

# XML. Almacenamiento de datos

## CONTENIDO

- Introducción
- Documentos XML
- Estructura jerárquica de un documento XML
- Modelo de datos de un documento XML. Nodos
- Corrección sintáctica: documento XML bien formado
- Documentos XML válidos
- Validación de documentos XML con DTD
- Validación de documentos XML con esquemas XML
- Otros mecanismos para validar XML
- Otros lenguajes basados en XML
- Otras formas de almacenar información
- Ejercicios propuestos

## 4.1. Introducción

En este tema se van a tratar, de manera casi exclusiva, aspectos relativos a una tecnología aparecida hace ya algunos años: el **XML**. Se trata de un formato de almacenamiento de información a base de etiquetas o marcas definidas por el usuario.

Por el hecho de ser un lenguaje de marcas, deberá cumplir una serie de reglas que harán que un **documento XML esté bien formado**: la forma en la que se abren y cierran las etiquetas, la forma en la que se escriben sus atributos, la existencia de un elemento que los contenga a todos, la aparición de comentarios y su formato, etc.

Adicionalmente, se aprenderán mecanismos para **validar un documento XML**. Esto significa “crear” un lenguaje de marcas para un uso específico, ya que se indica qué elementos y atributos pueden aparecer (vocabulario), el orden en el que aparecen, qué elemento contiene a cual, qué atributos tiene un elemento, qué elementos o atributos son optativos y cuáles son obligatorios, etc.

Aquí se revisarán fundamentalmente dos técnicas: los **DTD** y los **esquemas XML**. La primera es una técnica algo obsoleta pero muy extendida hasta principios de los 2000. La segunda, que es una técnica más sofisticada que permite unos niveles de definición mucho más precisos, se revisará en profundidad.

Se describirán brevemente otras técnicas mucho menos usadas para validar documentos XML: Relax NG y Schematron.

Se hará un repaso ligero de **algunos lenguajes basados en XML** para usos específicos, como SVG, WML, RSS, FO...

Se citarán dos formatos de almacenamiento de información alternativos al XML: **JSON** y **YAML**.

Este es un **capítulo prioritario** del libro. Los **contenidos más importantes** del capítulo son la comprensión del formato XML, qué representa que un documento esté bien formado, las técnicas de validación de documentos XML, sobre todo los esquemas.

## 4.2. Documentos XML

El XML (eXtensible Markup Language – Lenguaje de Marcado eXtensible) es un estándar (o norma), no una implementación concreta.

Es un metalenguaje de marcas, lo que significa que no dispone de un conjunto fijo de etiquetas (como sucedía con HTML) que todo el mundo debe conocer. Por el contrario, XML permite definir a los desarrolladores los elementos que necesiten y con la estructura que mejor les convenga.

Define una sintaxis general para maquetar datos con etiquetas sencillas y comprensibles al ojo humano (a cualquier humano). Provee, asimismo, un formato estándar para documentos informáticos. Es un formato flexible, de manera que puede ser adaptado al campo de aplicación que se deseé.

**Ejemplo:**

En el campo de la química, tendría sentido la existencia de etiquetas como <átomo>, <molécula> o <enlace>.

En el campo de la composición musical, tendría sentido que hubiera etiquetas como <entera>, <negra> o <semitonos>.

De ahí viene el extensible del nombre XML: se pueden crear nuevas etiquetas en función de las necesidades.

**Lo que no es XML**

- XML no es un lenguaje de programación, de manera que no existen compiladores de XML que generen ejecutables a partir de un documento XML.
- XML no es un protocolo de comunicación, de manera que no enviará datos por nosotros a través de internet (como tampoco lo hace HTML). Protocolos de comunicación son HTTP (Hyper-Text Transfer Protocol – Protocolo de Transferencia de Hipertexto), FTP (File Transfer Protocol – Protocolo de Transferencia de Ficheros), etc. Estos y otros protocolos de comunicación pueden enviar documentos con formato XML.
- XML no es un sistema gestor de bases de datos. Una base de datos relacional puede contener campos del tipo XML. Existen, incluso, bases de datos XML nativas, que todo lo que almacenan son documentos con formato XML. Pero XML en sí mismo no es una base de datos.
- No es propietario, es decir, no pertenece ninguna compañía, como sucede con otros formatos.

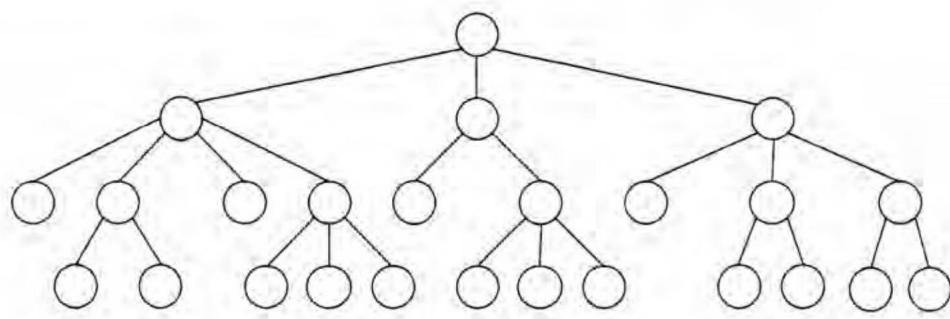
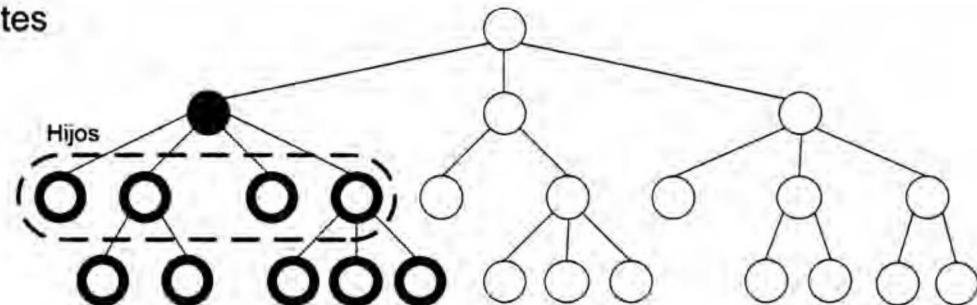
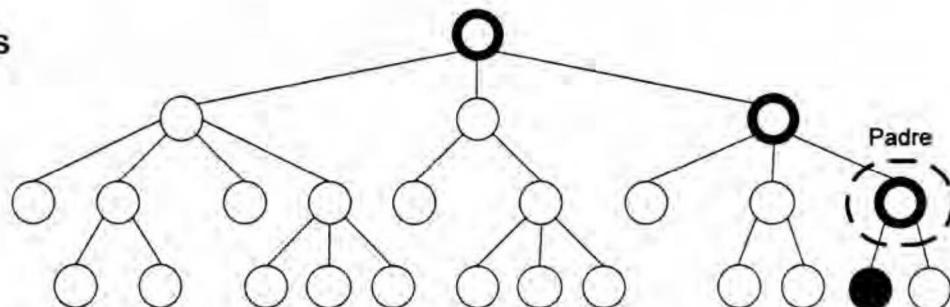
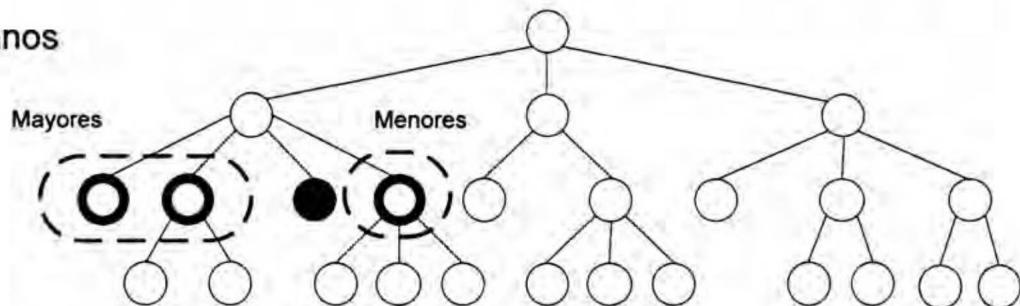
**Formato adecuado para el almacenamiento y la transmisión**

