una solución bastante ambigua.

Otra posible solución sería asignar un nombre único a cada etiqueta. Para ello habría que:

- Crear una autoridad mundial que asigne nombres.
- Utilizar un mecanismo ya existente: los URIs o identificadores globales únicos.

Según la segunda opción, se podría asociar a cada etiqueta una URI que indica a qué espacio de nombres pertenece. Esto es:

[http:www.bolsa.com]:capital
[http:www.geografia.es]:capital

Sería una solución fácil, pero bastante engorrosa como se puede ver a continuación.

```
<[http://www.bolsa.com]:inversiones>
        <[http://www.geografia.es]:pais
                [http://www.geografia.es]:nombre="España">
          <[http://www.geografia.es]:capital>Madrid
          </[http://www.geografia.es]:capital>
          <[http://www.bolsa.com]:capital>700000€
          </[http://www.bolsa.com]:capital>
        </[http://www.bolsa.com]:pais>
. . .
</[http://www.bolsa.com]:inversiones>
```

Otra solución consistiría en asociar un alias a los elementos de un espacio de nombres dentro de un ámbito.

Es decir, xmlns:*alias* define *alias* en el ámbito de un elemento.

Se debe considerar que las URIs sólo se utilizan para que el nombre sea único, no son enlaces, ni tienen que contener información.

```
<bolsa:inversiones
            xmlns:bolsa="http://www.bolsa.com"
            xmlns:geog="http://www.geografia.es">
  <geog:país geog:nombre="España">
      <geog:capital>Madrid</geog:capital>
      <bolsa:capital>700000€</bolsa:capital>
  </geog:país>
```

```
. . .
</bolsa:inversiones>
```

Puede darse el caso de que sea necesario ir asociando espacios de nombres a los elementos según van apareciendo, es lo que se denomina Asignación Dinámica.

```
<bolsa:inversiones
                xmlns:bolsa="http://www.bolsa.com">
    <geog:país
                        xmlns:geog="http://www.geografia.es"
                        geog:nombre="España">
       <geog:capital>Madrid</geog:capital>
       <bolsa:capital>700000€</bolsa:capital>
  </geog:país>
. . .
</bolsa:inversiones>
```

Mediante **xmlns="..."** se define un espacio de nombres por defecto (sin alias)

```
<inversiones
        xmlns="http://www.bolsa.com">
  <geog:país
           xmlns:geog="http://www.geografia.es"
           geog:nombre="España">
     <geog:capital>Madrid</geog:capital>
     <capital>700000€</capital>
   </geog:país>
. . .
</inversiones>
```

En este caso <capital>700000€</capital> se refiere a **http://www.bolsa.com**

Los espacios de nombre y la validación con DTDs

Los espacios de nombres son posteriores a los DTDs. Por este motivo los DTDs no dan soporte a espacios de nombres. Se deben definir los espacios de nombre usados.

```
<!DOCTYPE inversiones [
<!ELEMENT inversiones (geog:país*)>
<!ELEMENT geog:país (geog:capital,capital) >
<!ELEMENT geog:capital (#PCDATA)>
<!ELEMENT capital (#PCDATA)>
<!ATTLIST inversiones
        xmlns CDATA #FIXED "http://www.bolsa.com">
<!ATTLIST geog:país
        geog:nombre CDATA #REQUIRED
```

```
        xmlns:geog CDATA #FIXED "http://www.geografia.es">
]>
```

Por lo tanto, los espacios de nombre consiguen distinguir entre sí a los elementos con el mismo nombre mediante la asignación de elementos y atributos a los URI.

Por lo general, todos los elementos de una aplicación XML se asignan a un URI y todos los elementos de una aplicación XML diferente se asignan a un URI diferente.

Los URIs, por lo tanto, se denominan nombres de espacios de nombre. Dividen los elementos y atributos en conjuntos inconexos.

