

## Unidad 4. Definición de esquemas y vocabularios en XML

Repositorio GitHub con el código fuente de esta página:

<https://github.com/rafaelMunoz77/LMSGI.git>

[Introducción](#)

[DTD](#)

[DTD interno](#)

[<!ELEMENT>](#)

[<!ATTLIST>](#)

[<!NOTATION>](#)

[<!ENTITY>](#)

[Ejercicios](#)

[XSD](#)

[<xs:element>](#)

[<xs:attribute>](#)

[Tipos de datos primitivos](#)

[Definición de tipos de datos simples](#)

[Expresiones regulares \(xs:pattern\)](#)

[Unión de tipos](#)

[Listas de valores](#)

[Derivación de tipos de datos simples](#)

[Tipos de datos complejos](#)

[Tipo de datos de elemento vacío](#)

[Elementos con atributos incluidos](#)

[Elementos con contenido de texto y atributos](#)

[Elementos con hijos](#)

[Elemento con atributos y elementos descendientes](#)

[Elementos con contenido mixto \(texto y elementos\)](#)

[Elementos con atributos y contenido mixto](#)

[Derivación de tipos de datos complejos](#)

[Ejercicios](#)

## Introducción

En las unidades 1 y 2 ya nos introdujimos en una serie de conceptos relacionados con los lenguajes de marcas. En concreto vamos a repasar algunos de estos conceptos, antes de comenzar a estudiar la validación de un código XML.

XML permite definir y almacenar información estructurada, a través de etiquetas creadas por el propio usuario.

Cuando un documento XML cumple con las normas generales del lenguaje XML se dice que está "bien-formado". Para repasar dichas normas generales, te recomiendo volver a verlas en el [capítulo 2](#). A esto se le conoce como corrección sintáctica.

Podemos definir normas específicas en un lenguaje XML a través de, principalmente, dos herramientas: DTD (plantillas) y XSD (esquemas). Si un documento XML cumple con las normas específicas del lenguaje en el que está escrito se dice que es "válido". Eso se conoce como corrección semántica. En este capítulo vas a aprender a crear estos dos nuevos instrumentos.

Los documentos XML tienen una estructura jerárquica, debido a la norma que obliga a que tengan un único elemento raíz. Debido a esto encontramos conceptos como el DOM (Modelo de Datos del Documento, basado en raíz, nodos y atributos) y la forma de movernos a través del mismo, con herramientas como XPath.

en XML, al igual que en HTML, los comentarios son escritos entre los símbolos <!-- .... -->.

En XML necesitamos, al igual que en HTML, usar entidades para expresar algunos caracteres especiales.

Dentro del valor de un elemento en XML podemos utilizar las secciones CDATA. Estas secciones permiten introducir texto que no debe ser procesado por un intérprete XML, parser o similar. Ejemplo:

```

<elemento>
  <![CDATA[
    <html><body><h1>T&iacute;tulo de la p&aacute;gina</h1>
  </body></html>
  ]]>
</elemento>

```

Reglas sobre los nombres de elementos XML:

Se puede comenzar con una letra (con o sin tilde, aunque procura evitarlas) o el carácter subrayado "\_" o dos puntos ":".

A continuación de la primera letra se pueden usar letras, dígitos, subrayados, guiones bajos, comas y dos puntos.

No se puede introducir en el nombre ni caracteres de espacio ni otros caracteres de puntuación que no sean los nombrados como válidos.

El parser XML es un programa procesador de la estructura XML. El parser representará, en alguna forma, el documento XML como un árbol de información.

## DTD (Document Type Definition)

Mediante DTD podemos establecer la estructura que deberá tener un documento XML en su interior. DTD nos permite establecer una serie de reglas sobre la estructura de la información a contener en el XML, e incluso sobre la información que puede o no puede almacenarse.

DTD se utilizaba mucho antes del XML, ya en tiempos del SGML. Actualmente DTD es una herramienta desfasada, aunque fue muy utilizada al principio de XML.

Para seguir los ejemplos propuestos te recomiendo que utilices el editor XML Copy Editor. [Página web oficial](#) del proyecto.

Vamos a entrar en un ejemplo muy sencillo de DTD. Para ello necesitamos un fichero XML:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes SYSTEM "xml-dtd-01.dtd">
<superheroes>
  <superheroe>Hulk</superheroe>

```

```
<superheroe>Iron Man</superheroe>  
<superheroe>Thor</superheroe>  
</superheroes>
```

Descarga el ejemplo [aquí](#). Todas las líneas que puedes ver en el código anterior son conocidas por ti o, al menos, deberías ser capaz de comprenderlas muy fácilmente. Todas las líneas excepto la segunda. Es a través de esta segunda línea con la que estamos especificando cual es el fichero que contiene las reglas específicas del lenguaje en el que está escrito el HTML, de la siguiente forma:

!DOCTYPE -> Definición del tipo de documento. Enlace al fichero DTD con el que se valida el documento XML.

superheroes -> Nodo raíz del documento, que deberá ser especificado en el fichero DTD.

SYSTEM "xml-dtd-01.dtd" -> La palabra clave SYSTEM indica que el uso del DTD es privado. A continuación se especifica que el DTD está en un fichero externo, cuyo nombre es "xml-dtd-01.dtd". Te recomiendo que este fichero lo alojes en la misma carpeta que el DTD.

En ocasiones, en lugar de la palabra clave SYSTEM podemos encontrar la palabra clave PUBLIC, en este caso el cometido del DTD será para uso público y el valor a continuación es lo que se conoce como un FPI (Identificador Público Formal), que debe ser único a nivel global. La estructura de un FPI está compuesta por cuatro campos separados por el carácter //:

Primer campo. Indica si la norma está aprobada de manera formal o no formal. Si el DTD no ha sido aprobado de manera formal el primer campo es un signo "-". Si el DTD ha sido aprobado de manera formal por un organismo no oficial el símbolo a escribir será "+". Si el DTD se ha aprobado de manera formal por un organismo oficial, se escribe una línea de referencia del estándar.

Segundo campo. Nombre del organismo responsable del estándar.

Tercer campo. Tipo de documento, normalmente se incluye el número de versión.

Cuarto campo. Idioma del DTD

Tras el FPI, aparecerá la cadena de texto que permite localizar el

DTD. El fichero DTD tiene el siguiente contenido:

```
<!ELEMENT superheroes (superhero*)>
```

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

Descarga el DTD [aquí](#). Este DTD es realmente simple, con únicamente dos líneas. La primera hace referencia al nodo raíz del documento XML, que se denominará "superheroes". Además encontramos el texto "(superhero\*)" que indica que cada elemento superheroes estará compuesto por un conjunto de elementos de tipo "superhero".

En la segunda línea encontrarás la definición de cada elemento "superhero", en la que únicamente especificamos que se permitirá un texto libre.

Ahora deberías abrir estos dos elementos con el editor XML Copy Editor y comprobar que el XML está "bien-formado" y es "válido". No se te ocurra comprobar estos dos aspectos en el fichero DTD, recuerda que el fichero DTD no está escrito en XML ni en nada parecido a un lenguaje de marcas.

## DTD interno

En ocasiones, encontraremos que el mismo fichero contiene el XML y el DTD. En esos casos hablamos de un DTD interno. Para poder especificar un DTD interno, la única diferencia será que dentro de la declaración de DOCTYPE no utilizaremos ni la palabra SYSTEM ni PUBLIC, si no que escribiremos el bloque de reglas DTD. Un ejemplo sería.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes [
  <!ELEMENT superheroes (superhero*)>
  <!ELEMENT superhero (#PCDATA)>
]>
<superheroes>
  <superhero>Hulk</superhero>
  <superhero>Iron Man</superhero>
  <superhero>Thor</superhero>
</superheroes>
```

Descarga este ejemplo [aquí](#). Comprueba si este documento está bien-formado y es válido con XML Copy Editor.

