

# Curso básico de tecnologías XML

## 4. Creando documentos XML válidos. Esquemas W3C XML

**INAP**

INSTITUTO NACIONAL DE  
ADMINISTRACIÓN PÚBLICA

## Contenido

1. Introducción.....	4
2. Objetivos.....	4
3. Espacio de nombres.....	4
3.1. Contenido.....	4
3.2. Espacio de nombres.....	6
4. Sintaxis de los esquemas W3C XML.....	7
5. Tipos de datos.....	9
5.1. Introducción.....	9
5.2. Categorías de los tipos de datos.....	9
5.3. Componentes del tipo de datos.....	13
6. Declaración de elementos de y atributos.....	14
6.1. Declarar elementos en un esquema XML.....	15
6.2. Declarar atributos en un esquema XML.....	15
7. Creando tipos simples de datos.....	15
7.1. Introducción.....	15
7.2. Crear tipos de listas.....	17
7.3. Combinar tipos simples de datos.....	18
7.4. Fijar el valor de un tipo.....	19
8. Crear tipos complejos de datos.....	19
9. Modelos de contenidos (cardinalidad, orden y agrupación).....	21
9.1. Atributos MinOccurs y MaxOccurs.....	21
9.2. Elementos Choice y All.....	22
9.2.1. Elemento Choice.....	22
9.2.2. Elemento All.....	22
10. Polimorfismo.....	23

11. Herramientas para la edición y validación de esquemas.....	24
11.1. Introducción.....	24
11.2. XML Copy Editor.....	25
11.3. Altova XMLSpy 2007.....	25
11.4. EXcelon Stylus Studio 2007.....	26
12. Conclusión.....	27
13. Ejercicio resuelto.....	28
13.1. Posible solución.....	28
13.2. Comentarios.....	30
14. Ejercicio propuesto.....	32



Este curso ha sido cedido por el Instituto Nacional de Administración Pública por medio de una licencia Creative Commons Reconocimiento-No comercial-Compartir igual, en los términos que se describen en <http://creativecommons.org/licenses/by-nc-sa/3.0/es> o texto oficial que, para esta modalidad de licencia, sustituya al indicado.

## 1. Introducción.

Hasta ahora sabemos crear definiciones de tipo de documento para validar los documentos XML. Ahora veremos una nueva posibilidad que nos ofrece la familia de tecnologías XML para llevar a cabo dicha validación. Se trata de los esquemas XML o XML Schema. Estos tienen una gran ventaja sobre los DTD y es que se escriben en lenguaje XML y no necesitan de un lenguaje particular como ocurría con las DTD. Debemos dejar claro que con ambas técnicas se consiguen resultados similares, por lo que una vez explicados los esquemas XML, es decisión del programador emplear un tipo u otro.

## 2. Objetivos.

En la siguiente unidad, profundizaremos en los siguientes conocimientos:

1. Conocer los **esquemas XML** (*Schema XML*).
2. Construir esquemas que contienen las definiciones tipo para construir documentos **XML válidos**.
3. Aprender a definir tipos simples y tipos complejos de datos para los esquemas de XML.
4. Entender los **esquemas W3C XML** como una alternativa a los **ficheros DTD** para la definición de sintaxis y estructura de los ficheros XML. Estos constituyen una herramienta más potente y flexible que los DTDs.

Una vez finalizada esta unidad didáctica, el alumno será capaz de crear **Esquemas W3C XML** que puedan ayudarnos a crear documentos XML interesantes.

## 3. Espacio de nombres.

### 3.1. Contenido.

Los **espacios de nombres** son imprescindibles a la hora de hacer de XML un lenguaje extensible. La definición más sencilla que podemos anticipar es que son contenedores de nombres de elementos y atributos. Únicamente contienen nombres. Cuando se usa una DTD, no se hace otra cosa que declarar una serie de elementos y atributos que son usados posteriormente en un documento XML. Para que nosotros podamos hacer uso de los contenidos de la DTD, así como controlar los contenidos del documento XML, el espacio de nombres debe situarse en un área que sea accesible para que el navegador o la aplicación que lea ese documento XML pueda comprobar su va-

lidad mediante la **DTD asociada**. El conjunto de nombres de elementos y atributos son los que forman el espacio de nombres.

Su funcionamiento es sencillo; se define un identificador único que se sitúa delante de una etiqueta. De este modo se permite utilizar nombres iguales para referirnos a cosas distintas. El identificador único no es más que un **URI** (*Universal Resource Identifier*), que en ocasiones se puede asociar a la dirección de Internet, a direcciones de correo o a otros identificadores únicos. Lo importante es que cada identificador este referido a un único espacio de nombres.

El término URI es una nueva denominación del concepto de URL. Se trata de un texto relativamente corto que identifica de manera única un recurso, sea del tipo que sea, en el ámbito de accesibilidad de una red. **El URI se compone de dos partes**; una primera parte que identifica el protocolo de acceso al recurso (ftp, http, etc.) y una segunda parte que identifica el nombre del recurso. La diferencia más clara entre un URL y un URI radica en que el primero se usa para direccionar un recurso dentro de Internet, mientras que el segundo abarca un concepto un poco más amplio, debido a que no tiene porque apuntar algo en concreto. En ese sentido, un URI es simplemente un nombre o identificador y no necesariamente un objeto.

A continuación mostraremos cuál es la sintaxis para definir un espacio de nombres para un elemento:

```
<Elemento xmlns:nombreLocal="URI">
...
</Elemento>
```

El siguiente ejemplo nos ayudará a verlo de forma más clara:

```
<Coches xmlns:espacio1="http://www.espacioNombre1.com/">
...
</Coches>
```

Como vemos en el recuadro anterior, para declarar un espacio de nombres se usa el atributo reservado `xmlns` seguido de dos puntos y del prefijo que usamos para identificar el espacio de nombres.

El espacio de nombres se aplicará al elemento y a todos sus elementos hijos. Es decir, si nosotros definimos un espacio de nombres en el elemento raíz de un documento XML, esta declaración afectará al resto de elementos del documento, ya que todos son hijos del elemento raíz. En el caso de las declaraciones de espacios de nombre para los atributos el procedimiento es análogo:

```
<Coches xmlns:modelo="http://www.espacioNombre1.com/">
  <unidad modelo:atrib="Valor_atributo"/>
  ...
</Coches>
```

En este caso, el elemento hijo **unidad**, que está dentro del alcance de **Coches**, tiene un atributo llamado **atrib** que está definido en el espacio de nombre del URI. Dicho atributo queda unido al prefijo de los espacios de nombre mediante un nombre calificado (es un conjunto formado por un prefijo de espacio de nombres, seguido de dos puntos y un nombre local). Estos nombres calificados se pueden aplicar a atributos o a elementos.

Cuando describimos un problema o de algún modo lo formalizamos, es frecuente usar conceptos que se aplican a problemas diferentes. Centrando el tema en la sintaxis de XML, hay veces que queremos utilizar una misma etiqueta para diferentes entidades o elementos. Los espacios de nombres surgen con el objetivo de eliminar la posibilidad de ambigüedad a la hora de describir un problema. Por eso, en muchas ocasiones, para distinguir posibles etiquetas iguales con significados diferentes, podemos definir varios espacios de nombres de la siguiente manera:

```
<ejemplo xmlns:pasarela="http://www.fi.upm.es/PASARELA/"
          xmlns:coches="http://www.fi.upm.es/COCHES/"
          xmlns:coleccion="http://www.fi.upm.es/COLECCION/">
  <pasarela:MODELO>Karolina Kurkova</pasarela:MODELO>
  <coches:MODELO>BMW 530d</coches:MODELO>
  <coleccion:MODELO>Americana de 3 botones</coleccion:MODELO>
</ejemplo>
```

En el ejemplo vemos como se han declarado tres espacios de nombres distintos (**pasarela**, **coches** y **coleccion**). Posteriormente se ha creado distinta información con la etiqueta **MODELO** que en cada caso tenía un significado diferente. Gracias a los espacios de nombres hemos evitado la ambigüedad del documento XML. Además este ejemplo nos sirve para entender bien el concepto de **nombre calificado** que mencionábamos anteriormente (en el ejemplo son nombres calificados 'pasarela:MODELO', 'coches:MODELO' y 'coleccion:MODELO').

### 3.2. Espacio de nombres.

Un documento XML no requiere estar vinculado estrictamente a un espacio de nombres. Siempre y cuando el documento XML tenga una DTD que lo valide, no será imprescindible su uso. También es posible declarar un espacio de nombres sin especificar algún prefijo. En este caso, se considera que los elementos declarados quedan vinculados a ese espacio de nombres definido.

```
<Coches xmlns="http://www.coches.com/namespace">
  <Tipo>Se trata de una berlina mediana</Tipo>
</Coches>
```

En este caso, los elementos hijos como **Tipo**, quedarán vinculados al espacio de nombres de forma predeterminada.