Los elementos con el mismo nombre; pero con URI diferentes son tipos diferentes. De igual forma, Los elementos con el mismo nombre y los mismos URI son iguales.

Normalmente existe una asignación de nombres uno a uno entre los espacios de nombre y las aplicaciones XML, aunque puede darse el caso de que algunas aplicaciones utilizan múltiples espacios de nombres para subdividir las distintas partes de la aplicación.

Sobre el alcance de los espacios de nombres se debe apuntar lo siguiente:

- El elemento para el que se defina y los elementos que contenga, a menos que se realice otra declaración de espacio de nombres con el mismo prefijo.
- El espacio de nombres predeterminado se elimina cuando se establece como nombre una cadena vacía (“”).
- Los atributos no están unidos, como opción predeterminada, a ningún espacio de nombres.

Veamos esto en un ejemplo:

```
<?xml version=”1.0” encoding=”iso-8859-1” standalone=”yes”?>
<x xmlns=”http://www.sunion-gesfor.com/personal”
  xmlns:servicios=”http://www.sunion-gesfor.com/servicios”>
<A servicios:B=”formacion” C=”tecnología”/>
</x>
```

A está asociada al xmlns predeterminado (“http://www.sunion-gesfor.com/personal”)

B está asociado al xmlns “http://www.sunion-gesfor.com/servicios”.

C no está asociado a ningún espacio de nombres.

En los espacios de nombres XML, ninguna etiqueta puede contener los mismos nombres universales.

Es decir, ninguna etiqueta debe contener dos atributos con:

- El mismo nombre.
- Nombres equivalentes calificados, es decir, las mismas partes locales y los mismos prefijos que corresponden al mismo URI.

Veamos esto con un ejemplo:

Ejemplo 1

```
<n1:x xmlns:n1="http://www.sunion-gesfor.com"
  Xmlns:n2="http://sunion-gesfor.com">
```

Los elementos XML siguientes no serían validos:

```
<n1:X a="1" a="2"/>

<n2:X n1:a="1"  n2:a="2"/>
```

Ejemplo 2

```
<n1:x xmlns:n1="http://www.sunion-gesfor.com"
  Xmlns:n2="http://sunion-gesfor.com">
```

Los elementos XML siguientes serían validos:

```
<n1:X a="1" b="2"/>

<n2:X n1:a="1"  a="2"/>
```

Tema 3. XML Schemas.

INTRODUCCIÓN

Antes de comenzar a exponer las diferencias entre las DTDs y los XML Schemas es importante revisar de dónde y cómo surgieron las dos principales soluciones en las que nos hemos detenido.

Inicialmente el uso de las DTDs se debe a SGML, en el cual describían no sólo el vocabulario necesario para identificar todos los elementos de que iba a constar nuestro documento, si no que también expresaba la estructura que dichos elementos debía respetar.

Con la creación de XML, que no es más que un subconjunto de SGML, fue necesario mantener la posibilidad de describir los elementos necesarios, al igual que en SGML, por ello se importaron las DTDs y todo su comportamiento.

Posteriormente, y debido al uso principalmente, se vio la necesidad de emplear otros métodos para describir esas necesidades inherentes a XML, con esta idea se crearon los XML Schemas. Con ellos se pretendía mejorar y ampliar la utilidad de las DTDs.

Un Esquema XML o XML Schema es un documento que contiene una descripción formal de lo que comprende un documento XML válido. Sería una definición de la estructura de un conjunto de documentos XML. Son similares a los DTDs en el sentido que definen qué elementos puede contener un documento XML, cómo están organizados, y qué atributos y de qué tipo pueden tener sus elementos.

Un documento XML se describe mediante un esquema denominado documento de instancia. Si un documento satisface todas las restricciones especificadas por el esquema, se considera que es un documento con un esquema válido. El documento del esquema se asocia con un documento de instancia a través de uno de los siguientes métodos:

- Atributo *xsi:schemaLocation* en un elemento contiene una lista de espacios de nombre usados dentro de dicho elemento y los URL de los esquemas con los que validan los elementos y atributos en dichos espacios de nombre.