## <!ELEMENT>

<!ELEMENT> es utilizado dentro de un DTD para indicar que se espera un elemento de un tipo concreto en el fichero XML. Su sintaxis es:

```
<!ELEMENT nombre-elemento contenido-del-elemento>
```

El nombre-elemento es el nombre que después encontraremos en el archivo XML, encerrado entre los símbolos < y >.

El contenido-del-elemento puede tomar los siguientes valores:

ANY. Si el contenido del elemento toma este valor significará que se admite cualquier cosa dentro del elemento. Esto suele hacerse de forma provisional y no debería aparecer en un DTD definitivo. Ejemplo:

```
<!DOCTYPE superheroes [  
  <!ELEMENT superheroes ANY>  
  <!ELEMENT superhero ANY>  

```

Descarga el ejemplo [aquí](#).

EMPTY. Este valor del contenido del elemento provocará que el elemento esté vacío, es decir, que no pueda admitir ningún valor en su interior. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE superheroes [  
  <!ELEMENT superheroes ANY>  
  <!ELEMENT superhero EMPTY>  
<superheroes>  
  <superhero/>  
  <superhero/>  
  <superhero/>  
</superheroes>
```

Descarga el ejemplo [aquí](#).

(#PCDATA). Parsed Character Data. Utilizamos este contenido del elemento cuando queremos que un elemento tenga su valor formado por caracteres, número, etc. pero no se podrán incluir elementos. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes [
  <!ELEMENT superheroes ANY>
  <!ELEMENT superhero (#PCDATA)>
]>
<superheroes>
  <superhero>Hulk</superhero>
  <superhero>Thor</superhero>
  <superhero>Iron Man</superhero>
</superheroes>
```

Descarga el ejemplo [aquí](#).

Otros elementos. Por supuesto, un elemento puede tener en su interior otros elementos. Para realizar esto utilizamos los diferentes símbolos de cardinalidad. Hasta ahora sólo hemos visto el símbolo '\*'. En concreto podemos utilizar:

- ? -> Un elemento puede aparecer 0 ó 1 vez.
- \* -> Un elemento puede aparecer de 0 a una cantidad ilimitada de veces.
- + -> Un elemento puede aparecer de 1 a una cantidad ilimitada de veces.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes [
  <!ELEMENT superheroes (superhero*)>
  <!ELEMENT superhero (poderes?)>
  <!ELEMENT poderes (poder+)>
  <!ELEMENT poder (#PCDATA)>
]>
<superheroes>
  <superhero></superhero> <!-- No tiene un
  elemento poderes. Esto es válido -->
```

```

<superheroe> <!-- Tiene un sólo elemento
poderes y un sólo elemento poder. Esto es válido
-->

  <poderes>
    <poder>Rayos X</poder>
  </poderes>
</superheroe>

<superheroe> <!-- Tiene un sólo elemento
poderes y varios elementos poder. Esto es válido
-->

  <poderes>
    <poder>Supervelocidad</poder>
    <poder>Superfuerza</poder>
  </poderes>
</superheroe>
</superheroes>

```

Descarga el ejemplo [aquí](#).

Los elementos dentro de otro pueden sucederse o repetirse de formas concretas. De forma básica utilizamos:

Elemento1, Elemento2 -> Si utilizo el separador ',' estaré indicando que se espera encontrar los elementos Elemento1 y Elemento2 en el orden concreto en el que los expongamos.

Elemento1 | Elemento2 -> Significa exclusión. Aparecerá el Elemento1 o el Elemento2, pero no ambos ni ninguno.

Ejemplo:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes [
  <!ELEMENT superheroes (superheroe*)>
  <!ELEMENT superheroe (nombre,
poder+,vida?)>
  <!ELEMENT nombre (#PCDATA)>
  <!ELEMENT poder (#PCDATA)>
  <!ELEMENT vida (vivo | muerto)>

```



```

<!ELEMENT vivo EMPTY>
<!ELEMENT muerto EMPTY>
]>
<superheroes>
  <superheroe>
    <nombre>Hulk</nombre>
    <poder>Superfuerza</poder>
    <poder>Rayos X</poder>
    <vida><muerto/></vida>
  </superheroe>
  <superheroe>
    <nombre>Thor</nombre>
    <poder>Martillo mágico</poder>
  </superheroe>
</superheroes>

```

Descarga el ejemplo [aquí](#).

En ocasiones podemos combinar el contenido de un elemento para que pueda admitir un conjunto de elementos, o que admita esos elementos junto con cualquier texto de tipo libre. Esto se consigue combinando los operadores '|', '\*' y '#PCDATA'. La única particularidad es el hecho de que la primera opción del contenido del elemento debe ser precisamente '#PCDATA'. De esta forma el código:

```
<!ELEMENT vida (#PCDATA | vivo | muerto)*>
```

especifica que el contenido del elemento "vida" puede ser un texto libre, un elemento "vivo" o un elemento "muerto". Este código, combinado con el operador '\*', otorga cardinalidad de 0 a ilimitado a este contenido. Ejemplo completo:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE superheroes [
  <!ELEMENT superheroes (superheroe)*>
  <!ELEMENT superheroe (nombre,
poder+,vida?)>
  <!ELEMENT nombre (#PCDATA)>
  <!ELEMENT poder (#PCDATA)>
  <!ELEMENT vida (#PCDATA | vivo | muerto)*>

```

```

<!ELEMENT vivo EMPTY>
<!ELEMENT muerto EMPTY>
]>
<superheroes>
  <superhero>
    <nombre>Hulk</nombre>
    <poder>Superfuerza</poder>
    <poder>Indestructible</poder>
    <vida><muerto/></vida>
  </superhero>
  <superhero>
    <nombre>Thor</nombre>
    <poder>Martillo mágico</poder>
  </superhero>
  <superhero>
    <nombre>Iron Man</nombre>
    <poder>Armadura</poder>
    <vida>El superheroe está desaparecido
    actualmente</vida>
  </superhero>
</superheroes>

```

Descarga el ejemplo [aquí](#).

Llegados a este punto te propongo que hagas los siguientes ejercicios:

1.- Realiza un XML, y su correspondiente DTD, que permita almacenar información sobre tarjetas de placa base de manera que cumplas los siguientes requisitos:

Se debe poder guardar información, para cada placa, de los componentes siguientes: microprocesador, memoria, modelo, fabricante. Debes guardar especial atención a la cardinalidad de cada elemento, para que sea la adecuada.

El microprocesador, la memoria y el modelo podrán almacenar texto.

El fabricante podrá ser "Asus" o "Intel". El valor del fabricante no puede ser un texto, tiene que ser un elemento vacío.

Recuerda que una placa base no puede tener más de un

fabricante, ni menos. Piensa en la cardinalidad.

Descarga la solución aquí.

2.- Realiza un XML, y su correspondiente DTD, que permita guardar información sobre correos electrónicos (en plural). Piensa en que un correo electrónico:

Puede tener varios destinatarios, y cada uno puede ser de tipo "para", "cc", "cco".

Debe tener un asunto y un texto que actúe como cuerpo del mensaje.

Puede contener, o no, archivos adjuntos. Cada archivo deberá tener un nombre, un mime type y una cifra que indicara la longitud del archivo adjunto.

Descarga la solución aquí.

## <!ATTLIST>

Los atributos de un elemento se pueden declarar en DTD, utilizando el tag <!ATTLIST>. La sintaxis a seguir es:

```
<!ATTLIST nombreElemento nombreAtributo tipoAtributo caracter>
```

Vamos a centrarnos en los tipos disponibles para el "caracter":

Un valor de texto significará simplemente un valor por defecto.

#IMPLIED. El atributo es opcional

#REQUIRED. El atributo es obligatorio, pero no tiene valor por defecto.

#FIXED. El atributo es obligatorio y tiene por defecto un valor, el único que puede tomar.

Los valores para "tipoAtributo" son:

CDATA. Caracteres sin etiquetas.

ENTITY. Una entidad, que se declarará en el DTD. No estudiaremos las entidades hasta un poco más adelante.

ENTITIES. Lista de nombres de entidades, que se deben especificar en el DTD.

Enumeración. Lista de valores excluyentes entre si. Se puede elegir uno de entre todos.

ID. Identificador único de un elemento en el XML.

IDREF. Referencia a un identificador existente. El valor del campo debe comenzar por letra y, a continuación, puede contener los caracteres válidos en los campos de tipo NMTOKEN.

IDREFS. Varios referencias a identificadores existentes.

NMTOKEN. Nombre sin espacios en blanco en su interior. Los caracteres permitidos en el valor del campo de tipo NMTOKEN son letras, dígitos, el punto, el guión, el subrayado y los dos puntos.

NMTOKENS. Lista de nombres con espacios en blanco en su

interior. NOTATION. Notación, que debe ser declarada en el DTD.

Ejemplos de uso de <!ATTLIST>

Un único tag declarando un único atributo. Haz clic [aquí](#).

Un único tag declarando varios atributos, con los diferentes caracteres que puede tomar un atributo. Haz clic [aquí](#).

Ejemplo de tipo de atributo CDATA, NMTOKEN y NMTOKENS. Haz clic [aquí](#).

Ejemplo de tipo de atributo ID e IDREF. Haz clic [aquí](#).

## <!NOTATION>

Los elementos <!NOTATION> son elementos avanzados en XML y sirve para declarar tipos de atributos que contengan datos de tipo diferente a XML, normalmente estos tipos serán los especificados por el IANA en los mime-types. Esto es un poco complicado de entender sin un ejemplo. En primer lugar te daré la sintaxis completa del tag y después estudiaremos un código. Sintaxis:

```
<!NOTATION nombre_notation SYSTEM "identificador">
```

Un código de ejemplo podría ser el siguiente:

```

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

<!DOCTYPE marvel [
  <!NOTATION JPG SYSTEM "image/jpeg">
  <!NOTATION GIF SYSTEM "image/gif">
  <!NOTATION PNG SYSTEM "image/png">
  <!ELEMENT marvel (superhero*)>
  <!ELEMENT superhero (#PCDATA)>
  <!ATTLIST superhero nombre CDATA #REQUIRED
                    tipomagen NOTATION (GIF | JPG | PNG) #IMPLIED>
]>

<marvel>
  <superhero nombre="Thor" tipomagen="JPG">Un Dios</superhero>
</marvel>

```

Descarga el ejemplo [aquí](#).

En este código de ejemplo hemos creado una definición de los elementos "marvel" y "superhero", con la particularidad de que el elemento "superhero" tiene un atributo de tipo NOTATION, y el valor de ese atributo queda restringido a una de las tres notaciones que hemos creado, para referenciar a diferentes tipos de ficheros basados en los tipos mime.

## <!ENTITY>

Las entidades son "alias" que establecemos sobre diferentes tipos de objetos. Una vez que las entidades están creadas es muy fácil utilizarlas. Existen diferentes tipos de entidades y cada uno es ligeramente más complicado que el anterior. Vamos a estudiar cada uno de ellos:

Entidad general interna. Sirve para crear un alias dentro del mismo fichero DTD. Ejemplo:

```

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

<!DOCTYPE marvel [
  <!ELEMENT marvel (superhero*)>
  <!ELEMENT superhero (#PCDATA)>
  <!ENTITY marvel "Ediciones Marvel">
]>

<marvel>

```

```

<superheroe>Thor el de &marvel;</superheroe>
<superheroe>Hulk</superheroe>
<superheroe>Iron Man</superheroe>
</marvel>

```

Descarga el ejemplo [aquí](#). En este código observamos como el alias "marvel" se utiliza para nombrar al texto "Ediciones Marvel". De esta forma, en el código XML, se puede utilizar la entidad &marvel;.

Entidad general externa. Permite que el alias referenciado se encuentre en un fichero diferente al propio DTD. Ejemplo:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE marvel [
  <!ELEMENT marvel (#PCDATA)>
  <!ENTITY marvel "Ediciones Marvel">
  <!ENTITY vengadores SYSTEM "superheroes.txt">
]>
<marvel>El equipo de vengadores &marvel; consta de
&vengadores;</marvel>

```

Descarga el ejemplo [aquí](#). Descarga el fichero superheroes.txt [aquí](#). En este ejemplo encontramos la forma de crear un texto en un fichero externo y que este sea referenciado dentro de XML a través de una entidad. Esta entidad se llama "Entidad general externa". La única diferencia con el tipo anterior es el hecho de que el texto referenciado se encuentra en otro fichero.

Entidad parámetro. La entidad es en realidad un alias para un código que utilizamos dentro del mismo DTD. Esto es muy utilizado para la reutilización de atributos entre varios elementos. Ejemplo:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE marvel [
  <!ENTITY marvel "Ediciones Marvel">
  <!ENTITY vengadores SYSTEM "superheroes.txt">
  <!ENTITY % caracteristicasHroe "<!ATTLIST
superheroe nombre CDATA #REQUIRED color CDATA
#IMPLIED">

```

```

<!ELEMENT marvel (superhero)>
<!ELEMENT superhero (#PCDATA)>
%caracteristicasHeroe;
]>
<marvel>
  <superhero nombre="Thor" color="azul">Su historia es
muy buena</superhero>
</marvel>

```

Descarga el ejemplo [aquí](#). La entidad "caracteristicasHeroe" nos permite definir unos atributos en cualquier otro lugar del DTD.

Utilizando este mismo tipo de entidad, podemos crear una mayor reutilización. Por ejemplo en el xml y dtd siguientes:

```

<!ENTITY marvel "Ediciones Marvel">
<!ENTITY vengadores SYSTEM "superheroes.txt">
<!ENTITY % caracteristicas "nombre CDATA
#REQUIRED color CDATA #IMPLIED">

<!ELEMENT marvel (superhero*, villano*)>
<!ELEMENT superhero (#PCDATA)>
<!ATTLIST superhero %caracteristicas; >

<!ELEMENT villano (#PCDATA)>
<!ATTLIST villano %caracteristicas; >

```

Descarga [aquí](#) el dtd. El xml será:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE marvel SYSTEM "dtd-interno-entidad
parametro-solo-atributos.dtd">
<marvel>
  <superhero nombre="Thor" color="azul">Su
historia es muy buena</superhero>
  <villano nombre="Loki" color="negro">Hermano
adoptivo de Thor</villano>
</marvel>

```

Descarga [aquí](#) el xml.