En el caso de los atributos, el comportamiento es distinto a lo que sucede con los elementos, ya que éstos no se vinculan al espacio de nombres predeterminado. Para ilustrar esta situación mostraremos un ejemplo:

```
<Coches xmlns="http://www.coches.com/namespace">
  xmlns:espacio1="http://www.coches.com/namespace1">

  <Tipo atributo1="valor"
    espacio1:atributo2="valor2">Información
  </Tipo>
</Coches>
```

En este ejemplo vemos como se han asignado dos espacios de nombre al elemento Coches apuntando cada uno de ellos a un URI diferente. Según lo comentado anteriormente, el primer atributo (**atributo1**) del elemento **Tipo** no se asignará a ningún espacio de nombres, mientras que el segundo atributo (**atributo2**) del elemento **Tipo** es asignado por defecto al espacio de nombres **espacio1** debido al uso del prefijo en la declaración del espacio de nombres.

## 4. Sintaxis de los esquemas W3C XML.

El concepto originario de esquema en XML fue el de un documento que describía una serie de restricciones y de convecciones en la estructura de una base de datos. Desde el punto de vista de XML, esa definición se ajusta en parte al propósito de dichos esquemas. Son documentos que describen el contenido que podemos usar en otros documentos. En ese sentido, su utilidad es más parecida a la de las DTDs, aunque restringen la estructura del documento de forma mucho más precisa. Los esquemas indican tipos de dato, número mínimo y máximo de ocurrencias y otras características más específicas. Tras varios intentos infructuosos a la hora de establecer un lenguaje estándar de desarrollo de esquemas, los esquemas W3C se han erigido como un estándar de creación de esquemas en XML.

Este lenguaje es muy extenso. La especificación del mismo está considerada más compleja que incluso la especificación 1.0 de XML. Para propósitos más modestos, existen otra serie de lenguajes más sencillos que pueden cumplir perfectamente su cometido.

Según la Especificación del W3C XML Schema (<http://www.w3.org/XML/Schema>), los esquemas expresan vocabularios compartidos que permiten a las máquinas

extraer las reglas hechas por las personas. Los esquemas proveen un significado para definir la estructura, contenido y semántica de los documentos XML. De algún modo, ofrecen nuevas posibilidades en el tratamiento de documentos. Podríamos señalar muchos puntos diferentes entre los esquemas y las DTD, pero nos centraremos en algunos puntos en los que hay ventajas al usar esquemas en lugar de DTDs.

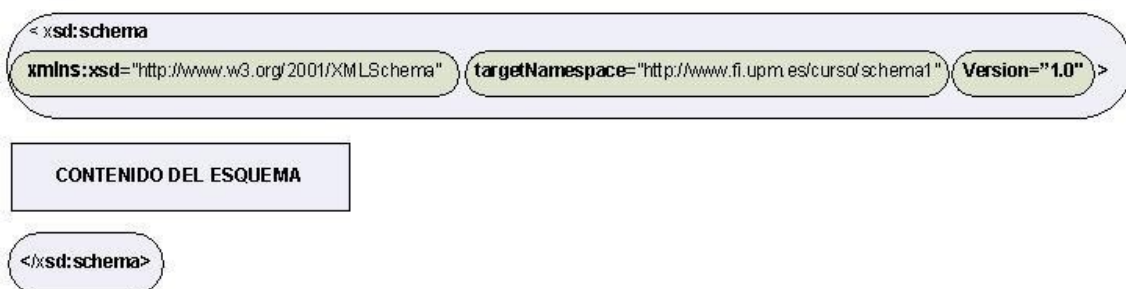
- Usan la **sintaxis propia de XML**, al contrario de lo que sucede con las DTDs.
- Permiten **especificar de manera precisa los tipos de datos**, mientras que las DTDs no los especifican.
- Son **extensibles**, es decir, podemos crear nuevos elementos.
- Para procesar el documento, las herramientas y analizadores empleados para tratar los documentos XML deben ser capaces de procesar también las DTDs.
- Con los esquemas W3C **es posible expresar carga semántica** y esto es muy importante para el tratamiento de la información.

Por otro lado, algunas desventajas claras de las DTDs son las siguientes:

- **No permite el uso de namespaces**, circunstancia que hace que sea más difícil y estos son muy útiles ya que permiten definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento.
- Tiene una **tipología para los datos del documento extremadamente limitada**, pues no permite definir el que un elemento pueda ser de un tipo número, fecha, etc. sino que sólo presenta variaciones limitadas sobre cadenas.
- El **mecanismo de extensión es complejo y frágil ya que está basado en sustituciones sobre cadenas** y no hace explícitas las relaciones, es decir, que dos elementos que tienen definido el mismo modelo de contenido no presentan ninguna relación.

Al igual que sucede con las bases de datos, en los esquemas XML hay parte de la información contenida que no es explícita, sino que es inherente a la estructura que se ha creado con dicho esquema. El gran logro de XML es la habilidad para modelar los datos partiendo desde distintas fuentes de datos. El manejo de los esquemas XML implica de algún modo, aprender como se escriben y modelan los datos básicos.

Podemos considerar un esquema como un contenedor de componentes que incluyen elementos, atributos, etc. Un ejemplo de la sintaxis de declaración del componente del esquema puede ser el que mostramos a continuación:





En este ejemplo se ha empleado el prefijo `xsd` para indicar que comienza el espacio de nombre del esquema XML. El espacio de nombre incluido funciona como designador del esquema que estamos construyendo. Hay que incluirlo porque el esquema que se está construyendo, depende del esquema XML del sitio Web W3C.

Las etiquetas `xs` y `xsd` son el prefijo de espacio de nombres XML más comúnmente usados, pero se puede utilizar cualquier prefijo declarándolo mediante los elementos `xmlns` en el elemento raíz. `Xs` es el usado por W3C y `xsd` se usa más en los esquemas de Microsoft, pero su uso es indiferente.

Lo importante es cómo se declara el espacio de nombres. Por ejemplo:

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

O

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

Ambos son válidos y equivalentes.

## 5. Tipos de datos.

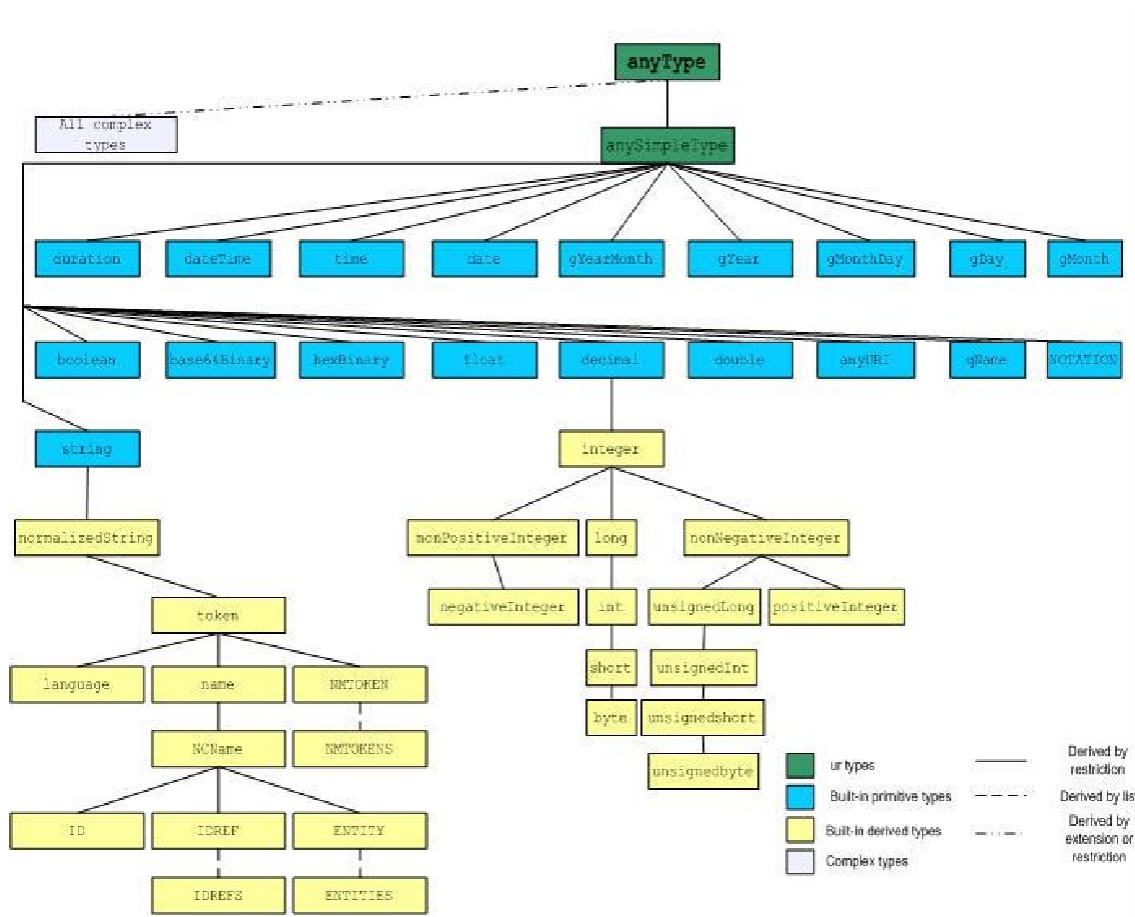
### 5.1. Introducción.