- Atributo *xsi:noNamespaceSchemaLocation* contiene una URL para el esquema usado para validar elementos que no se encuentran en ningún espacio de nombres.

- Se puede instruir a un analizador de validación para que valide un determinado documento frente a un esquema proporcionado explícitamente, ignorando cualquier sugerencia que podría proporcionarse dentro del propio documento.

ESQUEMAS Y DTD

Originalmente se utilizaron los DTDs, posteriormente se han desarrollado los XML Schemas.

Las DTD proporcionan la capacidad de realizar una validación básica de los siguientes elementos:

- Anidamiento de elementos.
- Limitaciones de ocurrencia de elementos.
- Atributos permitidos.
- Tipos de atributos y valores predeterminados.

Sin embargo, las DTD no proporcionan un buen control sobre el formato y tipos de dato del elemento y valores de atributos. En los tipos de atributo distintos a los especiales (ID, IDREF, ENTITY…) cuando un elemento o atributo se ha declarado para contener datos de caracteres, no se deben incluir límites sobre la longitud, el tipo o el formato del contenido.

Para documentos narrativos, como libros o páginas Web, este nivel control probablemente sea suficiente. Pero, para el caso de llamadas a procedimientos remotos o serialización de objetos, es importante tener un control preciso sobre el contenido del texto de los elementos y atributos.

Los DTDs tienen las siguientes limitaciones:

- Su sintaxis, al no ser XML, son difíciles de manipular.
- No soportan los espacios de nombres.
- No permiten especificar tipos de datos (por ejemplo: enteros, flotantes, fechas, etc.)
- No permiten especificar secuencias no ordenadas como ((e1,e2,e3)|(e1,e3,e2)|(e2,e1,e3)|...(e3,e2,e1)).
- No hay soporte para declaraciones sensibles al contexto. Los elementos se definen todos a nivel de documento, ejemplo, contenido con el mismo nombre cuya estructura cambia en diferentes contextos.
- Soporte limitado para referencias cruzadas, no es posible formar claves a partir de varios atributos o de elementos.
- No son extensibles (una vez definido, no es posible añadir nuevos vocabularios a un DTD).

En este sentido, el estándar de esquemas XML incluye las siguientes particularidades:

- Sintaxis XML.
- Soporte para espacios de nombres.
- Gran cantidad de tipos de datos predefinidos y creación de tipos de datos por el usuario.

- Extensibilidad: Inclusión/Redefinición de esquemas y herencia de tipos de datos.

ELEMENTOS BÁSICOS DE LOS ESQUEMAS

Veamos a través de un ejemplo los elementos básicos de los esquemas:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.sunion-gesfor/alumnos"
        xmlns="http://www.sunion-gesfor.com/alumnos">
<xs:element name="alumnos">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="alumno" minOccurs="1" maxOccurs="200"
                    type="TipoAlumno"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="TipoAlumno">
  <xs:sequence>
     <xs:element name="nombre" type="xs:string"/>
     <xs:element name="apellidos" type="xs:string"/>
     <xs:element name="nacim" type="xs:gYear"/>
   </xs:sequence>
   <xs:attribute name="dni" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

Alumnos.xsd

En este apartado del fichero alumnos.xsd se encuentra el elemento raíz schema y el espacio de nombres determinado:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.sunion-gesfor/alumnos"
        xmlns="http://www.sunion-gesfor.com/alumnos">
```

Permite especificar rangos de inclusión:

```
<xs:element name="alumno" minOccurs="1" maxOccurs="200"
                    type="TipoAlumno"/>
```

Permite también especificar tipos de datos:

```
      <xs:element name="nacim" type="xs:gYear"/>
```

El fichero alumnos.xml sería de la siguiente forma teniendo en cuenta que los espacios de nombres deben coincidir:

```
<alumnos
   xmlns=”http://sunion-gesfor.com/alumnos”
   xsi:SchemaLocation=”http://www.sunion-gesfor.com/alumnos/alumnos.xsd”
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance”>
…
</alumnos>
```

En los XML se pueden utilizar Tipos anónimos y con nombre:

Tipos anónimos (código más legible)

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

Tipos con nombre (código más reutilizable)

```
…
<xs:element name="alumno" type="TipoAlumno"/>
…
<xs:ComplexType name="TipoAlumno">
      <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellidos" type="xs:string"/>
      </xs:sequence>
</xs:ComplexType>
```

En los XML Schema es posible nombrar agrupaciones de elementos y de atributos para hacer referencias a ellas.

```
<xs:group name="nombApell">
      <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellidos" type="xs:string"/>
      </xs:sequence>
</xs:group>
```

```
<xs:complexType name="TipoAlumno">
      <xs:group ref="nombApell" />
      <xs:element name="curso" type="xs:string"/>
</xs:complexType>
```

Los tipos complejos son aquellos que pueden contener elementos o atributos. Su construcción básica se realiza mediante una enumeración de elementos.

```
<xs:complexType name="TipoAlumno">
        <xs:sequence>
           <xs:element name="nombre" type="xs:string"/>
           <xs:element name="apellidos" type="xs:string"/>
           <xs:element name="nacim" type="xs:gYear"
                                  minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="dni" type="xs:integer"/>
</xs:complexType>
```

```
<alumno dni="53032765">
    <nombre>Lorenzo</nombre>
    <apellidos>Fernandez Gil</apellidos>
    <nacim>1979</nacim>
</alumno>
```

Una alternativa es utilizar Choice, que conlleva una o exclusiva.

```
<xs:complexType name="Transporte">
     <xs:choice>
          <xs:element name="autobus" type="xs:string"/>
          <xs:element name="tren" type="xs:string"/>
          <xs:element name="avion" type="xs:string"/>
     </xs:choice>
</xs:complexType>
```

```
<transporte>
<tren>Talgo 770</coche>
</transporte>
```

Trabajando con tipos complejos, el contenido mixto permite mezclar texto con elementos.

```
<xs:complexType name="TCom" mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="emph" type="xs:string"/>
   </xs:choice>
</xs:complexType>

<xs:element name="comentarios" type="TCom" />
```

```
<comentarios>
Se distrae con <emph>mucha facilidad</emph>
</comentarios>
```

Para secuencias no ordenadas se utiliza se utiliza “all” para que aparezcan todos los elementos en cualquier orden.

En los DTDs se requeriría enumerar las combinaciones: (A,B,C) / (A,C,B) /…/ (C,B,A)

```
<xs:complexType name="TipoLibro">
         <xs:all>
                  <xs:element name="autor" type="xs:string"/>
                  <xs:element name="titulo" type="xs:string"/>
         </xs:all>
</xs:complexType>
<xs:element name="libro" type="TipoLibro" />
```

<libro> <autor>Nuria Roca</autor> <título>Los caracoles no saben que son caracoles</título> </libro>	<libro> <título>La sombra del viento</título> <autor>Carlos Ruiz Zafon</autor> </libro>

En los XML Schema, los Tipos simples no pueden contener elementos o atributos. Pueden ser:

- Predefinidos (o también llamados built-in o definidos en la especificación):

  ➔ Primitivos: string, boolean, Lumber, flota, double, duration, dateTime, time, date. gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, QName (nombre cualificado con espacio de nombres) o Notation (notación binaria).

  ➔ Derivados: normalizedString, token, language, IDRefs, Entities, NMToken, NMTokens, Name, NCName, ID, IDRef, Entity, integer, nonPositiveInteger,

negativeInteger, log, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger.

- Definidos por el usuario a partir de tipos predefinidos.

En los Esquemas XML, dependiendo de la relación entre los valores se puede utilizar:

- Equal: Igualdad entre valores de un tipo de datos.
- Ordered: Relaciones de orden entre valores.
- Bounded: Límites inferiores y superiores para valores.
- Cardinality: Define si es finito o infinito (contable o no contable).

Y para limitar valores y expresiones:

- Length, minlength o maxlength: Longitudes del tipo de datos.
- Pattern: Restricciones sobre valores mediante expresiones regulares.
- Enumeration: Restringe a una determinada enumeración de valores.
- Whitespaces: Define la política de tratamiento de espacios (preserve / replace / collapse).
- (max/min)(in/ex)clusive: Límites superiores/inferiores del tipo de datos.
- totaldigits, fractionDigits: número de dígitos totales y decimales.

Veamos a continuación ejemplos en los que se emplea enumeration y restriction:

Enumeration