Puesto que es un formato de *texto plano* (no se trata de archivos binarios) es adecuado para almacenar información y trasmisirla. Con el más simple editor de textos se puede editar un documento XML. Asimismo, los documentos XML son relativamente ligeros para ser almacenados y enviados, puesto que ocupan lo que los datos que contienen más las etiquetas que los delimitan. Suelen tener extensión .xml, aunque no es imprescindible.

Estas etiquetas (y sus atributos) son meta-information, es decir, información sobre la información (sobre los datos). Nos permiten estructurar el documento y facilitan su procesamiento, pero no son información en sí mismas.

**4.3. Estructura jerárquica de un documento XML**

En un documento XML la información se organiza de forma jerárquica, de manera que los elementos del documento se relacionan entre sí mediante relaciones de padres, hijos, hermanos, ascendentes, descendentes, etc.

**Original****Descendentes****Ascendentes****Hermanos****Figura 4.1:** Relaciones entre nodos

A esta estructura jerárquica se la denomina árbol del documento XML. A las partes del árbol que tienen hijos se las denomina nodos intermedios o ramas, mientras que a las que no tienen se conocen como nodos finales u hojas.

Ejemplo:

El elemento `<persona>` es “padre” (contiene) a los elementos `<nombre>` y `<apellido>`, que son “hermanos” entre sí.

```
<persona>
    <nombre>María</nombre>
    <apellido>González</apellido>
</persona>
```

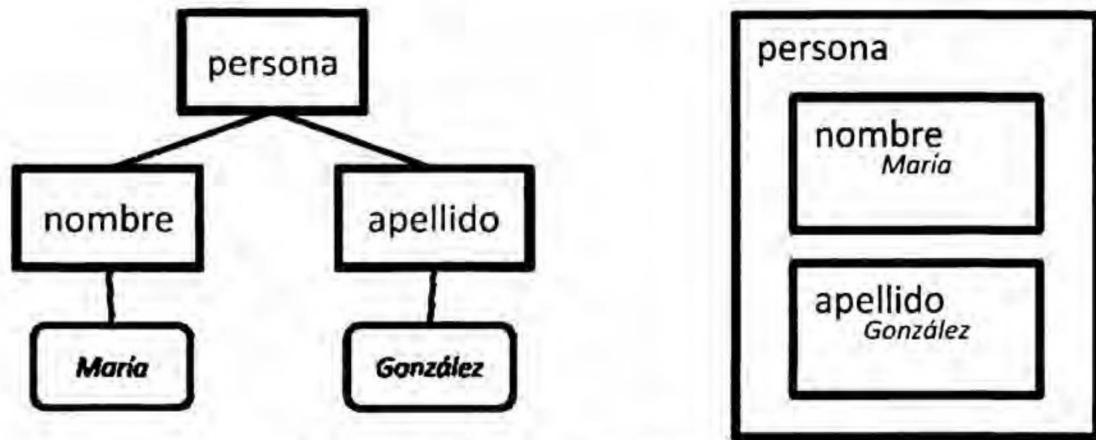


Figura 4.2: Modelo de datos: diagrama de árbol y diagrama de cajas

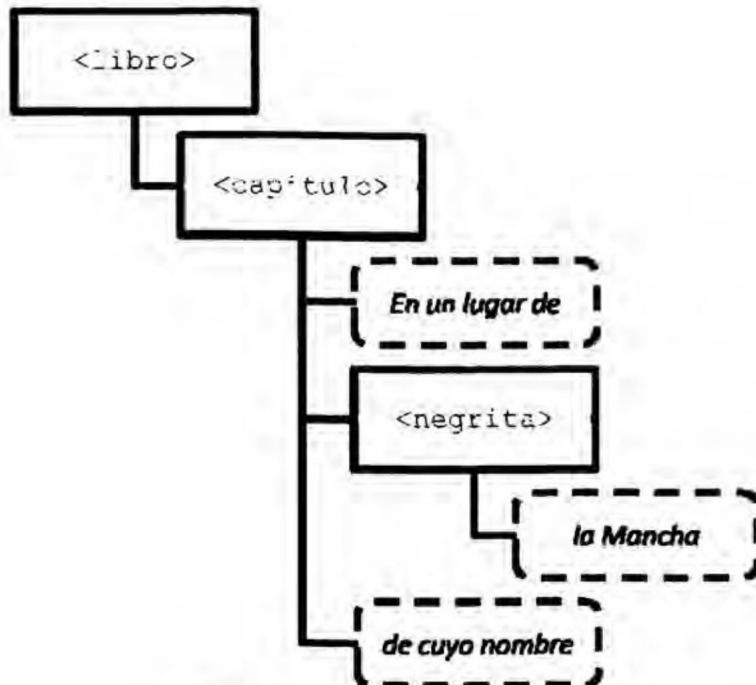
Ejemplo:

Figura 4.3: Estructura de nodos

```

<libro>
  <capítulo>
    En un lugar de <negrita>la Mancha</negrita> de cuyo nombre...
  </capítulo>
</libro>

```

El elemento `<capítulo>` tiene tres hijos:

- El nodo hoja (contenido textual) *En un lugar de*
- El nodo `<negrita>`
- El nodo *de cuyo nombre...*

## Visualización de un documento XML

No dispone de una visualización concreta en un navegador puesto que el documento no refleja una apariencia, sino unos datos.

Existen varias maneras para representar visualmente los datos de un documento XML:

- Una forma es mediante una hoja de estilo CSS que indique al navegador cómo convertir cada elemento del documento XML en un elemento visual. Se lograría con la siguiente instrucción de procesamiento que se verá más adelante:

```
<?xml-stylesheet type="text/css" href="estilos.css"?>
```

- Otra manera es mediante el uso de una hoja de transformaciones XSLT. La instrucción equivalente para asociar a un documento XML una hoja de transformaciones XSLT es:
- ```
<?xml-stylesheet type="text/xsl" href="transforma.xsl"?>
```
- También se podría hacer mediante el uso de un lenguaje de programación, como Java o JavaScript, que procese el documento XML.

## 4.4. Modelo de datos de un documento XML. Nodos

Como se ha comentado, un documento XML consta de una determinada estructura, formada por los siguientes tipos de componentes o nodos:

- Raíz:

Por encima de cualquier elemento se ubica el nodo raíz, que se designa como “/”. No es un componente que tenga representación dentro del documento XML, pero se utilizará más adelante como punto de partida para recorrer el árbol XML y ubicar el resto de nodos.