Los atributos y los elementos que declaramos en un esquema XML pertenecen a un tipo de datos concreto. Los mecanismos de XML para definir los tipos de datos en sus esquemas son bastante variados, pudiendo definir desde **tipos complejos** a los tipos más simples de datos. La **recomendación W3C** separa los tipos de datos en diversas categorías que veremos a continuación.

### 5.2. Categorías de los tipos de datos.

- **Tipos de datos atomizados.** Es la unidad más pequeña de tipo de datos que manejamos en XML. Este grupo de datos recibe el nombre de atómicos debido a que no se pueden subdividir en más categorías.
- **Tipos de listas.** Se trata de tipos que constituyen una lista de valores de tipo atomizados. Las cadenas enumeradas forman una restricción del tipo de datos de cadena.
- **Tipos de datos unión.** Funcionan de forma similar a las listas pero pueden contener elementos de diferentes tipos. A cada uno de esos elementos se le pueden aplicar distintas facetas (más adelante veremos este concepto en profundidad). Los tipos de datos incluidos se llaman tipos de miembros y el orden en que están declarados debe ser el orden en el que suceden esos valores.

La forma de concebir los distintos tipos de datos en los esquemas XML se puede resumir en la siguiente jerarquía:



En la figura vemos como anyType engloba todos los tipos de datos permitidos en los esquemas XML. De esa clase raíz derivan:

1. Los **tipos complejos** que incluyen contenido de texto y elementos. Estos derivan de anyType pro extensión (se puede aumentar el rango de valores respecto del tipo del que derivan) o restricción.
2. Los **tipos simples**. A su vez dentro de los tipos simples encontramos dos grandes grupos:
  - **Tipos primitivos o básicos** (En la figura en color azul)
  - **Tipos derivados**. Estos tipos a su vez se desglosan en:
    - Tipos derivados por **restricción**: Es decir, incluyen un subconjunto de los posibles valores del tipo del que derivan.
    - Tipos derivados por **lista**: Se utilizan porque constituyen listas del tipo del que derivan.

Comentaremos a continuación algunos tipos de datos que se emplean con frecuencia:

- **xsd:integer:** Un elemento de este tipo podrá ser cualquier entero positivo, incluidos el cero y los enteros negativos. A continuación mostraremos un ejemplo de uso:

```
<xsd:element name="Potencia" type="xsd:integer"/>

  <Potencia>
    100
  </Potencia>
```

Del mismo modo se utilizarían los tipos `positiveInteger` (rango restringido a los números positivos)

- **xsd:float:** Un elemento de este tipo podrá ser cualquier número en coma flotante de 32 bits (se admiten representaciones desde 12,45 a 5,4e+10). También se admite la representación del infinito positivo (INF), infinito negativo (-INF), y 'no es número' (NaN). Veamos un ejemplo a continuación:

```
<xsd:element name="Par_Motor" type="xsd:float"/>

  <Par_Motor>
    100,34
  </Par_Motor>
```

- **xsd:double:** Un elemento de este tipo podrá ser cualquier número de coma flotante con 64 bits.

```
<xsd:element name=" Resistencia" type="xsd:double"/>

  <Resistencia>
    100,98e+3
  </Resistencia>
```

- **xsd:date:** Un elemento de este tipo podrá contener cualquier fecha en el siguiente formato: AAAA-MM-DD. A continuación mostraremos un ejemplo:

```
<xsd:element name="FechaRevision" type="xsd:date"/>

  <FechaRevision>
    2006-12-18
  </FechaRevision>
```

Del mismo modo se utilizarían los tipos **month** (especifica un mes y año), **year** (especifica un año) y **century** (especifica un siglo).

- **xsd:time**: Un elemento de este tipo podrá contener cualquier hora expresada en los siguientes formatos:
  - hh:mm:ss.ss
  - hh:mm:ss.ss + hh.mm (indican diferencias respecto a la GMT)
  - hh:mm:ss.ss - hh.mm (indican diferencias respecto a la GMT)

Añadir una Z para indicar que la zona horaria se ajusta a la hora media de Greenwich o a la hora universal coordinada.

A continuación mostraremos un ejemplo:

```
<xsd:element name="registro" type="xsd:time"/>

<registro>
    03:22:01
</registro>
```

- **xsd:boolean**: Un elemento de este tipo podrá contener un valor lógico 'verdadero' (true) o 'falso' (false). También es posible usar el 0 o el 1. A continuación mostraremos un ejemplo:

```
<xsd:element name="abierto" type="xsd:boolean"/>

<abierto>
    true
</abierto>
```

**xsd:language**: Un elemento de este tipo podrá especificar un idioma con el formato de dos caracteres que establece la norma ISO630. A continuación mostraremos un ejemplo:

```
<xsd:element name="diccionario" type="xsd:language"/>

<diccionario>
    ES
</diccionario>
```

Con este ejemplo, se ha especificado que el lenguaje empleado es español (código ES).

- **xsd:uri-reference:** Un elemento de este tipo podrá especificar una URL (localizador universal de recursos). Nótese que únicamente está permitido introducir una URL, quedando por el momento restringido el uso de URI (a pesar del nombre del tipo de datos). A continuación mostraremos un ejemplo:

```
<xsd:element name="Favoritos" type="xsd:uri-reference"/>

<Favoritos>
    http://www.fi.upm.es
</Favoritos>
```

- **xsd:NMTOKEN:** Un elemento de este tipo especifica que el texto que se declara posteriormente entre las etiquetas sea un nombre XML válido. A continuación mostraremos un ejemplo:

```
<xsd:element name="modelo" type="xsd:NMTOKEN"/>

<modelo>
    Porsche
</modelo>
```

Hasta aquí hemos visto algunos de los tipos básicos más empleados con algunos ejemplos de uso.

### 5.3. Componentes del tipo de datos.

Antes hemos comentado que en algunos tipos de datos como por ejemplo sucede con los tipos booleanos, es posible utilizar el valor *'true'* o *'1'* y *'false'* o *'0'*. Esto lo conocemos con el nombre de representaciones léxicas. Además de esta posibilidad, existe lo que conocemos con el nombre de facetas de limitación de tipos o restricciones de tipos. Éstas se pueden definir como mecanismos que permiten a un desarrollador de código XML, modificar el contenido de los valores que puede tomar un tipo de datos. Mediante restricciones podemos indicar que un valor debe estar comprendido en un rango concreto, debe ser un valor de una lista de valores "cerrada", o debe ser mayor o menor que otro valor.

Los tipos de datos de XML pueden tener dos tipos de facetas:

- **Fundamental:** Van ligadas de forma natural al tipo de datos y no se pueden evitar. Entre las más importantes están las siguientes:
  - **Equal:** Denota igualdad entre valores de un tipo de datos.
  - **Ordered:** Relaciones de orden entre valores de un tipo de datos.
  - **Bounded:** Límites superiores e inferiores para valores.

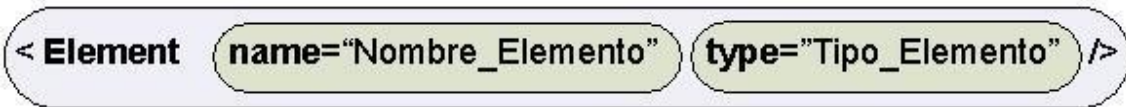
- **Cardinality:** Define si un valor es finito o infinito.
- **Numeric:** Define si un valor es numérico o no.
- **Restictiva:** Se pueden aplicar u omitir. En el caso de que se omitan no limitan al tipo de datos. En el caso de que las apliquemos, el tipo de datos quedará restringido. Existen los siguientes tipos:
  - **Length:** Contemplamos tres casos sobre los que aplicar el concepto de longitud. En el caso de una cadena, la longitud es el número de caracteres. Si nos referimos a una lista, la longitud es el número de elementos de la lista. Si nos referimos a valores binarios, es el número de octetos en una secuencia de bits.
  - **Enumeration:** Indica un conjunto de valores permitidos. Como en el caso de las enumeraciones de una DTD no tienen relación de orden prefijada.
  - **whiteSpace:** Se encarga del tratamiento de los espacios en blanco y de los tabuladores. Ofrece tres configuraciones distintas. La sintaxis que debemos emplear para este tipo de faceta es la siguientes:
    - **Preserve:** No afecta a ningún espacio en blanco, salto de línea o tabulador.
    - **Replace:** Reemplaza los tabuladores, avances de línea y retornos de carro con espacios en blanco.
    - **Collapse:** Reemplaza en el texto todos aquellos espacios en blanco contiguos a otro espacio en blanco. De este modo, podemos eliminar los espacios en blanco dobles.
  - **maxInclusive, minInclusive, maxExclusive y minInclusive:** Limitan los valores máximos y mínimos de los valores que puede contener un tipo de datos.
  - **Precision:** Especifica la cantidad de datos permitidos para números que son del tipo decimal.
  - **Scale:** Especifica la cantidad de dígitos que contiene la parte fraccionaria de un número decimal.
  - **Encoding:** Expresa la el tipo de codificación para un flujo de datos binario. Puede ser hexadecimal (hex) o en base 64 (Base64).
  - **Duration/Period:** Se refieren a la duración de una cantidad de tiempo o al periodo transcurrido entre duraciones.