```
<xs:simpleType name="TipoCarrera">
<xs:restriction base="xs:token">
    <xs:enumeration value="Informatica de Gestion"/>
    <xs:enumeration value="Informatica de Sistemas"/>
</xs:restriction>
</xs:simpleType>
```

Restriction

```
<xs:simpleType name="mes">
<xs:restriction base="xs:integer">
     <xs:minInclusive value="1" />
     <xs:maxInclusive value="31" />
</xs:restriction>
```

```
</xs:simpleType>
```

Veamos un ejemplo de listas con Restriction

```
<xs:simpleType name="ComponentesRGB">
      <xs:list itemType="ComponenteRGB"/>
</xs:simpleType>

<xs:simpleType name="ComponenteRGB">
<xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="255" />
</xs:restriction>
</xs:simpleType>
```

Se puede aplicar length, maxLength y minLength

```
<xs:simpleType name="ColorRGB">
      <xs:restriction base="ComponentesRGB">
            <xs:length value="3" />
      </xs:restriction>
</xs:simpleType>
```

```
<color>255 255 0</color>
```

Y a continuación veamos un ejemplo de Uniones:

```
<xs:simpleType name="TipoNota">
<xs:union>
      <xs:simpleType>
          <xs:restriction base="xs:float">
                <xs:maxInclusive value="10" />
                <xs:minInclusive value="0" />
          </xs:restriction>
       </xs:simpleType>
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="No presentado" />
</xs:restriction>
</xs:simpleType>
</xs:union>
</xs:simpleType>
<xs:element name="nota" type="TipoNota" />
```

<nota>8.50</nota>
<nota>No presentado</nota>

Pueden aparecer las siguientes expresiones regulares:

Expresión	Posibles valores
Elemento /d	Elemento 2
a*b	b, ab, aab, aaab,…
[xyz]b	xb,yb,zb
a?b	b, ab
a+b	ab, aab, aaab,…
[a-c]x	ax, bx, cx
[^0-9]x	Carácter =/= dígito seguido de x
\Dx	Carácter =/= dígito seguido de x
.abc	Cualquier carácter seguido de abc
\n	Salto de línea

Veamos un ejemplo:

```
<xs:simpleType name="NIF">
    <xs:restriction base="xs:token">
        <xs:pattern value="\d{7,8}[A-Z]" />
        </xs:restriction>
    </xs:simpleType>
<xs:element name="nif" type="NIF" />
```

<nif>9003664S</nif>
<nota>11079846X</nota>

También podemos encontrarnos tipos derivados por extensión, sería similar a las subclases de Programación Orientada a Objetos, que consiste en añadir elementos a un tipo base.