- Elementos:

Es la unidad básica de un documento XML. Son delimitadores/contenedores de información. Al igual que los elementos de HTML, se identifican por una etiqueta de apertura (como `<persona>`) y una de cierre (como `</persona>`). Lo que se ubica entre ambas es el contenido de ese elemento, que puede ser textual, otros elementos o vacío.

`<persona> ... </persona>`  
 Apertura Contenido Cierre

Algunos tipos de elementos especiales son:

- Elemento raíz: todo documento XML bien formado debe contener un único elemento raíz que contiene a todos los demás (no tiene ascendentes ni hermanos). También se le llama elemento documento.
- Elementos sin contenido: aunque puede tener atributos, se abre y se cierra con una sola etiqueta.

Ejemplo:

Elemento sin contenido y sin atributos. `<separador />`

Elemento sin contenido pero con atributos. `<separador cantidad="7" />`

- Atributos:

Son como los atributos de HTML. Son pares nombre-valor que permiten especificar datos adicionales de un elemento. Se ubican en la etiqueta de apertura del elemento. Para asignar un valor a un atributo se utiliza el signo igual. Todos los atributos, independientemente del tipo de datos que representen, se tratarán como texto y aparecerán entre comillas simples o dobles.

Se usan fundamentalmente para almacenar información sobre la información (meta-information) contendida en el documento.

Ejemplo:

```
<distancia unidades="km">70</distancia>
```

También se emplean para recoger información identificativa del elemento que permita distinguirlo de otro elemento.

Ejemplo:

```
<persona nif="12345678Z">...</persona>
```

- Texto:

El texto, como tipo de nodo que representa los datos del documento XML, puede aparecer, bien como contenido de un elemento, bien como valor de un atributo. No puede aparecer en ningún otro lugar.

Un tratamiento especial tienen los espacios en blanco. Hay cuatro tipos de caracteres de espaciado en XML:

- Tabulador: \t      &#9;      TAB
- Nueva línea: \n      &#10;      LF
- Retorno de carro: \r      &#13;      CR
- Espacio: \s      &#32;      NO-BREAK SPACE

Dentro del contenido textual de un elemento se mantendrán como están y así serán tratados por el procesador. Como valor de un atributo, los espacios en blanco adyacentes se condensarán en uno solo. Los espacios en blanco entre elementos serán ignorados.

Ejemplo:

|                               |        |                              |
|-------------------------------|--------|------------------------------|
| <A> Dato </A>                 | $\neq$ | <A>Dato</A>                  |
| <A b="otro dato" />           | =      | <A b="otro dato" />          |
| <A>Más datos</A> <B>Y más</B> | =      | <A>Más datos</A><B>Y más</B> |

- Comentarios:

Son iguales que los de HTML. Empiezan por los caracteres `<!--` y se cierran con los caracteres `-->`. Dentro de ellos se puede escribir cualquier signo (sin necesidad de caracteres de escape) menos el doble guion (`--`) que confundiría al analizador con un posible cierre del comentario.

Pueden ubicarse en cualquier lugar de un documento, menos dentro de una etiqueta de apertura, ni dentro de una etiqueta de cierre.

- Espacio de nombres:

Es un mecanismo para distinguir etiquetas cuando se mezclan distintos vocabularios. Se verán en detalle más adelante.

- Instrucciones de procesamiento:

Las instrucciones de procesamiento empiezan por `<?` y terminan por `?>`. Son instrucciones para el procesador XML, de manera que son dependientes de él. No forman parte del contenido del documento XML. Se utilizan para dar información a las aplicaciones que procesan el documento XML.

Ejemplo:

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

- Entidades predefinidas:

Representan caracteres especiales de marcado, pero son interpretados como texto por parte del procesador XML.

| Entidad                 | Carácter |
|-------------------------|----------|
| <code>&amp;amp;</code>  | &        |
| <code>&amp;lt;</code>   | <        |
| <code>&amp;gt;</code>   | >        |
| <code>&amp;apos;</code> | '        |
| <code>&amp;quot;</code> | "        |

Ejemplo:

```
<libreria>Barnes & Noble</libreria>
```

El contenido del elemento `<bebida>` será interpretado por el analizador como Barnes & Noble.

- Secciones CDATA:

Son conjuntos de caracteres que el procesador no debe analizar (parecidos a un comentario). La definición de estas secciones permite agilizar el análisis del documento y deja libertad al autor del documento para introducir libremente en ellas caracteres como < y &.

No pueden aparecer antes del elemento raíz ni después de su cierre. No pueden contener el propio signo delimitador de final de sección CDATA, es decir, la combinación de caracteres ]]>

Ejemplo:

Se quiere introducir como contenido de un elemento <codigo> un trozo de código HTML y se desea que el analizador XML no lo tome como etiquetas, sino que lo deje sin procesar.

```
<codigo>
  <![CDATA[
    <html><body><h3>Título de la página</h3></body></html>
  ]]>
</codigo>
```

- Definición de Tipo de Documento (DTD):

Permite definir reglas que fuercen ciertas restricciones sobre la estructura de un documento XML aunque su existencia no es obligatoria. Un documento XML bien formado que tiene asociado un documento de declaración de tipos y cumple con las restricciones allí declaradas se dice que es **válido**. Más adelante se verán en profundidad éste y otros mecanismos de validación de documentos XML.

Debe aparecer en la segunda línea del documento XML, entre la instrucción de procesamiento inicial y el elemento raíz.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> ①
<!DOCTYPE document system "elementos.dtd"> ②
<!-- Aquí viene un comentario --> ③
<?xml-stylesheet type="text/css" href="elementos.css"?> ④
<elementos> ⑤
  <elemento> ⑥
    <nombre>Agua</nombre>
    <color>Azul</color>
  </elemento>
  <elemento> ⑥
    <nombre>Fuego</nombre>
    <color>Rojo</color>
  </elemento>
</elementos> ⑦
```

① Instrucción de procesamiento que declara que el documento está en formato XML.

② Inclusión de un DTD externo, ubicado en el archivo elementos.dtd.

③ Un comentario.

- ④ Instrucción de procesamiento que vincula al documento XML una hoja de estilos ubicada en el archivo elementos.css.
- ⑤ Elemento raíz (también llamado elemento documento). En este punto se abre.
- ⑥ Diversos elementos descendientes del elemento raíz y los descendientes de éstos.
- ⑦ Cierre del elemento raíz.

## Nombres XML

En XML se utilizan una serie de reglas para definir nombres correctos de elementos. Son las mismas que para los atributos.

Un nombre XML, que así se denomina:

- Puede empezar con una letra (con o sin tilde), que podría ser de un alfabeto no latino, subrayado o dos puntos (este último carácter es desaconsejado ya que se reserva su uso para los espacios de nombres).
- Los siguientes caracteres pueden ser letras, dígitos, subrayados, guiones bajos, comas y dos puntos.
- Los nombres que empiezan por las letras XML, en cualquier combinación de mayúsculas y minúsculas, se reservan para estandarización.
- No pueden contener:
  - Ningún carácter de espacio.
  - Ningún otro carácter de puntuación que los ya citados como válidos. Esto incluye: comillas simples o dobles, signo de dólar, acento circunflejo, signo de porcentaje, punto y coma.

### Ejemplo:

Nombres correctos:

```
<Número_Seguridad_Social>50-12345678</Número_Seguridad_Social>
<primerApellido>Rodríguez</primerApellido>
<_cuenta_Tweeter>@follower</_cuenta_Tweeter>
<персна>Галина"Нванов</персна>
```

Nombres incorrectos:

```
<O'Donnell>General</O'Donnell> ①
<día/mes/año></día/mes/año> ②
<fecha nacimiento>2011-02-02 </fecha nacimiento> ③
```

- ① No es válido por el carácter apóstrofe
- ② No es válido por el signo de dividir
- ③ No es válido por el espacio

## Uso de elementos frente a uso de atributos

Los elementos:

- se emplean para representar jerarquías o contenido de unos dentro de otros.
- se pueden extender con otros elementos en su interior.
- el orden en el que aparecen es representativo.
- pueden tener atributos.
- puede haber múltiples ocurrencias de un elemento.

Los atributos:

- van asociados a los elementos.
- son modificadores de la información.
- se suelen usar para registrar metadatos.
- el orden en que aparecen dentro del elemento al que van asociados no es representativo.
- no se pueden extender con otros elementos contenidos en su interior.
- no puede haber múltiples ocurrencias de un atributo dentro de un mismo elemento.

## Espacios de nombres

Es un mecanismo para evitar conflictos de nombres, de manera que se puedan diferenciar elementos o atributos dentro de un mismo documento XML que tengan idénticos nombres pero diferentes definiciones. No aparecieron en la primera especificación de XML, pero sí pronto después.

Se declaran como atributo de elementos de la forma:

```
<nombre_elemento xmlns:prefijo="URI_del_espacio_de_nombres">
```

Y se usan anteponiendo a elementos y atributos con el prefijo asociado al espacio de nombres además del carácter dos puntos ":".

### Ejemplo:

```
<info:pedido xmlns:info="empresa:espacios:info">
  <info:item info:id="i_13">Afeitadora eléctrica</info:item>
  ...
</info:pedido>
```

Cualquier cadena de texto puede ser usada como prefijo del espacio de nombres. El URI del espacio de nombres sí debe ser único, aunque realmente no se comprueba mediante conexión alguna. El URI no es más que un nombre lógico del espacio de nombres. A veces se usa como URI <http://~>.

El **ámbito** de declaración o uso de un espacio de nombres, incluye los lugares donde se puede referenciar este espacio de nombres. Cubre el elemento donde se ha declarado y sus elementos descendientes. A todos estos elementos se les puede anteponer el prefijo del espacio de nombres.

Asimismo, se puede declarar un espacio de nombres diferente para un elemento descendiente de otro, en el cual ya se declarado otro espacio de nombres.

Ejemplo:

En un documento XML se declaran dos espacios de nombres, `empresa:espacios:dept` y `empresa:espacios:emp`, de prefijos `dept` y `emp` respectivamente.

```
<dept:departamentos xmlns:dept="empresa:espacios:dept"
                     xmlns:emp="empresa:espacios:emp">
    <dept:departamento dept:deptno="10">
        <emp:empleado emp:empno="7654">
            <emp:nombre>Roberto</sec:nombre>
            <emp:apellido>Mate</emp:apellido>
        </emp:empleado>
        <emp:empleado emp:empno="7891">
            <emp:nombre>Lucía</emp:nombre>
            <emp:apellido>Palmito</emp:apellido>
        </emp:empleado>
    </dept:departamento>
</dept:departamentos>
```

Los atributos pueden pertenecer al mismo espacio de nombres al que pertenece el elemento en el que están asociados o a otro diferente. La definición es la misma que con los elementos, anteponiendo el prefijo del espacio de nombres y los dos puntos al nombre el atributo. Si no aparece ningún prefijo de espacio de nombres, el atributo no pertenecerá a ningún espacio de nombres.

Ejemplo:

El atributo `trabajo:empno` del elemento `<emp:datosPersonales>` pertenece a un espacio de nombres diferente que éste. El prefijo del espacio de nombres del atributo es `trabajo`. El atributo `deptno` del elemento `<emp:departamento>` no pertenece a ningún espacio de nombres.

```
<emp:empleado xmlns:emp="empresa:espacios:emp">
    <emp:datosPersonales trabajo:empno="7583"
                          xmlns:trabajo="empresa:espacios:trabajo">
        <emp:departamento deptno="10">Ventas</emp:departamento>
        <emp:nombre>Roberto</emp:nombre>
        <emp:apellido>López</emp:apellido>
    </emp:datosPersonales>
</emp:empleado>
```

**Actividad 4.1:**

Indica a qué espacio de nombres pertenece el elemento <info:venta> del siguiente documento XML y justifica tu respuesta.

```
<?xml version="1.0" ?>
<info:venta xmlns:info="empresa:espacios:info"
    xmlns:prod="empresa:espacios:prod">
    <prod:producto prod:id="p_987">Mesa de despacho</prod:producto>
    <prod:precio prod:moneda="Euro">300</prod:precio>
    <prod:unidades>200</prod:unidades>
</info:venta>
```

- a. A empresa:espacios:info
- b. A empresa:espacios:prod
- c. A empresa:espacios:info y empresa:espacios:prod
- d. A ningún espacio de nombres

Dos atributos con el mismo nombre pero con distinto prefijo de espacio de nombres son diferentes, lo que significa que pueden estar asociados al mismo elemento.

**Ejemplo:**

Los atributos trabajo:empno y emp:empno son diferentes, luego pueden aparecer asociados al mismo elemento.

```
<emp:empleado xmlns:emp="empresa:espacios:emp"
    xmlns:trabajo="empresa:espacios:trabajo">
    <emp:datosPersonales trabajo:empno="7583" emp:empno="e_7583">
        <emp:departamento deptno="10">Ventas</emp:departamento>
        <emp:nombre>Mercedes</emp:nombre>
        <emp:apellido>López</emp:apellido>
    </emp:datosPersonales>
</emp:empleado>
```

**Espacio de nombres por defecto**

Un espacio de nombres por defecto es aquel en el que no se define un prefijo. El ámbito de aplicación de ese espacio de nombres es el del elemento en el que se ha declarado y sus elementos descendientes, pero no a sus atributos.

Si se tuviera que añadir un espacio de nombres con prefijo a un documento XML grande ya creado, habría que ir añadiendo el prefijo a los elementos y atributos, algo tedioso y tendente a producir errores. De esta manera, se puede evitar tener que escribir el prefijo.

El mayor inconveniente de esta medida, citado ya antes, es que el espacio de nombres por defecto sólo afecta al elemento en el que está declarado y a sus descendientes, no a los atributos.

**Ejemplo:**

El espacio de nombres por defecto, empresa:espacios:emp, no afectará al atributo empno del elemento <datosPersonales> ni al atributo deptno del elemento <departamento>.