## 6. Declaración de elementos de y atributos.

### 6.1. Declarar elementos en un esquema XML.

Al igual que sucede con las ya comentadas DTDs, en los esquemas XML hay que declarar los elementos que más tarde emplearemos en el documento XML.

La sintaxis que se debe emplear para declarar un elemento es la siguiente:



```
< Element name="Nombre_Elemento" type="Tipo_Elemento" />
```

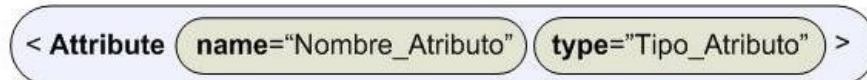
Como vemos, en la declaración del elemento, este queda definido por un nombre de elemento y por un tipo. Más adelante veremos los diferentes tipos de datos que soporta un elemento en los esquemas XML.

### 6.2. Declarar atributos en un esquema XML.

Ya se ha visto anteriormente la importancia y flexibilidad que proporcionan los atributos en XML. Como no podía ser de otra forma, los esquemas XML tienen un mecanismo similar al empleado con los elementos para declarar atributos. Al margen de indicar el nombre del atributo y el tipo del mismo, los atributos tienen a su vez un atributo en su declaración que nos permite definir la cardinalidad de dicho atributo.

Este atributo define el número máximo o mínimo de ocurrencias que debe tener ese atributo declarado en un elemento. Además, también existen dos atributos que ya habíamos visto en la declaración de un atributo dentro de un documento XML y cuyo funcionamiento es el mismo. Éstos son **default** y **fixed**. Su objetivo es usar el valor predeterminado y fijar un valor respectivamente.

La sintaxis que se debe emplear para declarar un atributo es la siguiente:



```
< Attribute name="Nombre_Atributo" type="Tipo_Atributo" >
```

La declaración de este atributo, queda definida por un nombre de atributo y por un tipo de atributo. Con el atributo **“use”** indicamos si la aparición del atributo en el elemento con el que se ha asociado es opcional (si le damos el valor **“optional”**, lo cual es el comportamiento por defecto), obligatorio (si le damos el valor **“required”**) o si está prohibido su uso (si le damos el valor **“prohibited”**).

## 7. Creando tipos simples de datos.

### 7.1. Introducción.

Como en otros lenguajes de programación, en XML es posible crear nuestros propios tipos de datos. Bastará con tener claro cuáles son los parámetros que definen el tipo que queremos obtener para crear así nuestro tipo de datos a la medida. Para la creación de un tipo simple de datos, se utiliza el elemento **simpleType**.



Con las DTDs vimos como podíamos hacer algo parecido a crear un tipo de datos. A continuación veremos como es posible hacerlo con los esquemas XML. Para ello introduciremos un ejemplo y comentaremos el código paulatinamente:

```
1 <?xml version="1.0" ?>
2 <xsd:Schema xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
3   <xsd:element name="coche">
4     <xsd:simpleType>
5       <xsd:restriction base="xsd:string">
6         <xsd:enumeration value="audi"/>
7         <xsd:enumeration value="volvo"/>
8         <xsd:enumeration value="mercedes"/>
9       </xsd:restriction>
10    </xsd:simpleType>
11  </xsd:element>
12 </xsd:Schema>
```

Con este esquema hemos creado un elemento coche. Como podemos comprobar en la primera línea, no hemos declarado ningún tipo. Posteriormente, se abre un elemento **simpleType** en el que englobamos la definición del tipo, que como vemos, requiere el uso de varias líneas.

El tipo debe estar basado en uno de los tipos predefinidos que tienen los esquemas XML. En este caso, el tipo está creado sobre una restricción del tipo **string** (indicado en la línea 5). En las líneas 6, 7 y 8 indicamos los valores que puede adoptar nuestro tipo. En este caso, usamos **xsd:enumeration** para indicar cuales son los valores del tipo coche que serán admisibles.

Debemos tener en cuenta que posteriormente, cuando en el documento XML queramos utilizar dicho tipo para un elemento, este podrá tomar únicamente un valor (de los tres declarados).

A continuación mostraremos un nuevo ejemplo para clarificar la declaración de tipos simples en los esquemas XML.

En esta ocasión, nuestro tipo 'PrecioVivienda' se basa en el tipo predefinido **float**. En la declaración hemos establecido un límite máximo y un límite mínimo de valores que podrá adoptar dicho tipo. Cualquier valor situado entre el rango especificado, será válido para nuestro tipo.

```
1 <?xml version="1.0" ?>
2 <xsd:Schema xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
3   <xsd:element name="Preciovivienda">
4     <xsd:simpleType>
5       <xsd:restriction base="xsd:float">
6         <xsd:minInclusive value="150.000,679"/>
7         <xsd:maxInclusive value="300.000,356"/>
8       </xsd:restriction>
9     </xsd:simpleType>
10  </xsd:element>
11 </xsd:Schema>
```



## 7.2. Crear tipos de listas.

En las declaraciones de tipos vistas hasta el momento siempre hemos visto que cada elemento constaba de un único ítem (fuera este un entero, un float o un string). Los esquemas XML están provistos de un mecanismo para la creación de listas. Para ello se emplea `xsd:list`. Veamos un ejemplo:

```
1 <?xml version="1.0" ?>
2 <xsd:Schema xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
3   <xsd:element name="Viviendas">
4     <xsd:simpleType>
5       <xsd:list itemType="xsd:string"/>
6     </xsd:simpleType>
7   </xsd:element>
8 </xsd:Schema>
```

Veamos como interpretar dicho ejemplo:

- **Líneas 1 a 3:** Declaración de documento XML, del esquema XML y del nombre del tipo (viviendas).
- **Línea 4:** Apertura de la etiqueta que engloba la definición del tipo.
- **Línea 5:** En esta línea se declara que el tipo viviendas va a ser una lista (**`xsd:list`**) en la que cada ítem o elemento de la lista va a ser de tipo **string** (**`itemType=xsd:string`**).
- **Líneas 6 a 8:** Cierre de las etiquetas de definición de tipo, elemento y esquema.

A continuación veremos posibles ejemplos que se ajustan al tipo vivienda que hemos declarado:

```
<Viviendas>
    Apartamento Piso Dúplex Adosado Pareado Independiente
</Viviendas>
```

Empleando las **facetas** que hemos comentado anteriormente, podemos acotar la longitud de la lista de diversas formas:

```
1   <xsd:simpleType>
2     <xsd:list itemType="xsd:string"/>
3     <xsd:minLength value="5"/>
4     <xsd:maxLength value="7"/>
5     <xsd:length value="10"/>
6   </xsd:simpleType>
```

En la **línea 3** de este ejemplo podemos ver como limitar la longitud mínima de la lista a cinco elementos o como limitar la longitud máxima de elementos de la lista a siete elementos (**línea 4**). Por otro lado, también es posible especificar la longitud de la lista de manera precisa como hemos hecho en la **línea 5**.

### 7.3. Combinar tipos simples de datos

En algunas situaciones podemos buscar un tipo que combine dos tipos distintos. Con XML es posible utilizar estos tipos conocidos con el nombre de tipos union (xsd:union). La sintaxis que puede parecer un poco más farragosa consiste en englobar dentro de un elemento xsd:union los distintos tipos simples que queremos usar. A continuación mostraremos un ejemplo:

```

1 <?xml version="1.0" ?>
2 <xsd:Schema xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema">
3 <xsd:element name="velocidad"
4 <xsd:simpleType>
5   <xsd:union>
6     <xsd:simpleType>
7       <xsd:restriction base="xsd:integer"/>
8       <xsd:enumeration value="180"/>
9       <xsd:enumeration value="51"/>
10    </xsd:simpleType>
11    <xsd:simpleType>
12      <xsd:restriction base="xsd:string"/>
13      <xsd:enumeration value="Excesiva"/>
14      <xsd:enumeration value="Alta"/>
15      <xsd:enumeration value="Media"/>
16      <xsd:enumeration value="Baja"/>
17    </xsd:simpleType>
18  </xsd:union>
19 </xsd:simpleType>
20 </xsd:element>
21 </xsd:Schema>

```

En este ejemplo, hemos combinado dentro de un mismo tipo 'velocidad' valores de tipo **integer** (declaración entre las **líneas 7 y 9**) y de tipo **string** (declaración entre las **líneas 12 y 16**). A la hora de emplear este código en un documento XML, podemos tener el siguiente ejemplo:

```

<velocidad>
    Excesiva
</velocidad>
<velocidad>
    180
</velocidad>

```

También podemos usar el tipo nuevo creado para la declaración de otros elementos.

```
<xsd:element name="prestacionesModelo1" type="velocidad"/>
<xsd:element name="prestacionesModelo2" type="velocidad"/>
```

### 7.4. Fijar el valor de un tipo.

Además de los diferentes tipos que hemos visto hasta ahora, XML permite fijar el contenido de un tipo de forma estricta, evitando que el elemento tome distintos valores. Para ello se emplean los modificadores o atributos **'fixed'** y **'default'**. Con **fixed** fijamos el valor de un tipo, mientras que con **default**, indicamos un valor por defecto para el tipo, de forma que el parser de XML añadirá directamente ese valor cuando sea omitido. El siguiente fragmento de código mostrará el concepto de forma más clara:

```
<xsd:element name="modelo" type="xsd:string" fixed="Renault"/>
<xsd:element name="modelo" type="xsd:string" default="Audi"/>
```

## 8. Crear tipos complejos de datos.

La diferencia básica que existe en los esquemas XML entre un tipo simple y un tipo complejo es la posibilidad de contener elementos y atributos en un mismo tipo. Para declarar un tipo complejo de datos en XML, utilizamos el elemento **complexType**. La declaración de un tipo complejo se suele dividir en tres zonas:

- declaraciones de elementos,
- referencias a esos elementos y
- declaraciones de atributos.