Entidad de datos no procesados. Esta es una combinación entre <!NOTATION> y <!ENTITY>, que permite definir atributos para un elemento que tengan que pertenecer a un determinado tipo, definido a través de <!NOTATION>. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE marvel [
  <!ENTITY marvel "Ediciones Marvel">
  <!ENTITY vengadores SYSTEM "superheroes.txt">
  <!ENTITY % caracteristicasHroe "<!--ATTLIST superheroe
nombre CDATA #REQUIRED color CDATA #IMPLIED-->">

  <!--NOTATION JPG SYSTEM "image/jpeg"-->
    <!--ENTITY thor SYSTEM "../image/Thor.jpg" NDATA JPG-->

  <!--ELEMENT marvel (superhero)>
  <!--ELEMENT superhero (#PCDATA)>
  %caracteristicasHroe;
  <!--ATTLIST superhero imagen ENTITY #IMPLIED-->
]>
<marvel>
  <superhero nombre="Thor" color="azul" imagen="thor">Su
historia es muy buena</superhero>
</marvel>
```

Descarga el ejemplo [aquí](#).

## Ejercicios de repaso

A continuación debes descargar el siguiente fichero

[Ejercicios XML-DTD con errores](#)

En los ejercicios del 01 al 09 tienes que corregir errores cometidos en la parte XML del fichero. En los ejercicios 10 al 18 debes reparar errores en la parte DTD del fichero. En los ejercicios 19 al 24 tienes que corregir errores cometidos en la parte XML del fichero. En los ejercicios 25 al 29 debes corregir errores cometidos en la parte DTD del fichero.



Ejercicio 30. Definir una DTD que valide el documento XML que puedes descargar [aquí](#). Construir un documento XML con DTD interna. Comprobar la buena formación y la validez del documento en ambos casos. Se deben tener en cuenta las siguientes características:

El número de factura (n\_fac), número de cliente (n\_cli) y número de pedido (n\_ped) son valores únicos, por cada factura, cliente y pedido distintos, y son obligatorios.

Los números de teléfono (telefono) y fax (fax) de la empresa no tienen porqué aparecer en la factura, pero siempre que lo hagan deberán tener los mismos valores (teléfono 917776688, fax 917776699).

La forma de pago puede tomar los valores “efectivo”, “tarjeta” y “plazos”.

La moneda tiene que aparecer siempre, y siempre toma al valor “euro”.

El iva tiene que aparecer siempre, y su valor no puede contener caracteres especiales

Ejercicio 31. Investiga el uso del tipo de atributos IDREFS. A continuación intenta resolver el ejercicio siguiente.

Se desea modelar en XML la información de los distintos modelos de coche que vende un concesionario. En un momento dado la lista de modelos puede estar vacía, o contener 1 o varios modelos diferentes. Cada modelo se identifica mediante una referencia única, y se puede incluir una descripción del mismo entre las que pueden aparecer el tipo de motor, potencia y consumo medio. Además, para cada modelo se puede incluir una lista de otros modelos con los que está relacionado (los otros modelos deben existir en el ejemplar del documento). El motor también debe incluir al menos una referencia a algún modelo relacionado. Para un mismo modelo se puede incluir más de un valor de consumo medio, según la velocidad. Construir una DTD que cumpla estos requisitos, y crear un ejemplar de documento válido para esa DTD con la siguiente información:

Modelo	Modelos relacionados	Descripción Motor relacionado con modelo
556	---	No disponible ---
555	444, 556	Este modelo tiene un motor 1.998cc 16v. de cuatro cilindros, que desarrolla una potencia de 128 CV. Dispone de cambio manual ... el consumo medio de este monovolumen es 11 a los 100 km. El precio base es de 2.225.000.

557 --- No disponible ---

154 --- No disponible ---

444 555, 154 Este modelo tiene los 100 km. El precio base es de 2.500.000.  
un motor 2.000cc 16v. de cuatro cilindros  
con 128 CV. Dispone de cambio manual ... el consumo medio de este monovolumen es 11 a

Descarga [aquí](#) la solución al ejercicio 31.  
556

Ejercicio 32. Repasa el uso de <!ENTITY> y realiza un fichero XML + DTD externo en el que reutilices la declaración de los atributos de un elemento en otro elemento. El tema del XML es libre.

## Validación con esquemas XML (XSD)

Los esquemas aportan las siguientes ventajas sobre los DTD:

Son documentos XML.

Disponen de muchos tipos de datos predefinidos para los elementos y los atributos.

Puede concretar la cardinalidad de los elementos con exactitud.

Se pueden mezclar diferentes vocabularios con diferentes espacios de nombres.

Los documentos de tipo XML que son validados contra un esquema determinado reciben el nombre de "instancias del esquema".

## <xs:element>

Al igual que los DTD tienen la instrucción <!ELEMENT> para la definición de elementos, en los esquemas disponemos del elemento (es XML) denominado <xs:element....>. Cualquier definición de elemento raíz debe ser obligatoriamente hija de la definición del elemento <xs:schema...>. A continuación vamos a examinar un ejemplo de validación xsd, el más simple posible, en el que podrás ver como se puede asociar un documento XML con su correspondiente xsd. En primer lugar el documento xsd:

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

Descarga el ejemplo [aquí](#).

En este primer ejemplo de xsd vemos como, en la primera línea, estamos creando un documento XML, después encontramos la definición del schema, que debe estar siempre asociada con el URI "http://www.w3.org/2001/XMLSchema". Por último encontramos la definición del elemento raíz "marvel", a través del atributo "name" del elemento <xs:element...>. Un documento XML que valida contra este esquema sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<marvel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-01.xsd">
  un montón de superhéroes
</marvel>
```

Descarga el ejemplo [aquí](#).

Fíjate en que el elemento raíz <marvel> ha incorporado un atributo llamado "xmlns:xsi" y un "xsi:noNamespaceSchemaLocation" que le permiten especificar que dicho atributo será validado contra un schema y en concreto indica la ruta del mismo, respectivamente. Esto es muy potente, ya que la definición de diferentes elementos puede encontrarse en diferentes documentos xsd.

Vamos a estudiar algunos atributos del elemento <xs:element>, seguro que recordarás similitudes con los DTD:

"name": indica el nombre que le damos a un elemento.

"type": indica el tipo del elemento. Más adelante hablaremos de los tipos, siendo esto el principal aspecto a estudiar en los xsd.

"minOccurs": indica la cardinalidad mínima de una colección de elementos. Su valor mínimo es 0 y su máximo es "unbounded" (ilimitado).

"maxOccurs": indica la cardinalidad máxima de una colección de elementos, similar a "minOccurs".

"default": valor por defecto del elemento.

"fixed": único valor posible para el elemento.