```
<empleado xmlns="empresa:espacios:emp"
           xmlns:trabajo="empresa:espacios:trabajo">
    <datosPersonales trabajo:empno="7583" empno="e_7583">
        <departamento deptno="10">Ventas</departamento>
        <nombre>Mercedes</nombre>
        <apellido>López</apellido>
    </datosPersonales>
</empleado>
```

Se puede desasignar a un elemento de un espacio de nombres por defecto con la orden:

```
<nombre_elemento xmlns="">
```

**Actividad 4.2:**

Indica a qué espacio de nombres pertenece el atributo moneda del elemento <precio> del siguiente documento XML y justifica tu respuesta.

```
<?xml version="1.0" ?>
<venta xmlns="empresa:espacios:info"
       xmlns:venta="empresa:espacios:venta">
    <producto id="p_987">Mesa de despacho</producto>
    <precio moneda="Euro">300</precio>
    <unidades>200</unidades>
</venta>
```

- A empresa:espacios:info
- A empresa:espacios:venta
- A empresa:espacios:info y empresa:espacios:venta
- A ningún espacio de nombres

**Atributos especiales**

Hay algunos atributos especiales en XML:

- `xml:space`: le indica a la aplicación que usa el XML si los espacios en blanco del contenido textual de un elemento son significativos.

**Ejemplo:**

Se le indica al XML que los espacios se mantengan (*preserve*) o se eliminen (*default*):

```
<!ATTLIST quitaEspacios xml:space #FIXED "default">
<!ATTLIST dejaEspacios xml:space #FIXED "preserve">
```

- `xml:lang`: permite especificar el idioma en el que está escrito el contenido textual de todo un documento o de sus elementos individuales. Los posibles valores son:
  - o un código de dos letras (ISO 639).
 

Ejemplo: `en` para English (existen subcódigos, como `en-GB` y `en-US` para inglés británico y americano respectivamente).
  - o un identificador de idioma registrado por el IANA (Internet Assigned Numbers Authority – Autoridad de Números Asignados en Internet). Estos identificadores empiezan por el prefijo `i-` o `I-`.
  - o un identificador definido por el usuario. Estos identificadores empiezan por el prefijo `x-` o `X-`
- `xml:base`: permite definir una URI distinta a la del documento.

## Parser XML

Un analizador XML (llamado en inglés *parser*), es un procesador que lee un documento XML y determina la estructura y propiedades de los datos en él contenidos. Un analizador estándar lee el documento XML y genera el árbol jerárquico asociado, lo que permite ver los datos en un navegador o ser tratados por cualquier aplicación.

Si el analizador comprueba las reglas de buena formación y además valida el documento contra un DTD o esquema, se dice que se trata de un **analizador validador**. Estos analizadores también comprueban la semántica del documento e informan de los errores existentes.

Existen validadores XML en línea, como XML Validation (<http://www.xmlvalidation.com>).

## Editores XML

Existen multitud de herramientas relacionadas con el manejo de XML y sus tecnologías asociadas (DTD, XSD, XPATH, XSLT, XSL-FO...). Algunas son muy sofisticadas y, en general con licencias propietarias. Hay otras más sencillas, también de código abierto. Por citar algunas:

- **XMLSpy**, de Altova (<http://www.altova.com/es>). Es un editor de XML, pero también de XSLT, XPath, XQuery... Forma parte de una suite de herramientas (MissionKit) interrelacionadas entre sí, entre las que cabe destacar también StyleVision, cuyo uso principal es el diseño visual de hojas de transformaciones XSLT o XSL-FO y la generación de documentos de salida en formatos como PDF, HTML, RTF, PostScript... La fortaleza más importante de estos programas es la existencia de representaciones gráficas de los distintos documentos, que pueden ser directamente editadas.  
En el momento de la impresión de este libro, la versión más moderna es la Enterprise 2012 y la licencia es propietaria.
- **<oxygen/>**, de Syncro Soft (<http://www.oxygenxml.com>). Al igual que el producto de Altova, además de XML, es un editor de diferentes tecnologías como XSD, XSLT o XQuery. También las interfaces de desarrollo gráfico están muy cuidadas.

Existen versiones para diversas plataformas (Windows, Linux, Mac...). Está compuesto de dos módulos principales, XML Developer (el editor de XML en sí) y XML Author (generador de documentos en distintos formatos), que se pueden instalar también por separado. En el momento de la impresión de este libro, la versión más moderna es la 13.2 y la licencia es propietaria.

- **XML Copy Editor** (<http://xml-copy-editor.sourceforge.net>). Es software libre bajo licencia GNU GPL, siendo Gerald Schmidt el desarrollador principal. La última versión existente es la 1.2.0.6.  
Es un editor de XML, XSD, XSLT...  
Esta es la herramienta que se usará para el seguimiento de este libro. Si bien las otras dos, citadas previamente son de una excelente calidad e incluyen funcionalidades más avanzadas que XML Copy Editor, esta última es suficiente.
- **XMLPad Pro Edition**, de WMHelp (<http://www.wmhelp.com/download.htm>). Es una herramienta sencilla cuyo desarrollo parece interrumpido en el momento de la publicación de este libro, pero que incluye funcionalidades como comprobación de buen formato de documentos XML, validación frente a DTD o esquemas XML e inferencia de DTD o esquema XML a partir de un XML.
- **Otras aplicaciones:**  
[Exchanger XML Editor](http://www.exchangerxml.com) (<http://www.exchangerxml.com>)  
[Liquid XML Editor](http://www.liquid-technologies.com/Xml-Studio.aspx) (<http://www.liquid-technologies.com/Xml-Studio.aspx>)

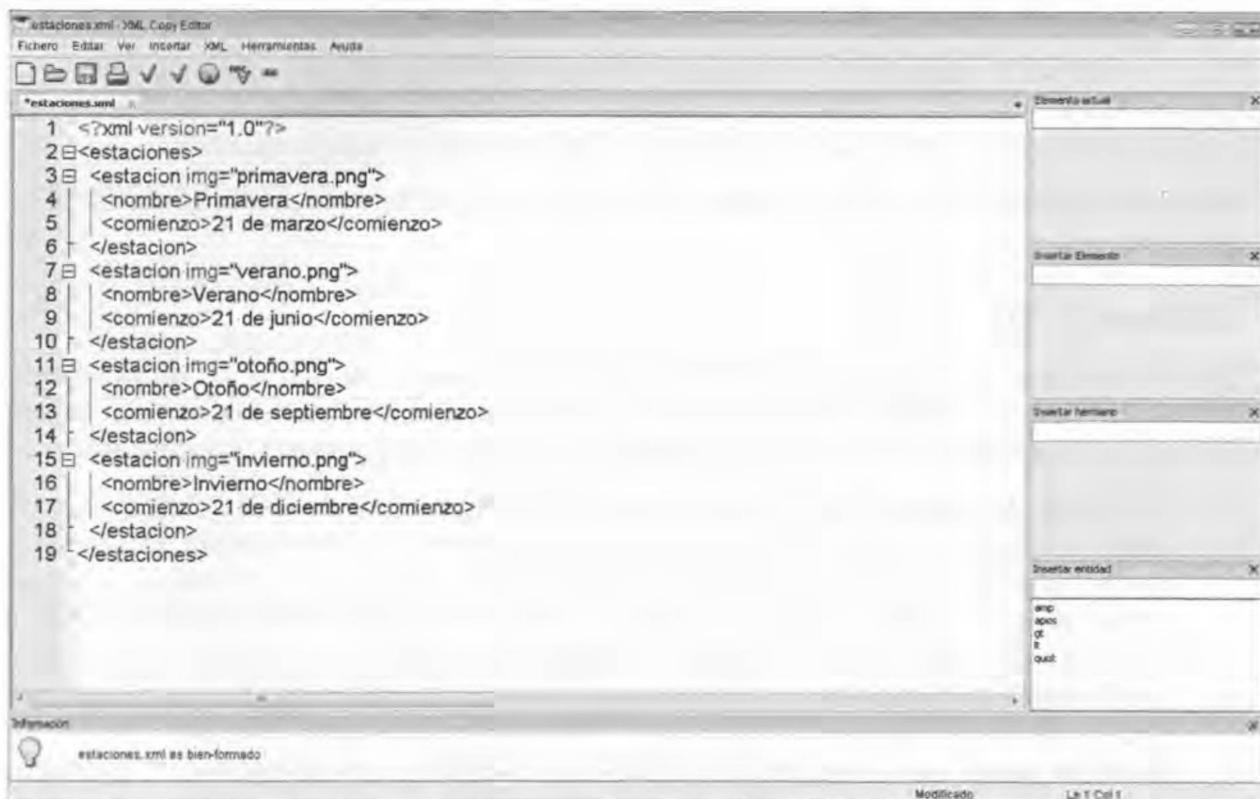


Figura 4.4: Interfaz del programa XML Copy Editor

## 4.5. Corrección sintáctica: documento XML bien formado

La especificación de XML define la sintaxis que el lenguaje debe seguir:

- Cómo se delimitan los elementos con etiquetas.
- Qué formato puede tener una etiqueta.
- Qué nombres son aceptables para los elementos.
- Dónde se colocan los atributos.

Un documento XML se dice que está **bien formado** si cumple las reglas establecidas por el W3C en las especificaciones para XML. Estas reglas son muy numerosas y, en ocasiones, complejas de entender, así que se citarán las más significativas

- El documento puede (el W3C lo recomienda) empezar por una instrucción de procesamiento `xml`, que indica la versión del XML y, opcionalmente, la codificación de caracteres (`encoding`), que por defecto es *UTF-8*, y si está listo para procesarse independientemente o requiere de otros archivos externos para dicha tarea (`standalone`), que por defecto vale *no*.

La instrucción de procesamiento correcta más simplificada es:

```
<?xml version = "1.0"?>
```

Otra más extensa e igualmente correcta:

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

Unos apuntes sobre juegos de caracteres y codificaciones:

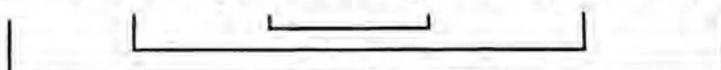
- . Un juego de caracteres es una colección de caracteres asociados cada uno a un número, llamado *punto de código*. Por ejemplo, en ASCII el punto de código de la A es 65.
- . Un ejemplo de juego de caracteres es Unicode (actualmente en su versión 6). Contiene todos los caracteres usados en todos los alfabetos existentes en el mundo.
- . Otro ejemplo de juego de caracteres es el ISO Latin 1. Una codificación de este juego de caracteres es la ISO-8859-1. Son 256 caracteres que coinciden en codificación con los 256 primeros de Unicode.
- . Una codificación (`encoding`) de caracteres determina cómo se representan en bytes los puntos de código. Por ejemplo, el punto de código de la A, el 65, se puede codificar como un byte con signo, dos bytes sin signo en formato little-endian, 4 bytes...
- . UTF-8 (Unicode Transformation Format de longitud variable de 8 bits) es un ejemplo de codificación de caracteres para el juego de caracteres Unicode.
- . El punto de código de la letra “A” en lenguas latinas es U+0041.
- . ASCII es un juego de caracteres y una codificación de caracteres. Unicode es un superconjunto de ASCII (lo contiene).
- Debe existir un único elemento raíz, que “cuelga” del nodo raíz (/). Este elemento tendrá como descendientes a todos los demás elementos. El documento XML bien formado más simple contendría sólo un elemento raíz, sin ningún descendiente.
- Los elementos que no sean vacíos deben tener una etiqueta de apertura y otra de cierre.
- Los elementos vacíos deber cerrarse con />. Ejemplo: <br />

- Los elementos deben aparecer correctamente anidados en cuanto a su apertura y su cierre, no solaparse. Es decir, los elementos se cierran en orden inverso al que se abren.

Ejemplo:

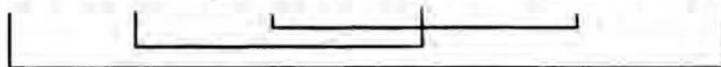
Se abre `<alpha>`, se abre `<beta>` y se abra `<gamma>`. Se cerrarán en orden inverso al de apertura. Primero se cierra `</gamma>`, luego `</beta>` y, por último, `</alpha>`.

```
<alfa> <beta> <qgamma>...</qgamma> </beta> </alfa> ✓
```



En este caso se ha cerrado  $\langle\beta\rangle$  antes que  $\langle\gamma\rangle$ .

```
<alfa> <beta> <qgamma>...</beta> </qgamma> </alfa> *
```



- Los nombres de elementos y atributos son sensibles a mayúsculas/minúsculas.
  - Los valores de los atributos deben aparecer entre comillas simples o dobles, pero del mismo tipo.

Ejemplo:

<libro signature="SIL001" /> ✓

<libro signature='SIL001' /> x

- No puede haber dos atributos con el mismo nombre asociados al mismo elemento.
  - No se pueden introducir ni instrucciones de procesamiento ni comentarios en ningún lugar del interior de las etiquetas de apertura y cierre de los elementos.

Ejemplo:

Un comentario incorrecto:

<persona <!-- comentario incorrecto --> >...</persona> **x**

- No puede haber nada antes de la instrucción de procesamiento <?xml ... ?>.
  - No puede haber texto antes ni después del elemento documento.
  - No pueden aparecer los signos < ni & en el contenido textual de elementos ni atributos.

Una manera de verificar que un documento XML está bien formado es abriéndolo en un navegador como Mozilla Firefox o Internet Explorer. Si se muestra el árbol de nodos, significa que está bien formado.