Los elementos, como sucede en los tipos simples, son declarados utilizando el elemento **element**. Los atributos se declaran utilizando el elemento **attribute**. Veamos a continuación la declaración de un tipo complejo para comentar ciertos aspectos de su definición:

```
1 <xsd:element name="modeloVehiculo">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element name="tipo" type="xsd:string"/>
5       <xsd:element name="precio" type="xsd:integer"/>
6       <xsd:element name="marca" type="xsd:string"/>
7     </xsd:sequence>
8     <xsd:attribute name="garantia" type="xsd:integer" fixed="2"/>
9   </xsd:complexType>
10 </xsd:element>
```

En la **línea 2** se realiza la declaración del tipo. Como vemos, lleva el elemento **complexType** que indica que se trata de un tipo complejo. Posteriormente, se introduce el nombre del tipo (**modeloVehiculo**).

Posteriormente en la **línea 3**, se abre el elemento **sequence** para indicar que el tipo complejo consta de una sucesión de elementos. Es importante reseñar que la secuencia de elementos debe respetar el orden introducido en la declaración. En la **líneas 4, 5 y 6** se realiza la declaración de los elementos que conforman el tipo complejo. Se declara un elemento **tipo** de tipo **string**, un elemento **precio** de tipo **entero** y un elemento **marca** de tipo **string**.

En la **línea 7** se cierra el elemento **sequence** para indicar que ha finalizado la secuencia de elementos que constituyen el tipo. En la **línea 8** se declara un atributo llamado **garantía** que es de tipo **integer** y cuyo valor ha sido fijado al número entero 2 con la faceta **fixed**.

Con este ejemplo, hemos declarado un tipo complejo que funcionará del siguiente modo; cada vez que se declare un elemento de este tipo, este deberá contener los tres elementos que integran el tipo (tipo, precio y marca), así como un atributo (garantía). Además, el orden será el especificado en la declaración del tipo.

A continuación, mostraremos un nuevo ejemplo en el que declararemos un nuevo tipo complejo, que use el anterior tipo declarado.

```

1  <xsd:element name="Garaje">
2    <xsd:complexType>
3      <xsd:sequence>
4        <xsd:element name="coche1" type="modeloVehiculo"/>
5        <xsd:element name="coche2" type="modeloVehiculo"/>
6        <xsd:element name="incidencia" type="xsd:string"/>
7      </xsd:sequence>
8      <xsd:attribute name="hora" type="xsd:time"/>
9    </xsd:complexType>
10 </xsd:element>

```

Con esta nueva declaración de tipo, cada vez que en un documento XML declaremos un elemento con el tipo **Garaje**, este tendrá tres elementos, de los cuales dos de ellos (**coche1** y **coche2**) tendrán a su vez tres subelementos o elementos hijos (declarados a su vez en el tipo **modeloVehiculo**) y un atributo (**garantia**). Además, el tipo **Garaje**, deberá incluir nuevamente un atributo cuyo tipo será **time**. Con este ejemplo, se puede comprobar como es posible combinar en un tipo complejo, distintos tipos (desde strings a enteros, pasando por time).

## 9. Modelos de contenidos (cardinalidad, orden y agrupación).

### 9.1. Atributos **MinOccurs** y **MaxOccurs**.

Al explicar las definiciones de tipo de documento (DTDs) en las lecciones previas, hemos visto los mecanismos que seguían las mismas para especificar las diferentes cardinalidades que podía tener un elemento. Recordamos que se usaban como modificadores de cardinalidad los símbolos \*, + e ?.

Pues bien, los esquemas XML están provistos de mecanismos similares para controlar la cardinalidad de los elementos que incluimos en las distintas declaraciones. A continuación veremos cuáles son:

- **Atributo minOccurs.** Indica el mínimo número de ocurrencias que se deben dar en la declaración del tipo. Por ejemplo, si este atributo toma el valor '1' indica que el elemento debe aparecer exactamente una vez. El atributo no ofrece ninguna restricción en el valor del elemento, por lo que éste podrá tomar cualquier valor. El valor por defecto en caso de que no se indique ningún otro es '1'.
- **Atributo maxOccurs.** Indica el máximo número de ocurrencias que se deben dar en la declaración del tipo. El valor que toma por defecto es, como ocurría con minOccurs, de 1 elemento. En este tipo de atributo, podemos usar como valor del mismo, la palabra 'unbounded' (sin límite) que nos indica que no existe un número máximo de ocurrencias.

Ahora podemos ver que con estas definiciones, sería equivalente usar las siguientes construcciones:

- **minOccurs='0'** y **maxOccurs='1'** es equivalente a usar **element?** en las DTDs.
- **minOccurs='1'** y **maxOccurs='1'** es equivalente a usar **element** en las DTDs.

Y usando el término '**unbounded**' podríamos sustituir el uso de **element\*** y **element+** respectivamente, con las siguientes construcciones:

- **minOccurs='0'** y **maxOccurs='unbounded'** para la declaración **element\***.
- **minOccurs='1'** y **maxOccurs='unbounded'** para la declaración **element+**.

Debemos tener en cuenta ahora que hemos visto estos atributos, que si un elemento es declarado sin el atributo **maxOccurs**, el elemento no puede aparecer más de una vez. Además, parece lógico que **XML no permita declarar un atributo minOccurs cuyo valor sea superior a al valor que tome el atributo maxOccurs**. Del mismo modo, **el atributo maxOccurs nunca podrá ser inferior al límite marcado en el atributo minOccurs**.

En el caso de los atributos, no necesitamos controlar la ocurrencia de los mismos, ya que estos aparecerán una vez o directamente no aparecerán. Éstos pueden ser declarados con un atributo **use** para indicar tres posibilidades:

- **Atributo required:** Indica que el uso del atributo es obligado.

- **Atributo optional:** Indica que el uso del atributo es opcional.
- **Atributo prohibited:** Indica que el uso del atributo está prohibido.

### 9.2. Elementos Choice y All.

Hasta ahora sólo hemos visto ejemplos de tipos complejos en cuya declaración utilizábamos el elemento sequence. Recordamos que su uso indicaba que los elementos de los que constaba el tipo deberían aparecer todos y respetando el orden que tenían en su declaración. Ahora veremos dos nuevos elementos que ofrecen los esquemas XML para restringir la estructura del tipo.

#### 9.2.1. Elemento Choice.

El indicador `xsd:choice` se utiliza cuando en la declaración de un tipo complejo, queremos indicar que sólo debemos escoger un elemento de los posibles elementos contenidos. A continuación mostraremos un ejemplo:

```
1 <xsd:element name="vivivenda">
2   <xsd:complexType>
3     <xsd:choice>
4       <xsd:element name="atico" type="xsd:string"/>
5       <xsd:element name="pareado" type="xsd:string"/>
6       <xsd:element name="adosado" type="xsd:string"/>
7       <xsd:element name="independiente" type="xsd:string"/>
8     <xsd:choice>
9   </xsd:complexType>
10 </xsd:element>
```

#### 9.2.2. Elemento All.

El indicador `xsd:all` indica que los elementos que contiene un tipo complejo, pueden aparecer en cualquier orden, pero como máximo sólo una vez. Es decir, su funcionamiento es similar al del elemento sequence, con la salvedad de que ahora los elementos no deben ceñirse al orden en el que están declarados en el tipo. Veamos un ejemplo del funcionamiento:

```
1 <xsd:element name="jardin">
2   <xsd:complexType>
3     <xsd:all>
4       <xsd:element name="olivo" type="xsd:string"/>
5       <xsd:element name="rosal" type="xsd:string"/>
6       <xsd:element name="arizonica" type="xsd:string"/>
7     <xsd:all>
8   </xsd:complexType>
9 </xsd:element>
```

Para este ejemplo de declaración de tipo, podríamos tener un código XML como el que vemos a continuación:

```
<jardin>
  <rosal>hay 50 unidades</rosal>
  <arizonica>hay 12 unidades</arizonica>
  <olivo>hay 10 unidades</olivo>
</jardin>
```



## 10. Polimorfismo.

La palabra polimorfismo se le aplica a algo que puede adoptar distintas formas. En los lenguajes de programación, sobre todo en los lenguajes orientados a objetos, se conoce como polimorfismo la capacidad que tienen dos objetos de diferentes clases de responder a un mismo mensaje. El concepto de polimorfismo es aplicable tanto a funciones como a tipos de datos. Así nacen los conceptos de funciones polimórficas y tipos polimórficos. Las primeras son aquellas funciones que pueden evaluarse o ser aplicadas a diferentes tipos de datos de forma indistinta; los tipos polimórficos, por su parte, son aquellos tipos de datos que contienen al menos un elemento cuyo tipo no está especificado.