Existen más atributos para el elemento <xs:element> pero los descubriremos al final del tema, son avanzados. De momento vamos a poner algún ejemplo de uso de los atributos que acabas de leer. Primero el xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="marvel" type="xs:string" fixed="un montón de
superhéroes" />
</xs:schema>
```

Descarga el ejemplo [aquí](#).

En este ejemplo observas uno de los principales tipos "xs:string" (cadena de caracteres) y la forma de establecer un valor fijo para un elemento. Ahora el XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<marvel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-02.xsd">un montón de
superhéroes</marvel>
```

Descarga el ejemplo [aquí](#).

## <xs:attribute>

Ahora debemos ir a por los atributos, al igual que en DTD podemos utilizar la instrucción <!ATTLIST> en xsd debemos utilizar el elemento (recuerda que hablamos de un XML) <xs:attribute....>. Un ejemplo rápido de uso sería el siguiente, primero el xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType>
      <xs:attribute name="nombre" type="xs:string"
use="required"/>
      <xs:attribute name="edad" type="xs:integer" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

En este ejemplo observas como, para poder indicar atributos de un elemento, debemos especificar que el tipo del mismo debe ser complejo. Esto es muy avanzado de momento. Te pido que te centres sólo en las filas en negrita. Ahora el XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-03.xsd" nombre="Hulk"
edad="46"/>
```

Descarga el ejemplo [aquí](#).

Además de los atributos vistos en el ejemplo, también tenemos el atributo "fixed" que se comporta igual en el elemento <xs:attribute> que en el elemento <xs:element> visto anteriormente.

## Tipos de datos primitivos

Ya habrás visto muy extraño el uso del elemento <xs:complexType>, así que vamos a

ir con calma y a estudiar los tipos de datos en xsd, que es principal asunto que nos ocupará en este tema. Los tipos de datos determinan la forma que puede tomar cada elemento o incluso cada atributo. En concreto decimos que existen unos tipos de datos predefinidos y luego podemos construir tipos de datos más complejos a partir de estos. Los tipos de datos predefinidos se engloban en cinco categorías: numéricos, de fecha y hora, de texto, binarios y booleanos:

#### Numéricos:

float: flotante de 32 bits.

double: flotante de 64 bits.

decimal: número real en notación científica.

integer: cualquier número entero.

nonPositiveInteger: enteros negativos y cero

negativeInteger: menores que cero.

nonNegativeInteger: enteros positivos y cero

positiveInteger: enteros positivos.

unsignedLong: enteros positivos de 64 bits.

unsignedInt: enteros positivos de 32 bits.

unsignedByte: enteros positivos de 8 bits.

long, int, short, byte: enteros que utilizan su primer bit para el signo, sus precisiones son de 63, 31, 15 y 7 bits respectivamente

#### De fecha y hora:

dateTime: fecha y hora en formato aaaa-MM-dd T hh:mm:ss.

La "T" viene de zona horaria, por ejemplo -06:00 o +11:00.

date: fecha en formato aaaa-MM-dd.

time: hora en formato hh:mm:ss.

gDay: sólo día en formato dd.

gMonth: sólo mes en formato MM.

gYear: sólo año en formato aaaa.

gYearMonth: sólo año y mes en formato aaaa-MM.

gMonthDay: sólo mes y día en formato MM-dd.

#### De texto:

string: cadena de texto.

normalizedString: cadena de texto donde los tabuladores, nuevas líneas y retornos de carro se convierten en espacios

simples.

token: cadena de texto sin espacios delante y detras, sin tabulador, ni nuevas líneas o retornos de carro y con espacios simples en su interior.

ID: tipo de datos para atributo, debe ser único en el documento, compatible con DTD.

IDREF: referencia a un ID, compatible con DTD y utilizado sólo en atributos.

En compatibilidad con DTD, además de ID e IDREF, también encontramos los tipos IDREFS, ENTITY, ENTITIES y NOTATION. Existen otros tipos de texto, pero son más avanzados.

## Binarios

hexBinary: secuencia de dígitos en base 16 o hexadecimales. base64Binary: secuencia de dígitos en base 64.

## Booleanos

boolean: admite los valores 0, 1, true y false.

# Definición de tipos de datos simples (no primitivos)

Los tipos de datos primitivos, asignables a elementos y atributos pueden complicarse o restringirse según nuestras necesidades, para crear nuevos tipos de datos. Un ejemplo de esto sería la necesidad de crear un tipo de datos que validase que un texto es una dirección de correo electrónico, o que un número entero debe tener entre 6 y 10 cifras de longitud. La forma de realizar esto será a través del elemento `<xs:simpleType>`. A continuación estudiaremos un ejemplo de creación de un tipo de datos simple, a partir de uno primitivo y su posterior utilización como tipo de un elemento. Empecemos por el xsd:

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

  <xs:simpleType name="nombreSuperhero">
    <xs:restriction base="xs:string">
```

```

        <xs:minLength value="1" />
    </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Descarga el ejemplo [aquí](#).

En este ejemplo puedes ver como se crea un nuevo tipo de datos simple, basándonos en el tipo primitivo string (cadena de caracteres) y utilizando una restricción para establecer una longitud mínima de la cadena. Para poder utilizar este nuevo tipo de datos creado sólo se necesita darle nombre, a través del atributo "name" de `<xs:simpleType>` y utilizarlo en un elemento o atributo, a través del atributo "type" del `<xs:element>` o `<xs:attribute>` correspondiente.

En el ejemplo anterior hemos creado un tipo de datos que se puede utilizar en multitud de elementos o atributos, sin embargo, si no quisiéramos tener esa capacidad de reutilización, podríamos crear el `<xs:simpleType>` directamente dentro de un `<xs:element>` o `<xs:attribute>` y sólo afectaría al mismo. Un ejemplo de esto sería:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superhero">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>

```

Fijate que en este ejemplo no se le ha dado nombre al `<xs:simpleType>`, ni se le ha asignado tipo al `<xs:element>`, y además el elemento `<xs:simpleType>` ahora está anidado dentro del `<xs:element>`.

El documento xml correspondiente sería:

```

<?xml version="1.0" encoding="UTF-8"?>
<superhero xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-04.xsd">Hulk</superhero>

```

Descarga el ejemplo [aquí](#). Si el valor del elemento `<superhero>` tuviera



una longitud inferior a 1 carácter, el documento no sería válido.

Has observado que para crear restricciones en los valores de un tipo de datos hemos utilizado el nuevo elemento `<xs:restriction>`. Estas restricciones deben ser detalladas a través de las llamadas facetas. A continuación estudiaremos diferentes facetas para retringir diferentes tipos de datos:

`xs:minInclusive`: límite inferior de un intervalo de valores, con el límite incluido. Se usa en tipos numéricos y de fecha y hora.

`xs:maxInclusive`: límite superior de un intervalo de valores, con el límite incluido. Se usa en tipos numéricos y de fecha y hora.

`xs:minExclusive`: límite inferior de un intervalo de valores, con el límite no incluido. Se usa en tipos numéricos y de fecha y hora.

`xs:maxExclusive`: límite superior de un intervalo de valores, con el límite no incluido. Se usa en tipos numéricos y de fecha y hora.

`xs:enumeration`: lista de valores admitidos. Se usa en cualquier tipo de datos.