```
<xs:complexType name="Figura" >
    <xs:attribute name="color" type="Color"/>
</xs:complexType>

<xs:complexType name="Rectangulo">
    <xs:complexContent>
    <xs:extension base="Figura">
    <xs:attribute name="base" type="xs:float" />
    <xs:attribute name="altura" type="xs:float" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="Circulo">
```

```
...habría que incluir el radio
</xs:complexType>
```

Los tipos derivados pueden utilizarse donde se utiliza la clase base:

```
<xs:element name="figuras">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="figura" type="Figura"
                        maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<figuras>
     <figura base="23" altura="3" xsi:type="Rectangulo" />
     <figura radio="3" xsi:type="Circulo" />
</figuras>
```

Es necesario especificar el tipo mediante xsi:type

En los XML Schema también pueden aparecer tipos abstractos. Se declaran mediante abstract=”trae”. No puede usarse directamente:

```
<xs:complexType name="Figura" abstract="true">
    <xs:attribute name="color" type="Color"/>
</xs:complexType>
```

Llegados a este punto nos detendremos en la declaración de atributos:

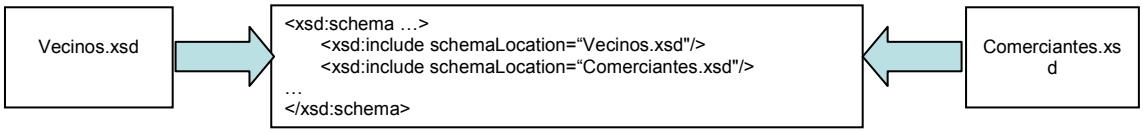
```
<xs:complexType name="Círculo">
   <xs:attribute name="radio"
                 type="xs:float"
                 use="required" />

<xs:attribute name="color"
                 type="Color"
                 default="255 0 0"/>

<xs:attribute name="tipo"
                 type="xs:string"
                 fixed="jpeg" />

</xs:complexType>
```

Por defecto los atributos son opcionales. Para indicar que son obligatorios se utiliza use="required"

Con default se establece el valor por defecto de un atributo.

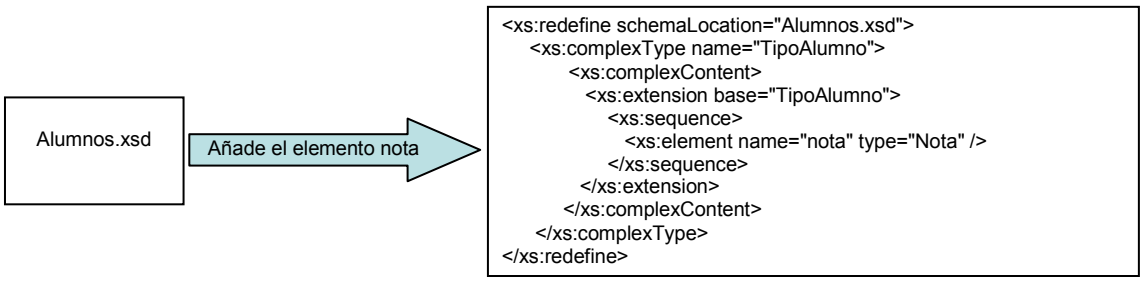
Para determinar el valor fijo de un atributo se utiliza fixed.

Para incluir elementos de otros esquemas se emplea *include*. Los elementos deben estar en el mismo espacio de nombres.



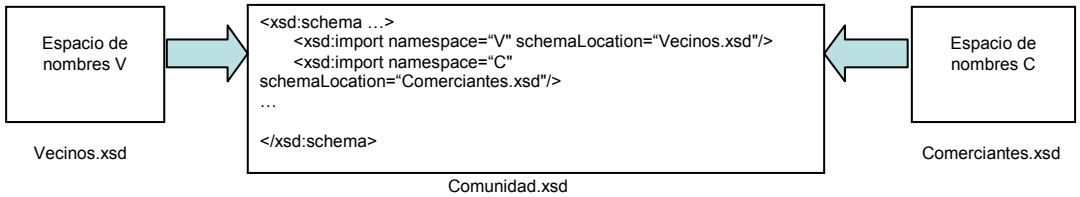
Comunidad.xsd

Redefine es similar a *include*; pero permite modificar los elementos incluidos:



Alumnosconnota.xsd

Import permite incluir elementos de otros esquemas con distintos espacios de nombres:



Para finalizar, sobre las claves y la unicidad en los XML Schema se comentará lo siguiente:

- Los DTDs proporcionan el atributo ID para marcar la unicidad (un valor ID era único en todo el documento).

- Los XML Schema tienen más posibilidades:

  ➔ Indicar que un elemento es único (con unique).
  ➔ Definir atributos únicos.
  ➔ Definir combinaciones de elementos y atributos como únicos.
  ➔ Distinguir entre unicidad y claves (con key). Además de ser único, debe existir y no puede ser nulo.
  ➔ Declarar el rango de un documento en el que algo es único

Tema 4. Recuerda Que…

Un documento XML correcto es aquel que cumple dos condiciones:

- Ser un documento válido.
- Ser un documento bien formado.

Las diferencias principales entre las DTD y los Schemas son:

- Las DTD tienen una sintaxis específica mientras que los Schema utiliza sintaxis XML.
- Un Schema se puede manipular, como cualquier otro documento XML.
- Hay muchas más herramientas para trabajar con DTD que con Schema.
- Un Schema soporta la integración de los espacios de nombres permitiendo asociar nodos de un documento con declaraciones de tipo de un esquema.
- La DTD sólo permite una asociación entre un documento y su DTD.

Si queremos tener la seguridad de que un fichero XML pueda ser perfectamente interpretado por cualquier herramienta, podemos incluir una definición de su construcción que preceda a los datos propiamente dichos. Este conjunto de meta-datos es el que recibe el nombre de DTD.

Las DTD son la clave de auto-descripción en documentos XML, es decir, permiten definir al usuario qué significa exactamente cada una de las marcas que va a incluir a continuación para identificar los datos.

Todos los elementos usados en un documento válido, deben declararse en la DTD de un documento con una declaración de elemento.

La sintaxis genérica es:

```
<!ELEMENT nombre_elemento especificación_de_contenido>
```

Un documento válido también debe declarar todos los atributos de los elementos. Para ello, se harán declaraciones *ATTLIST*.

XML, tal y como se explicó en el módulo anterior, dispone de cinco entidades predefinidas:

- & para el &
- < para el <
- > para el >
- ' para el '
- " para el "

En XML se llama espacio de nombres a una colección de nombres que proporciona un mecanismo por el que los nombres de elementos y atributos pueden asignarse para diferentes usos dependiendo de cada caso, utilizando prefijos los adecuados.

En XML, los espacios de nombre tienen dos propósitos:

- Distinguir entre elementos y atributos con distintos significados, pero que tienen el mismo nombre.

- Agrupar todos los elementos y atributos relacionados de una sola aplicación XML para que el software pueda reconocerlos con facilidad.

Un Esquema XML o XML Schema es un documento que contiene una descripción formal de lo que comprende un documento XML válido. Sería una definición de la estructura de un conjunto de documentos XML.

Un documento XML se describe mediante un esquema denominado documento de instancia. Si un documento satisface todas las restricciones especificadas por el esquema, se considera que es un documento con un esquema válido. El documento del esquema se asocia con un documento de instancia a través de uno de los siguientes métodos:

- Atributo *xsi:schemaLocation* en un elemento contiene una lista de espacios de nombre usados dentro de dicho elemento y los URL de los esquemas con los que validan los elementos y atributos en dichos espacios de nombre.

- Atributo *xsi:noNamespaceSchemaLocation* contiene una URL para el esquema usado para validar elementos que no se encuentran en ningún espacio de nombres.

- Se puede instruir a un analizador de validación para que valide un determinado documento frente a un esquema proporcionado explícitamente, ignorando cualquier sugerencia que podría proporcionarse dentro del propio documento.