```

- <correo>
  <de>José Ramón</de>
  <para>JuanMa</para>
  <asunto>Partido de rugby en TV</asunto>
  - <mensaje>
    <saludo>Hola Juanma:</saludo>
    - <cuerpo>
      El próximo sábado ponen en la tele el Francia-Inglaterra del seis naciones. ¿te apetece que lo veamos juntos? Espero tus noticias.
    </cuerpo>
    <despedida>Un abrazo.</despedida>
  </mensaje>
</correo>

```

**Figura 4.5:** Visualización en un navegador de un documento XML

### Actividad 4.3:

Identifica cuáles de los siguientes documentos están bien formados y cuáles no. Cuando no lo estén, explica el (los) motivo(s). Para comprobar el resultado se puede usar un editor XML, como el XML Copy Editor, para que valide por nosotros los documentos.

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

```
<documento>Texto de prueba</documento>
```

```
<?xml?>
<documento>Texto de prueba</documento>
```

```
<?xml version="1.0"?>
<DOCUMENTO/>
```

```
<?XML version="1.0"?>
<Documento codigo="135">
  <nombre>Artículo</nombre>
  <amplitud>Media</amplitud>
</Documento>
```

```
<?xml version="1.0"?>
<El documento>
  <nombre>Artículo</nombre>
  <amplitud>Media</amplitud>
</El documento>
```

## 4.6. Documentos XML válidos

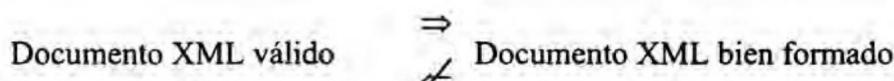
Hasta ahora se ha visto qué componentes forman un documento XML (elementos, atributos, comentarios...) y qué reglas deben de cumplir para que el documento sea bien formado, es decir, sea sintácticamente correcto.

Nada se ha dicho sobre qué elementos o atributos concretos pueden aparecer, en qué orden, qué valores pueden tomar, etc.

Si se hiciera esto, se estaría definiendo un lenguaje XML concreto, con su vocabulario (elementos y atributos permitidos), las relaciones entre elementos y atributos, los valores permitidos para ellos, etc.

Se dice que un documento XML es **válido** si existen unas reglas de validación (por ejemplo en forma de definición de tipo de documento (DTD)) asociadas a él, de manera que el documento deba cumplir con dichas reglas. Estas reglas especifican la estructura gramatical y sintaxis que debe tener el documento XML.

Todo documento XML válido está bien formado, pero no a la inversa.



### Ejemplo:

Para entender la diferencia entre un documento bien formado y un documento válido, veremos el equivalente con una frase en castellano.

montaña La nevada está  $\leftrightarrow$  La montaña está nevada

Ambas frases están **bien formadas** en castellano desde el punto de vista léxico, es decir, las palabras que en ellas aparecen son correctas en castellano. Ahora bien, desde el punto de vista gramatical, la primera se trata de una frase incorrecta (no se cumple la clásica regla de sujeto + predicado). Así, sólo la segunda frase es **válida** con respecto a la gramática castellana.

## 4.7. Validación de documentos XML con DTD

Un DTD (Document Type Definition – Definición de Tipo de Documento) es una descripción de la estructura de un documento XML. Se especifica qué elementos tienen que aparecer, en qué orden, cuáles son optativos, cuáles obligatorios, qué atributos tienen los elementos, etc.

Es un mecanismo de validación de documentos que existía antes de la aparición de XML. Se usaba para validar documentos SGML (Standard Generalized Markup Language – Lenguaje de Marcas eStándar Generalizado), que es el precursor de todos los lenguajes de marcas actuales. Cuando apareció XML, se integró en su especificación como modelo de validación gramatical.

En el momento de la publicación de este libro, la técnica de validación con DTDs ha quedado algo obsoleta, si bien se usó de manera intensiva en los años 1990 y principios de los 2000.

Actualmente, se usan más otras técnicas como los esquemas XML, también conocidos por las siglas XSD (XML Schema Document – Documento de Esquema XML). Sin embargo, muchos documentos han quedado validados únicamente mediante DTDs.

Ejemplo:

DTD estricto y DTD transicional para HTML 4.01

- <http://www.w3.org/TR/html4/sgml/dtd.html>
- <http://www.w3.org/TR/html4/sgml/loosedtd.html>

## Generación automática de DTDs

Cuando los documentos XML contienen muchos datos, los DTD que los validan pueden ser muy extensos y su escritura tediosa. Para facilitar este proceso, existen herramientas de generación automática de DTDs a partir de un documento XML, proceso conocido como inferencia. Se detallará su uso más adelante.

Estas herramientas realizan deducciones “gruesas” de las reglas gramaticales, que el desarrollador debe después afinar para que se ajusten a los requisitos semánticos.

- Generadores on-line:

[http://www.hitsw.com/xml\\_utilites/default.html](http://www.hitsw.com/xml_utilites/default.html)

- Generadores instalables:

- Trang (<http://www.thaiopensource.com/download>): es un programa desarrollado en Java, ejecutable desde la línea de comandos, que genera un DTD a partir de un XML dado.

- Editores XML que generan un DTD a partir de un documento XML. Prácticamente todos los citados anteriormente:

- XMLSpy de Altova
- <oXygen/>, de Syncro Soft
- XMLPad Pro Edition, de WMHelp

### 4.7.1. Estructura de un DTD. Elementos

Al igual que ya sucedía con las hojas de estilo en cascada (CSS), el DTD puede aparecer integrado en el propio documento XML, en un archivo independiente (habitualmente con extensión .dtd) e incluso puede tener una parte interna y otra externa. La opción del documento externo permitirá reutilizar el mismo DTD para distintos documentos XML, facilitando las posibles modificaciones de una manera centralizada.

Siempre que se quiera declarar un DTD, se hará al comienzo del documento XML. Las reglas que lo constituyen son las que podrán aparecer a continuación de la declaración (dentro del propio documento XML) o en un archivo independiente.

**Declaración del DTD:**

&lt;!DOCTYPE&gt;

Es la instrucción donde se indica qué DTD validará el XML. Aparece al comienzo del documento XML. El primer dato que aparece es el nombre del elemento raíz del documento XML.

En función del tipo de DTD la sintaxis varía. Las características que definen el tipo son:

- Ubicación: dónde se localizan las reglas del DTD.
  - o Interno: las reglas aparecen en el propio documento XML.
  - o Externo: las reglas aparecen en un archivo independiente.
  - o Mixto: mezcla de los anteriores, las reglas aparecen en ambos lugares. Las reglas internas tienen prioridad sobre las externas.
- Carácter: si es un DTD para uso privado o público.
  - o Para uso privado: se identifica por la palabra SYSTEM.
  - o Para uso público: se identifica por la palabra PUBLIC. Debe ir acompañado del FPI (Formal Public Identifier – Identificador Público Formal), una etiqueta que identifica al DTD de manera “universal”.

Las distintas combinaciones son:

Sintaxis	Tipo de DTD
<!DOCTYPE elemento_raíz [reglas]>	Interno (luego privado)
<!DOCTYPE elemento_raíz SYSTEM URL>	Externo y privado
<!DOCTYPE elemento_raíz SYSTEM URL [reglas]>	Mixto y privado
<!DOCTYPE elemento_raíz PUBLIC FPI URL>	Externo y público
<!DOCTYPE elemento_raíz PUBLIC FPI URL [reglas]>	Mixto y público

Tabla 4.1: Diferentes sintaxis para DOCTYPE

La combinación interno y público no tiene sentido, ya que el hecho de ser público fuerza que el DTD esté en un documento independiente.

**Atributos:**

- Nombre del elemento raíz del documento XML: aparece justo a continuación de la palabra DOCTYPE.
- Declaración de privacidad/publicidad: aparece a continuación del nombre del elemento raíz e indica si el DTD es de uso público (PUBLIC) o de uso interno de la organización que lo desarrolla (SYSTEM).
- Identificador: sólo existe cuando el DTD es PUBLIC e indica el FPI por el que se conoce el DTD.

### Estructura del FPI:

Está compuesto de 4 campos separados por el carácter //.

- Primer campo: norma formal o no formal
  - Si el DTD no ha sido aprobado por una norma formal, como por ejemplo un DTD escrito por uno mismo, se escribe un signo menos (-).
  - Si ha sido aprobado por un organismo no oficial, se escribe un signo más (+).
  - Si ha sido aprobado por un organismo oficial, se escribe directamente una referencia al estándar.
- Segundo campo: nombre del organismo responsable del estándar.
- Tercer campo: tipo del documento que se describe, suele incluir un número de versión.
- Cuarto campo: idioma del DTD.

### Ejemplo:

Se va a declarar un DTD respecto a un documento XML cuyo elemento raíz se llama html y que es de carácter público.

The diagram shows the following structure of a DOCTYPE declaration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- ① A sign minus (-) indicating it's not a formal standard.
- ② The name of the responsible organization is W3C.
- ③ The type of document is XHTML Transitional, version 1.0.
- ④ The language of the DTD is English (EN).

- ① Al aparecer un signo menos (-) significa que el DTD no ha sido aprobado por una norma formal.
  - ② El nombre del organismo responsable del DTD es el W3C.
  - ③ El tipo de documento es XHTML Transicional, en su versión 1.0.
  - ④ El idioma del DTD es el inglés (EN).
- url: sólo existe cuando el DTD (en parte o en su totalidad) se encuentra declarado en un archivo externo, del que da su ubicación. Como ya se ha comentado, puede darse el caso que un DTD esté definido en parte en un archivo externo y en parte en el documento XML.

### Ejemplo:

Se declara el tipo de un documento XML cuyo elemento raíz se llama `<planetas>`. Se trata de un DTD de uso privado (SYSTEM) y la ubicación de las reglas del DTD es externa, en un archivo llamado `planetas.dtd` que se encuentra en el mismo directorio que el archivo XML.