`xs:pattern`: patrón o expresión regular. Se estudiará a fondo un poco más adelante. Se usa en tipos de datos de texto.

`xs:whitespace`: especifica lo que hacer con los espacios en blanco, saltos de línea, retornos de carro y tabuladores. Los valores aceptados son:

`preserve` (conservar), `replace` (reemplazar por espacios en blanco) y `collapse` (convertir en un sólo espacio). Se usa en tipos de datos de texto

`xs:length`: longitud exacta de caracteres. Se usa en tipos de datos de texto.

`xs:minLength`: longitud mínima de caracteres. Se usa en tipos de datos de texto.

`xs:maxLength`: longitud máxima de caracteres. Se usa en tipos de datos de texto.

`xs:fractionDigits`: número máximo de posiciones decimales en números flotantes. Sólo se usa en datos numéricos flotantes.

`xs:totalDigits`: número exacto de dígitos. Se usa en datos de tipo numérico.

Basándote en la teoría estudiada y los ejercicios de ejemplo que has estudiado, te pido que realices, utilices y valides nuevos tipos de datos de tipo simple que cumplan las siguientes características:

Tipo numérico entero contenido en el límite inferior de 10 (no incluido) y 89 (incluido).

Tipo de datos que almacena el nombre de uno de los cuatro posibles cantantes: John Lennon, Sting, Lady Gaga, Ana Torroja. Es posible que necesites buscar en Internet el uso de la faceta que permite hacer esto en concreto.

Crear un tipo de datos numérico que permita números con 2 dígitos decimales y 10 dígitos enteros.

Tipo de datos de cadena de texto, que tenga un mínimo de 6 caracteres de longitud y un máximo de menos de 16 caracteres.

## Expresiones regulares (xs:pattern)

Las expresiones regulares se merecen un apartado propio dentro de estos apuntes, nos van a permitir toda la potencia necesaria para validar cualquier tipo de dato simple. Para ello se basan en una forma de especificar los diferentes tipos de caracteres que podemos encontrar en un texto, y una cardinalidad asociada a cada uno de ellos. En concreto, vamos a estudiar las siguientes formas de representar información:

. -> Representa cualquier caracter.

\w -> Cualquier letra, mayúscula o minúscula.

\d -> Un dígito.

\D -> Cualquier carácter que no sea un dígito.

\s -> Cualquier carácter similar a un espacio, como tabuladores, saltos de línea, etc.

\S -> Cualquier carácter que no sea similar a un espacio.

[abc] -> Cualquiera de los caracteres contenidos dentro de los corchetes, sólo se permitirá un único carácter.

[A-Z] -> Intervalo de valores, se permitirá cualquiera que este dentro del intervalo. Recuerda que los caracteres están representados a través de datos numéricos.

[^abc] -> Significa cualquier caracter que no sea alguno de los contenidos entre corchetes.

(a|b) -> uno de los dos caracteres. A efectos prácticos sería igual a [ab].

En cuanto a la cardinalidad de las ocurrencias de los caracteres tenemos:

? -> De 0 a 1 ocurrencias.

- \* -> De 0 a infinitas ocurrencias.
- + -> De 1 a infinitas ocurrencias.
- {n} -> n ocurrencias.
- {n,m} -> Mínimo de n ocurrencias y máximo de m.
- {n,} -> Mínimo de n ocurrencias y máximo de infinitas.

De esta forma podremos crear las siguientes expresiones regulares:

- \d{4,8} -> Sucesión de dígitos de un mínimo de 4 y un máximo de 8.
- \d{8}[A-Z] -> DNI con letra final en mayúscula.
- \w+ -> de 1 a infinitos caracteres
- \w+@\w+\.\w+ -> Correo electrónico
- \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} -> Dirección IPv4

A continuación puedes observar un ejemplo de uso de una expresión regular, en primer lugar el archivo xsd:

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

  <xs:simpleType name="nombreSuperhero">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{8}[A-Z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Descarga el ejemplo [aquí](#). Fíjate en el hecho de que para utilizar una expresión regular debemos basarnos en un tipo "xs:string". En el ejemplo se utiliza una expresión regular que valida un DNI (8 caracteres numéricos y una letra mayúscula). Ahora el archivo xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<superhero xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-05-exp
reg.xsd">79220905D</superhero>
```

Descarga el ejemplo [aquí](#).

Investiga que tipos de expresiones se pueden formar a partir de las siguientes expresiones regulares:

```

\d{7,8}[A-Z]
Elemento \d
a*b
[xyz]b
a?b
a+b
[a-c]x
[^0-9]x
\Dx
(pa){2}rucha
.abc
(a|b)+x
a{1,3}x
\d{3}-[A-Z]{2}
[(\d{3}[l])\s\d{3}-\d{4}

```

## Unión de tipos

En xsd tenemos la opción de crear tipos de elementos a partir de otros que existan previamente o que creamos específicamente. En el caso de las uniones podemos simplemente añadir un tipo de datos a otro (u otros) y asignarle un nombre. Un ejemplo dejará todo esto un poco más claro:

```

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

  <xs:simpleType name="talla">
    <xs:union memberTypes="talla-texto talla-numero" />
  </xs:simpleType>

  <xs:simpleType name="talla-texto">
    <xs:restriction base="xs:string">
      <xs:enumeration value="s" />
      <xs:enumeration value="m" />
      <xs:enumeration value="x" />
      <xs:enumeration value="xl" />
    </xs:restriction>
  </xs:simpleType>

```

```

<xs:simpleType name="talla-numero">
  <xs:restriction base="xs:positiveInteger">
    <xs:enumeration value="38"/>
    <xs:enumeration value="40"/>
    <xs:enumeration value="42"/>
    <xs:enumeration value="44"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Descarga el ejemplo [aquí](#).

En el archivo puedes observar como se crean dos tipos simples de elementos, llamados "talla-texto" y "talla-numero". Estos dos tipos de datos son simples enumeraciones, en un caso de texto y en otro de números enteros positivos. Los dos tipos de datos se han unido en uno a través de la instrucción "xs:union" y se le asigna el nombre "talla". Posteriormente se le asigna este tipo de datos compuesto a un elemento.

El archivo xml que valida contra el xsd anterior puede ser algo parecido a esto:

```

<?xml version="1.0" encoding="UTF-8"?>
<tallaSuperheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema
instance"
xsi:noNamespaceSchemaLocation="xml-xsd-06-
union.xsd">x</tallaSuperheroe>

```

Descarga el ejemplo [aquí](#).