Se puede clasificar el polimorfismo en dos grandes grupos:

- Polimorfismo **dinámico** (o polimorfismo **paramétrico**) es aquél en el que el código no incluye ningún tipo de especificación sobre el tipo de datos sobre el que se trabaja. Así, puede ser utilizado a todo tipo de datos compatible.
- Polimorfismo **estático** (o polimorfismo **ad hoc**) es aquél en el que los tipos a los que se aplica el polimorfismo deben ser explicitados y declarados uno por uno antes de poder ser utilizados. Después de esta breve explicación sobre el concepto de polimorfismo de un modo más general, veremos como podemos emplear tipos polimórficos en XML. Se trata de convertir un tipo simple en un conjunto de tipos polimórficos. Para ello necesitamos un tipo genérico que llamaremos T, que contiene una serie de alternativas T1...Tn (en el ejemplo que vemos será el tipo **Pago**, que contiene las alternativas de pago en metálico y con tarjeta).
  1. Construimos un tipo abstracto P que contiene elementos de T.
  2. Para cada T en T1..Tn, construimos un tipo extendido E de P (utilizando el modificador extensión base) que contiene los elementos de T.

Ahora mostraremos un ejemplo que aclare esta explicación:

```

<!--Declaración de un tipo complejo 'Pago' que contenga las
alternativas descritas en el elemento choice-->

<complexType name="Pago">
  <sequence>
    <element name="cantidad" type="int"/>
    <choice>
      <element name="metalico" type="string"/>
      <element name="tarjeta" type="tns:tarjeta"/>
    </choice>
  </sequence>
</complexType>

<!--Definición del tipo complejo tarjeta-->

<complexType name="tarjeta">
  <sequence>
    <element name="numeroTarjeta" type="string"/>
    <element name="caducidad" type="date"/>
  </sequence>
</complexType>

```

Ahora mostraremos el tipo declarado anteriormente de forma que sea polimórfico:

```
<!--Declaración de un tipo complejo polimórfico denominado Pago.-->
<!--Expresamos Pago como un tipo complejo con el único elemento
cantidad.-->

<complexType name="Pago">
  <sequence>
    <element name="cantidad" type="int"/>
  </sequence>
</complexType>

<!--Declaramos las dos alternativas de pago como una extensión del
tipo Pago, del que heredan el elemento cantidad. -->

<complexType name="PagoMetalico">
  <complexContent>
    <extension base="tns:Pago">
      <sequence>
        <element name="metalico" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="PagoTarjeta">
  <complexContent>
    <extension base="tns:Pago">
      <sequence>
        <element name="tarjeta" type="tns:tarjeta"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

A partir de ahora, la declaración de un tipo Pago, debemos hacerla de la siguiente manera:

```
<!--Uso de un tipo complejo polimórfico denominado Pago para un caso
de pago con tarjeta de crédito.-->