## Listas de valores

En XSD podemos permitir que el contenido de un elemento o atributo se componga de una lista de valores, separados por un espacio en blanco. Para lograrlo, primero definiremos un tipo de valores permitido y después obtenemos una lista de esos valores. Es decir, derivamos el tipo de datos y creamos una modificación del original, permitiendo una lista de valores, a través del elemento "<xs:list>". Un ejemplo es:

```

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

```

```

<xs:simpleType name="listaNumeros">
  <xs:list itemType="listaNumerosRestringida" />
</xs:simpleType>

<xs:simpleType name="listaNumerosRestringida">
  <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="10"/>
    <xs:maxExclusive value="51"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

Descarga el ejemplo [aquí](#).

En el ejemplo se observa como se crea un tipo de datos llamado "listaNumerosRestringida" que permite valores únicos, positivos enteros comprendidos entre 10 y 50 (ambos incluidos). A continuación se utiliza este tipo de datos creado para crear un nuevo, que aporta la capacidad de listar o agrupar varios de estos elementos. Este nuevo tipo de datos se denomina "listaNumeros" y se puede aplicar después a elementos o atributos. Un ejemplo de utilización puede ser:

```

<?xml version="1.0" encoding="UTF-8"?>
<numeros xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-07-list.xsd">22 12 24
34</numeros>

```

Descarga el ejemplo [aquí](#).

## Derivación de tipos de datos simples

La derivación de tipos de datos simples se basa en utilizar un tipo de datos simple como punto de partida para crear otro tipo de datos simple, añadiéndole diferentes facetas que sirvan para restringir, aún más, el dato. El tipo de datos "original" recibe el nombre de tipo de datos "padre" y el tipo de datos "derivado" recibe el nombre de tipo de datos "hijo". Un ejemplo de esto puede ser el siguiente esquema:

```

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

```

```
<xs:element name="NumeroDelimitado" type="numero-50-60"/>
```

```
<xs:simpleType name="numero-50-60">
```

```
  <xs:restriction base="numero-10-100">
```

```
    <xs:minInclusive value="50" />
```

```
    <xs:maxInclusive value="60" />
```

```
  </xs:restriction>
```

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

```
<xs:simpleType name="numero-10-100">
```

```
  <xs:restriction base="xs:positiveInteger">
```

```
    <xs:minInclusive value="10"/>
```

```
    <xs:maxInclusive value="100"/>
```

```
  </xs:restriction>
```

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

```
</xs:schema>
```

Descarga el ejemplo [aquí](#).

En el ejemplo puedes observar como el tipo de datos llamado "numero-10-100" sirve como punto de partida para el tipo de datos "numero-50-60", que es el que finalmente se utiliza como tipo del elemento. Un código xml que valide contra dicho esquema puede ser el siguiente:

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

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

```
xsi:noNamespaceSchemaLocation="xml-xsd-08-ejemplo
```

```
Derivacion_de_tipos.xsd">51</NumeroDelimitado>
```

Descarga el ejemplo [aquí](#).

Puedes observar como si cambias el valor de "51", y le haces que salga del intervalo [50-60], el código dejara de ser válido.

## Tipos de datos complejos

Entramos de lleno en el tercer tipo de datos: los tipos de datos complejos. Tienes que recurrir a este tipo de datos cada vez que desees que un elemento pueda admitir atributos y/o elementos hijos. Al principio vas a estudiar las tres primeras formaciones

admitidas:

Secuencia (sequence). Se podrán incorporar elementos hijos en un determinado orden y con una cardinalidad determinada.

Elección (choice). De los elementos declarados sólo podrás utilizar uno, con una determinada cardinalidad.

Todos (all). Puede utilizar todos los elementos declarados y en cualquier orden y con cardinalidad determinada. Se desaconseja el uso de esta estructura.

Para introducir la cardinalidad de los elementos puedes utilizar los atributos de "`<xs:element...>`": "`minOccurs`" y "`maxOccurs`". A través de los anteriores podrás establecer la cardinalidad mínima y máxima de cada elemento. Sin embargo, recuerda que el hecho de estar dentro de una secuencia, una elección o una estructura de tipo "todos" limita la cantidad de ocurrencias que puedes utilizar para cada elemento. La cardinalidad mínima, de forma general, es 0 (cero) y la máxima es ilimitada "unbounded".

Un ejemplo de uso de la secuencia:

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

  <xs:complexType name="caracSuperheroe">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="edad" type="xs:string" />
      <xs:element name="bando" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

En esta secuencia, la cardinalidad mínima de los elementos: "nombre", "edad" y "bando" es de 0 y la máxima de ilimitado. Un xml válido sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-09-sequence.xsd">
```



```
<nombre>Hulk</nombre>
<edad>86</edad>
<bando>Buenos</bando>
</superheroe>
```

Descarga el ejemplo [aquí](#).

Un ejemplo de como utilizar una estructura de opción (choice) es el siguiente:

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

  <xs:complexType name="caracSuperheroe">
    <xs:choice>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="edad" type="xs:string" />
      <xs:element name="bando" type="xs:string" />
    </xs:choice>
  </xs:complexType>

</xs:schema>
```

Descarga el ejemplo [aquí](#).

La única diferencia entre el ejemplo de secuencia y el de opción es el elemento "xs:choice". La cardinalidad mínima es de 0 y la máxima de ilimitado, para cada elemento. Un ejemplo de xml que valida con este esquema es:

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-10-choice.xsd">
  <nombre>Hulk</nombre>
  <!-- <edad>86</edad>
  <bando>Buenos</bando> -->
</superheroe>
```

Descarga el ejemplo [aquí](#).

Fíjate como ahora sólo puedes utilizar uno de los elementos: nombre, edad y bando.

La estructura de "todos" (all) encuentra un ejemplo en el siguiente código:

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

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superhero" type="caracSuperhero"/>

  <xs:complexType name="caracSuperhero">
    <xs:all>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="edad" type="xs:string" />
      <xs:element name="bando" type="xs:string" />
    </xs:all>
  </xs:complexType>

</xs:schema>

```

Descarga el ejemplo [aquí](#).

En la estructura de "todos" la cardinalidad mínima de cada elemento es de 0 y la máxima de 1. Recuerda que estas cardinalidades se pueden establecer a través de los atributos de "xs:element": "minOccurs" y "maxOccurs".

## Tipo de datos de elemento vacío

Para permitir que un elemento tenga vacío su contenido debemos utilizar un tipo de datos complejo, en una forma muy sencilla. Un ejemplo es:

```

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

```

Descarga el ejemplo [aquí](#).

Un código xml que valida contra el esquema anterior es:

```

<?xml version="1.0" encoding="UTF-8"?>
<superhero xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-12-elemento-vacio.xsd">
</superhero>

```

Descarga el ejemplo [aquí](#).

## Elementos con atributos incluidos

Para poder incluir atributos en un elemento necesitamos utilizar los tipos de datos complejos. Es muy sencillo hacerlo a través del elemento "xs:attribute". Un ejemplo de código es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType>
      <xs:attribute name="nombre" type="xs:string" />
      <xs:attribute name="edad" type="xs:positiveInteger" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

Un código xml que valida contra el esquema anterior es:

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-13-elemento-vacio-con
atributos.xsd"
nombre="Hulk" edad="86"></superheroe>
```

Descarga el ejemplo [aquí](#).