<Pago xsi:type="PagoTarjeta">
  <cantidad>6.000</cantidad>
  <tarjeta>
    <numeroTarjeta>10504589563214598</numeroTarjeta>
    <caducidad>15/10/2008</caducidad>
  </tarjeta>
</Pago>
```

## 11. Herramientas para la edición y validación de esquemas.

### 11.1. Introducción.

Hay diversas herramientas para editar y validar esquemas. Nosotros en este curso, comentaremos algunos detalles sobre la herramienta gratuita **XML Copy Editor** y las he-



herramientas con licencia **XML Spy** y **eXcelon Stylus**, todas ellas disponibles en entorno Windows y que abarcan prácticamente todos los niveles de la tecnología XML.

Detallemos a continuación las herramientas propuestas.

### 11.2. XML Copy Editor.

Es una herramienta libre bajo licencia GNU (General Public License). Está disponible en varios idiomas.

Aplicación muy sencilla que se encarga de editar el código fuente de los archivos XML, y dispone de opciones que nos facilitan la edición XML.

Su interfaz es muy intuitiva y se puede usar en Windows o en Linux. Entre las características que podemos destacar es que nos permite validación de DTD/XML Schema/RELAX NG, resalta la sintaxis, comprueba la ortografía y permite el plegado de fragmentos de código.

Otras de las funciones que nos permite es importar o exportar documentos de Word sin que haya pérdida de formato, como también una función para comprobar los estilos y el deletreo de la sintaxis de XML.

Su descarga está disponible en <http://xml-copy-editor.sourceforge.net/>.

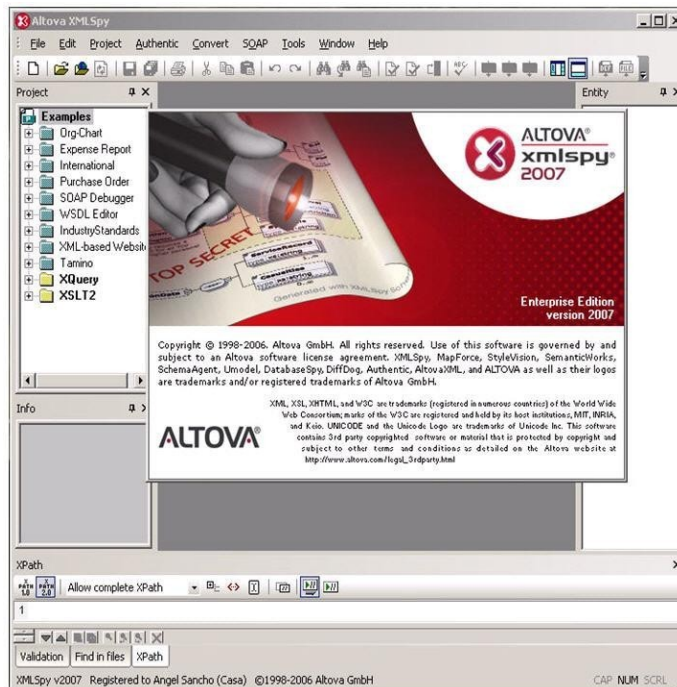
### 11.3. Altova XMLSpy 2007.

XML Spy 2007, es una aplicación que tiene la potencia necesaria para trabajar en cualquier proyecto relacionado con XML. La habilidad de desarrollar y administrar bases de datos, la combinación de soporte hacia los lenguajes de programación más avanzados de la actualidad, y las facilidades de uso, que igualmente requieren de un usuario con experiencia, son algunas de sus características.

Hay tres versiones de XML Spy 2007:

- la Home Edition,
- la Professional Edition, y finalmente
- la Enterprise Edition. (la que podemos ver en la imagen inferior).

#### 4. Creando documentos XML válidos. Esquemas W3C XML



Hay que reseñar que **no es una herramienta de libre distribución** y que el usuario debe contratar una licencia para usarla. Para este curso, hemos descargado desde la página <http://www.altova.com/> una versión de evaluación gratuita y totalmente funcional durante un mes.

El precio, y las características de cada versión varían sustancialmente, adaptándose a las necesidades y al bolsillo de cada usuario.

La versión Enterprise Edition posee la capacidad para realizar conversiones de archivos HTML a XML, utilidades para generar e incorporar código en Java y C++, y opciones especialmente destinadas al desarrollo Web. La Professional Edition no tiene todas las funciones de la Enterprise Edition, pero nos ofrece de todas formas, suficientes herramientas como para trabajar a fondo con documentos XML, bases de datos y aplicaciones Web.

Finalmente, la versión más pequeña y la más compacta es la Home Edition. Es una opción económica que posee herramientas básicas, pero bastante potentes. Esta versión 2007 incorpora un XSTL debugger que resultará de gran utilidad a los desarrolladores de hojas de estilo. Otra interesante función nos posibilita la conversión de documentos HTML a XML, y aunque pueden surgir algunos errores, resulta una herramienta de mucha ayuda.

Para ver algunos detalles más concretos sobre la herramienta, podemos ver un viewlet en el siguiente punto.

#### 11.4. EXcelon Stylus Studio 2007.

Stylus Studio es un completo entorno de desarrollo que integra la mayoría de las tecnologías XML en un único entorno. Incluye un potente editor de XML, un "debugger" XSLT y otras muchas herramientas pensadas especialmente para facilitar y mejorar la productividad en el desarrollo de sitios Web y aplicaciones.

#### 4. Creando documentos XML válidos. Esquemas W3C XML

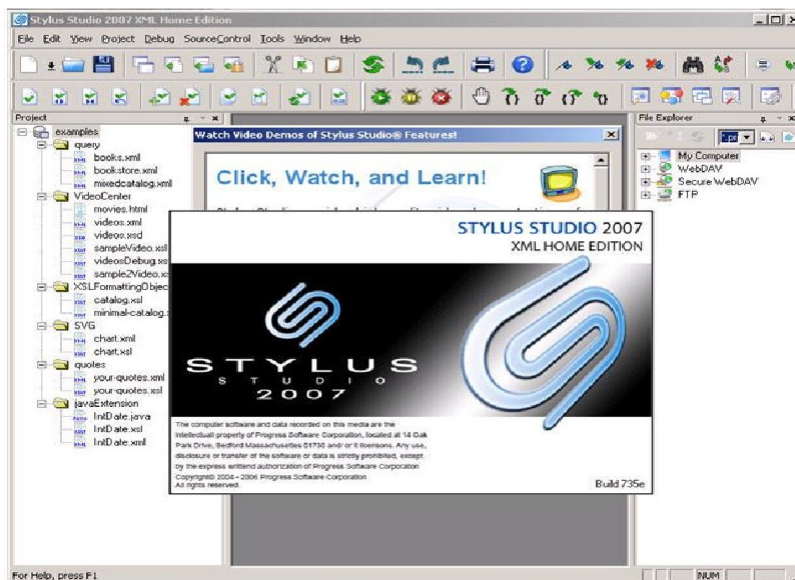
Permite la edición de XML en modo visual y sincronizado, y un completo conjunto de herramientas para desarrollo en XSLT entre las que se incluyen un debugger, un ma-peador y una utilidad de diseño de hojas de estilo de HTML a XSLT.

Esta última versión soporta una edición visual de XQuery (lenguaje de consultas sobre contenido XML), e incluye un editor DTD y utilidades para XPath.

Al igual que sucede con Altova XMLSpy, eXcelon Stylus Studio tampoco es una herramienta gratuita. Se puede adquirir en tres versiones:

- Stylus Studio 2007 XML Home Edition,
- Stylus Studio 2007 XML Professional Suite y
- Stylus Studio 2007 XML Enterprise Suite,

A continuación veremos una imagen del interfaz que presenta este entorno de desarrollo.



Los precios de las versiones citadas anteriormente, oscilan desde los 600\$ de la versión más cara (enterprise suite) hasta los 100 \$ de la versión home. Para el desarrollo de este curso, también ha sido posible descargar una versión de evaluación por un periodo de una semana desde la página Web: <http://www.stylusstudio.com/>.

También es posible utilizar en plataformas UNIX el editor **Emacs** que tiene un modo PSGML/XSL que permite crear y validar tanto código XML como esquemas.

## 12. Conclusión.

Con esta unidad didáctica ya conocemos las posibilidades que ofrece XML para la validación de su código. El alumno ha debido aprender cuál es la sintaxis y los pasos que debe dar un programador para crear su primer esquema de acuerdo a la recomendación W3C. Además el alumno debe haber comprendido los múltiples tipos de datos que ofrece el lenguaje de forma predefinida, además de las técnicas que debemos em-

plear para crear nuevos tipos de datos complejos. También es de esperar que se hayan fijado los conceptos de cardinalidad y polimorfismo que han sido explicados a través de dos lecciones de esta unidad didáctica.

### 13. Ejercicio resuelto.

Para que el alumno se familiarice con el interfaz de las dos aplicaciones comentadas, mostraremos a continuación un documento XML breve sobre la interfaz de la aplicación Stylus Studio 2007 XML. El documento XML contiene una breve declaración de un dominio relacionado con bibliotecas. A continuación mostraremos el código de dicho documento XML:

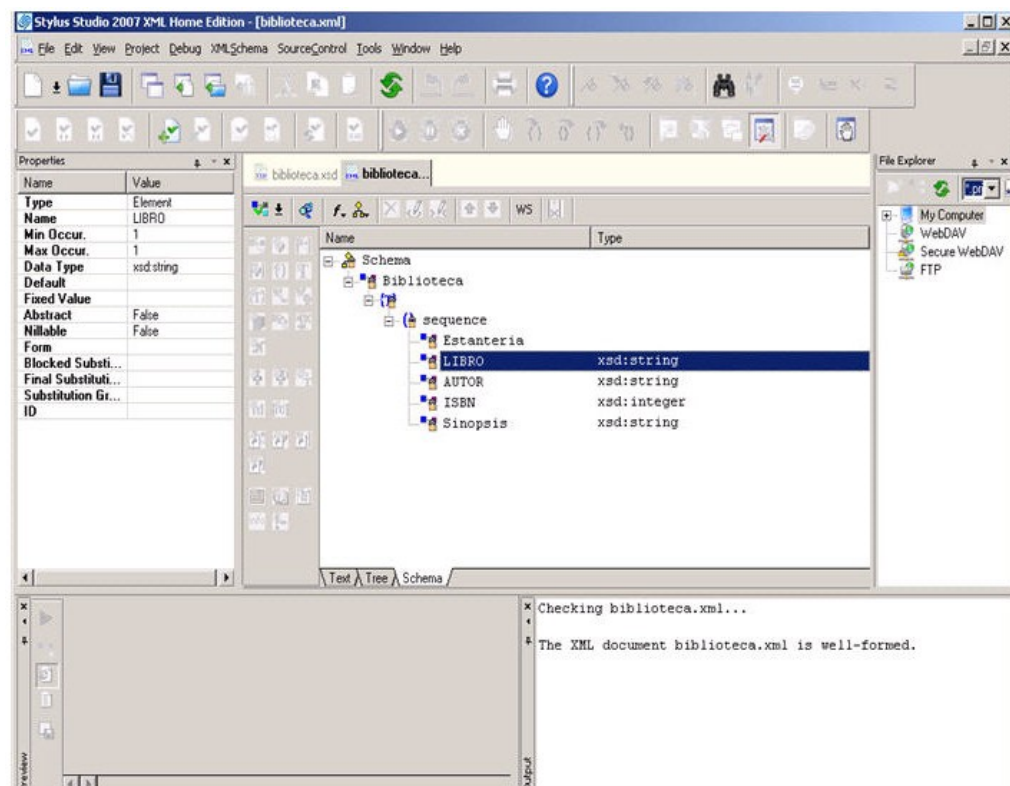
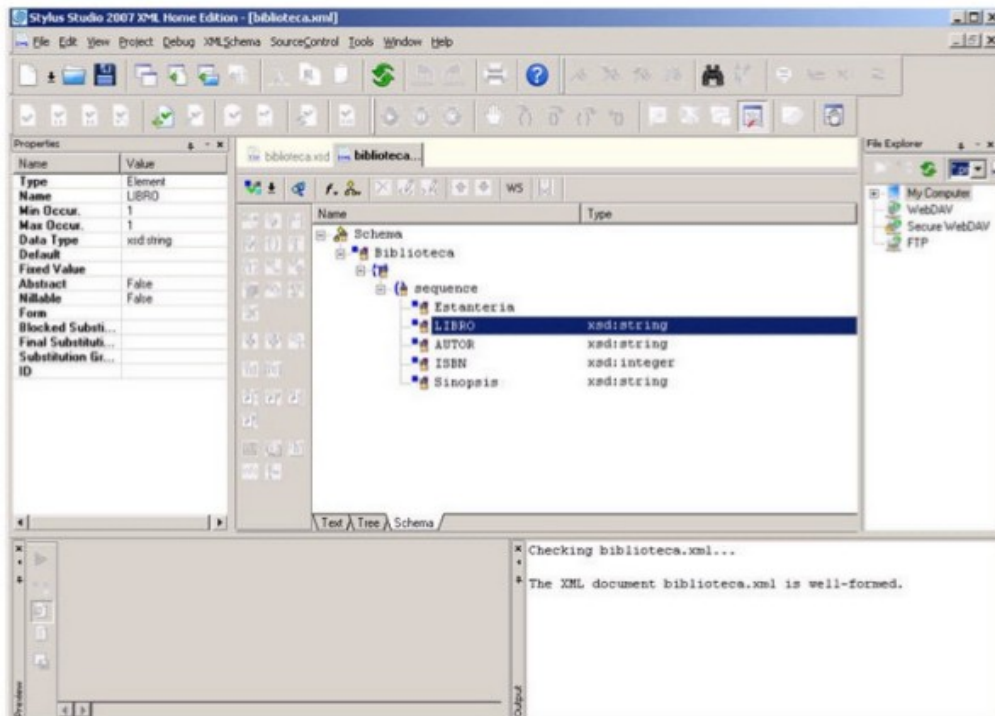
```
<?xml version="1.0" ?>
<Biblioteca
  xmlns="http://www.miPaginaDePruebaXML.com"
  xmlns:xsi="http://www.w3.org//2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.miPaginaDePruebaXML.com biblioteca.xsd">
  <Estanteria>H34</Estanteria>
  <Libro>Angeles y Demonios</Libro>
  <Autor>Dam Brown</Autor>
  <ISBN>5484985</ISBN>
  <Sinopsis>Cuando el reconocido profesor de simbología de Harvard Robert Langdon
acude a un laboratorio de investigación en Suiza para analizar un misterioso
símbolo, grabado a fuego en el pecho de un físico al que han asesinado,
descubre las pruebas de lo inimaginable: el resurgimiento de una antigua
hermandad secreta</Sinopsis>
</Biblioteca>
```

Realizar a continuación un esquema XML que valide el documento que se muestra :

#### 13.1. Posible solución.

Como se ha comentado en el enunciado de este ejercicio, aprovecharemos su resolución para mostrar algunas de las opciones que permite la aplicación con la cual hemos desarrollado el ejercicio. Para comenzar, mostraremos la estructura en forma de árbol que sigue el documento XML (llamado biblioteca.xml). En este tipo de vista, la pantalla se divide en dos partes; una en la que lista las etiquetas de forma jerarquizada y otra parte en la que indica el valor que tienen dichas etiquetas. La imagen es la siguiente:

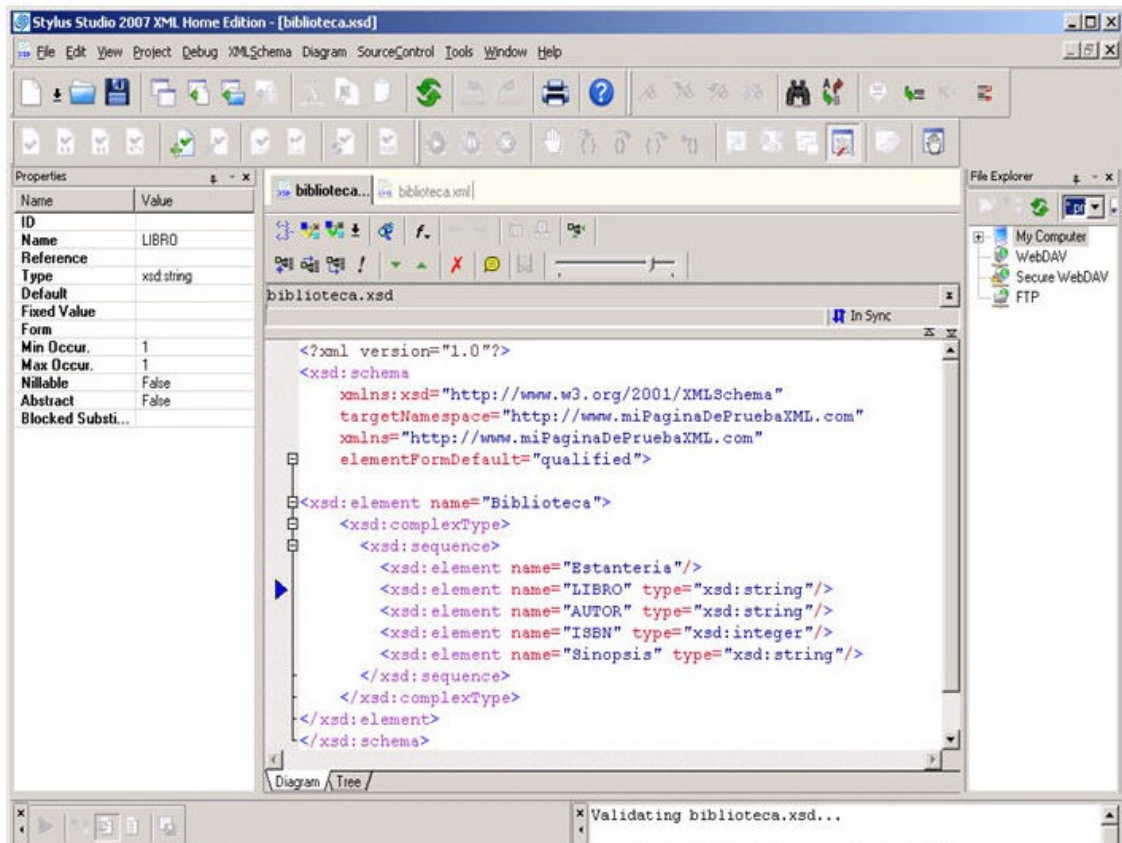
#### 4. Creando documentos XML válidos. Esquemas W3C XML



Esta segunda imagen, nos muestra la vista de esquema, en la que aparecen los distintos elementos de forma jerárquica y en la parte izquierda podemos ver los distintos tipos de atributos que tiene cada uno de ellos (máximo número de ocurrencias, mínimo número de ocurrencias, valor por defectos, etc.). Ahora veremos las capturas de la solución propuesta:



#### 4. Creando documentos XML válidos. Esquemas W3C XML



### 13.2. Comentarios.

Como vemos, la declaración del esquema comienza con la etiqueta `xsd:schema` en la que incluimos el espacio de nombres del esquema. Con la tercera línea, definimos este documento como un esquema XML.

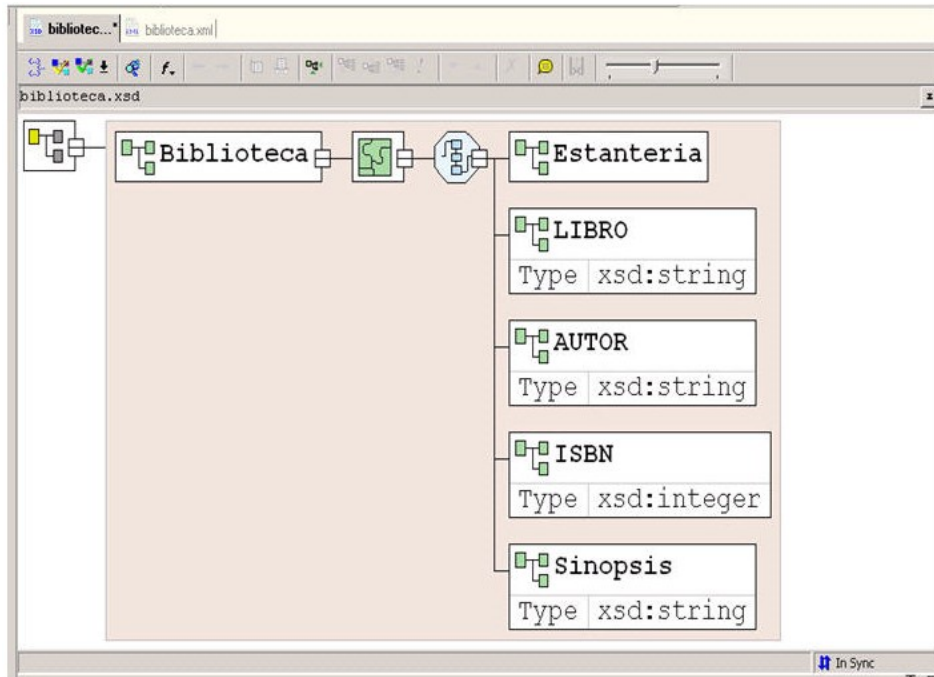
Además se incluye el **targetNamespace** que determina que nombre de espacio de nombre (un URI) puede usar el código XML. Para validar el **targetNamespace**, debemos declarar un espacio de nombres en el documento de esquema XML usando el mismo URI que el que hemos empleado en la definición de **targetNamespace** (línea 4).

Posteriormente se crea el elemento Biblioteca que comienza con la declaración de un tipo complejo (etiqueta `complexType`) que define una secuencia de elementos. Estos elementos deben ser los que hemos declarado previamente en el documento XML. Es decir, Estantería, LIBRO, AUTOR, ISBN y Sinopsis. Para cada uno de ellos se declara el tipo de datos admitido (Strings en todos ellos, salvo en ISBN que es un entero).

Finalmente se cierran todas las etiquetas que se habían abierto (inicio de tipo complejo, inicio de secuencia e inicio de elemento Biblioteca). Con esto queda finalizada el esquema para el documento biblioteca.xml (llamado biblioteca.xsd).

Para terminar el ejercicio, mostraremos dos vistas que genera la aplicación Stylus Studio. La primera de ellas, es la vista gráfica del esquema XML:

#### 4. Creando documentos XML válidos. Esquemas W3C XML



La segunda de ellas es la vista que proporcionaría un navegador Web:

```
<?xml version="1.0" ?>
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.miPaginaDePruebaXML.com"
  xmlns="http://www.miPaginaDePruebaXML.com"
  elementFormDefault="qualified">
- <xsd:element name="Biblioteca">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element name="Estanteria" />
  <xsd:element name="LIBRO" type="xsd:string" />
  <xsd:element name="AUTOR" type="xsd:string" />
  <xsd:element name="ISBN" type="xsd:integer" />
  <xsd:element name="Sinopsis" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

## 14. Ejercicio propuesto

Se propone realizar un esquema XML que valide el siguiente documento XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <Agenda>
    <ListaContactos>
      <Contacto>
        <Nombre>Pedro</Nombre>
        <Apellidos>Del Moral Cepeda</Apellidos>
        <Telefono Prefijo="91">3365895</Telefono>
        <Movil>654985632</Movil>
        <Email>pmoral@hotmail.com</Email>
        <Direccion>c/ las hayas nº 15</Direccion>
        <Poblacion Comunidad="Madrid">Pozuelo de Alarcón</Poblacion>
        <Codigo_Postal>28085</Codigo_Postal>
      </Contacto>
      <Contacto>
        <Nombre>Inma</Nombre>
        <Apellidos>Mayoral Padrón</Apellidos>
        <Telefono Prefijo="93">4785895</Telefono>
        <Movil>658596472</Movil>
        <Email>imayoral@hotmail.com</Email>
        <Direccion>c/ retamar nº 2</Direccion>
        <Poblacion Comunidad="Cataluña">Hospitalet de Llobregat</Poblacion>
        <Codigo_Postal>08085</Codigo_Postal>
      </Contacto>
    </ListaContactos>
  </Agenda>
```