## Elementos con contenido de texto y atributos

Para soportar un tipo de contenido de texto y atributos en el elemento debemos recurrir a los elementos "xs:simpleContent" y "xs:extension". Un ejemplo es:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="nombre" type="xs:string" />
          <xs:attribute name="edad" type="xs:positiveInteger" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:schema>

```

Descarga el ejemplo [aquí](#).

Un ejemplo de código xml que valida contra este xsd es:

```

<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-13-elemento-contenido
texto-y-atributos.xsd"
nombre="Hulk" edad="86">Buen superheroe</superheroe>

```

Descarga el ejemplo [aquí](#).

## Elementos con hijos

Como ya has estudiado las estructuras que permiten a un nodo tener hijos: sequence, choice y all. A continuación te pongo un ejemplo de algo un poco más complejo, donde se utiliza la derivación de tipos de datos simples, los tipos de datos sin nombre (internos a un elemento "xs:element") y un ejemplo de secuencias de nodos hijos:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="amigo" type="amigosSuperheroe"
minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="amigosSuperheroe">
    <xs:sequence>

```

```

    <xs:element name="amigoDeSuperheroe" type="textoNoVacio"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="textoNoVacio">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Descarga el ejemplo [aquí](#).

Un ejemplo de código xml que valida contra el esquema anterior es:

```

<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-15-elemento-con
descendientes.xsd">
  <nombre>Hulk</nombre>
  <amigo>
    <amigoDeSuperheroe>Thor</amigoDeSuperheroe>
  </amigo>
  <amigo>
    <amigoDeSuperheroe>Hulk</amigoDeSuperheroe>
  </amigo>
  <amigo>
    <amigoDeSuperheroe>Bulleye</amigoDeSuperheroe>
  </amigo>
</superheroe>

```

Descarga el ejemplo [aquí](#).

## Elemento con atributos y elementos descendientes

Basándonos en el ejemplo anterior, podemos crear elementos con elementos hijos y atributos. Un ejemplo es:

```

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

```

```

<xs:element name="superheroe">
  <xs:complexType>
    <xs:all>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="amigo" type="textoNoVacio" minOccurs="0"
maxOccurs="1"/>
    </xs:all>
    <xs:attribute name="edad" type="textoNoVacio"/>
    <xs:attribute name="color" type="color"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="textoNoVacio">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="color">
  <xs:restriction base="xs:string">
    <xs:enumeration value="rojo"/>
    <xs:enumeration value="verde"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Descarga el ejemplo [aquí](#).

Un ejemplo de código xml que valida contra el anterior esquema es:

```

<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-16-elemento-atributos-y
descendientes.xsd"
edad="86" color="verde">
  <nombre>Hulk</nombre>
  <amigo>Thor</amigo>
</superheroe>

```

Descarga el ejemplo [aquí](#).

## Elementos con contenido mixto (texto y elementos)

Para permitir el contenido mixto habilitaremos el atributo de "xs:complexType" correspondiente, llamado "mixed" y estableceremos un valor de verdadero en el mismo. Un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType mixed="true">
      <xs:choice>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="amigo" type="textoNoVacio" minOccurs="2"
maxOccurs="unbounded"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="textoNoVacio">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

Un ejemplo de código xml que valida con el esquema anterior es:

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-17-elemento-contenido
mixto.xsd">

Este texto no se podría poner sin especificar contenido mixto

  <amigo>Thor</amigo>
  <amigo>Iron Man</amigo>
  <amigo>Bulleye</amigo>
</superheroe>
```

Descarga el ejemplo [aquí](#).

## Elementos con atributos y contenido mixto

A partir del ejemplo anterior, sólo tenemos que incluir varios elementos de tipo "xs:attribute" para dotar de atributos a la estructura anterior. Un ejemplo es:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe">
    <xs:complexType mixed="true">
      <xs:choice>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="amigo" type="xs:string" minOccurs="2"
maxOccurs="unbounded"/>
      </xs:choice>
      <xs:attribute name="edad" type="xs:string"/>
      <xs:attribute name="color" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

Un código xml válido con el ejemplo anterior es:

```
<?xml version="1.0" encoding="UTF-8"?>
<superheroe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml-xsd-18-elemento-con-atributos-
y-contenido-mixto.xsd"
edad="86" color="verde">Este texto no se podría poner sin especificar
contenido mixto
  <amigo>Thor</amigo>
  <amigo>Iron Man</amigo>
  <amigo>Bulleye</amigo>
</superheroe>
```

Descarga el ejemplo [aquí](#).



## Derivación de tipos de datos complejos

Por último, para finalizar el tema vas a encontrar un ejemplo de derivación de tipos de datos complejos. Es decir, vas a estudiar como un tipo de datos complejo se ve "extendido" con nuevas características, tales como un nuevo elemento y nuevos atributos. El código de ejemplo es:

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

  <xs:complexType mixed="true" name="superheroePrimitivo">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="amigo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="superheroeDerivado">
    <xs:complexContent mixed="true">
      <xs:extension base="superheroePrimitivo">
        <xs:sequence>
          <xs:element name="poderPrincipal" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="edad" type="xs:string"/>
        <xs:attribute name="color" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Descarga el ejemplo [aquí](#).

Observa el ejemplo para comprender que se ha comenzado creando un tipo de datos complejo llamado "superheroePrimitivo" que tiene en su interior simplemente una secuencia de dos elementos. A continuación se ha partido de este tipo de datos para crear otro, llamado "superheroeDerivado" que incorpora un nuevo elemento y dos atributos.

Una última advertencia: si utilizas el atributo "mixed=true" en el elemento "xs:element" de un tipo de datos "padre", lo tienes que utilizar obligatoriamente en el tipo de datos "hijo".

## Ejercicios

1.- Realiza un esquema xsd que valide el siguiente xml:

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

Descarga el fichero [xml](#) y el fichero [xsd](#) de solución de este ejercicio

2.- Escribir un XML Schema para el siguiente documento XML, e incluir los cambios necesarios en el mismo para referenciar al esquema creado. Se debe cumplir también lo siguiente:

Los elementos "vehículo", "nombre" y "modificacion" deben aparecer mínimo una vez, y el máximo no está limitado. El resto de los elementos deben aparecer 1 vez.

Todos los elementos que aparecen en el documento instancia de abajo son obligatorios y deben aparecer siempre en el mismo orden.

Los elementos que contienen información de fecha son todos de tipo cadena.

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

```
<vehiculos>
```

```
  <vehiculo>
```

```
    <nombre>Count
```

```
    Zero</nombre> <modelo>Series I, 80"
```

```
  </modelo> <fabricacion>
```

```
    <inicio>
```

```
      <dia>21</dia> <mes>July</mes> <anyo>1949</anyo>
```

```

        </inicio> <fin>

        <dia>9</dia> <mes>August</mes> <anyo>1949</anyo>

    </fin>

    </fabricacion> <modificaciones>

        <modificacion>Change
        Engine</modificacion> <modificacion>Change
        pedals</modificacion> <modificacion>Change
        gearbox</modificacion> <modificacion>Fit
        Rollcage</modificacion>

    </modificaciones>

</vehiculo>

</vehiculos>

```

Descarga el fichero [xml](#) y el fichero [xsd](#) de la solución de este ejercicio.

3.- Realiza de nuevo el ejercicio 30 del bloque de conocimientos de DTD, validando el xml a través de un esquema xsd de tu invención.

Descarga el fichero [xml](#) y el fichero [xsd](#) de la solución de este ejercicio

4.- Realiza de nuevo el ejercicio 31 del bloque de conocimientos de DTD, validando a través de un esquema xsd. Los atributos de tipo identificador deberán ser convertidos a tipo texto.

Descarga el fichero [xml](#) y el fichero [xsd](#) de la solución de este ejercicio.

5.- Descarga el [xml](#) del archivo y realiza un archivo [xsd](#) que lo

valide. 6.- Descarga el [xml](#) del archivo y realiza un archivo [xsd](#) que

lo valide.