```
<!DOCTYPE planetas SYSTEM "planetas.dtd">
```

### Ejemplo:

Un DTD mixto y privado tendrá sus reglas repartidas entre la cabecera del documento XML y un archivo externo.

El documento XML, `charlas.xml`, será:

```

<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE charlas SYSTEM "charlas.dtd" [
  <!ELEMENT charlas (charla)*>
  <!ELEMENT charla (nombre, lugar, despedida)>
  <!ENTITY despedidaInglesa "Thank you, good bye!">
  <!ENTITY despedidaFrancesa "Merci, au revoir!">
]>
<charlas>
  ...
</charlas>

```

El archivo de reglas externo, *charlas.dtd*, será:

```

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT lugar (#PCDATA)>
<!ELEMENT despedida (#PCDATA)>

```

Nótese que el archivo externo no incluye la declaración del tipo de documento (DOCTYPE), puesto que sólo aparece en la cabecera del documento XML.

Conviene resaltar también que el atributo *standalone* de la instrucción de procesamiento `<?xml ... ?>` tiene el valor *no*, lo que significa que para el correcto procesado del documento necesitamos del uso de otros documentos externos (en este caso, de *charlas.dtd*).

### Componentes del DTD:

Hay cuatro tipos posibles de componentes que se pueden declarar en un DTD:

- Elemento
- Atributo
- Entidad
- Notación

#### **<!ELEMENT>**

Es una declaración de tipo de elemento. Indica la existencia de un elemento en el documento XML.

Sintaxis general: `<!ELEMENT nombre_elemento modelo_contenido>`

El nombre del elemento tiene que ser el mismo que el correspondiente del documento XML, sin los signos de menor (`<`) y mayor (`>`) propios de las etiquetas.

El modelo de contenido puede indicar:

- Una regla, en cuyo caso será:
  - ANY (cualquier cosa): se puede utilizar al construir el DTD para dejar la descripción de un elemento como válida en cualquier caso, eliminando cualquier comprobación sintáctica. Es un comodín que no debe aparecer en el DTD definitivo.

- EMPTY (elemento vacío): describe un elemento sin descendientes.

Ejemplo:

La descripción del elemento `<br />` de HTML será:

```
<!ELEMENT br EMPTY>
```

- Datos (caracteres), sean textuales, numéricos o cualquier otro formato que no contenga marcas (etiquetas). Se describe como #PCDATA (Parsed Character Data – Datos de Caracteres Procesados) y debe aparecer entre paréntesis.

Ejemplo:

Una regla de un DTD puede ser:

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

Que se corresponde con un elemento llamado `<titulo>` con contenido textual.

```
<titulo>Lenguajes de marcas</titulo>
```

- Elementos descendientes. Su descripción debe aparecer entre paréntesis y se basa en las siguientes reglas:

Cardinalidad de los elementos:

Para indicar el número de veces que puede aparecer un elemento o una secuencia de elementos existen ciertos símbolos:

Símbolo	Significado
?	El elemento (o secuencia de elementos) puede aparecer 0 o 1 vez
*	El elemento (o secuencia de elementos) puede aparecer de 0 a N veces
+	El elemento (o secuencia de elementos) puede aparecer de 1 a N veces

Secuencias de elementos:

En las secuencias de elementos se utilizan símbolos para indicar el orden en que un elemento debe aparecer, o bien si aparece como alternativa a otro:

Símbolo	Significado
A, B	El elemento B aparecerá a continuación del elemento A
A   B	Aparecerá el elemento A o el B, pero no ambos

Se pueden combinar el uso de los símbolos de cardinalidad de los elementos con los de la secuencias de elementos.

Ejemplo:

En un correo electrónico, se podría describir el elemento raíz `<email>` como una secuencia de elementos `<para>`, `<cc>` (optativo), `<cco>` (optativo), `<asunto>` y `<cuerpo>`. Las reglas que representan esto serán:

```
<!ELEMENT email (para, cc?, cco?, asunto, cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT cc (#PCDATA)>
...

```

**Ejemplo:**

Un contrato tiene una lista de cláusulas. Cada una de las cláusulas está compuesta de varios epígrafes y sus desarrollos asociados, y concluyen por un único epílogo:

```
<!ELEMENT clausulas (clausula)+>
<!ELEMENT clausula ((epígrafe, desarrollo)+, epilogo)>
...

```

- Contenido mixto, mezcla de texto más elementos descendientes.

**Ejemplo:**

Se quiere describir un elemento `<párrafo>` que simule el párrafo de un editor de textos, de forma que pueda contener texto sin formato, texto en `<negrita>` (rodeado de esta etiqueta) o texto en `<cursiva>` (rodeado de esta etiqueta). Estas dos últimas etiquetas podrán a su vez contener, bien texto sin formato, bien la otra etiqueta. Por tanto, `<párrafo>`, `<negrita>` y `<cursiva>` son elementos de contenido mixto. Las reglas que describen esto serán:

```
<!ELEMENT parrafo (negrita | cursiva | #PCDATA)*>
<!ELEMENT negrita (cursiva | #PCDATA)*>
<!ELEMENT cursiva (negrita | #PCDATA)*>
```

**!** NOTA: No se puede usar el cuantificador `+` con un contenido mixto, por eso se ha usado el `*`.

Un documento XML válido con respecto a las reglas del anterior DTD:

```
<parrafo>
  Aquí un <cursiva>tema <negrita>importante</negrita></cursiva>
</parrafo>
```

**Actividad 4.4:**

Se quiere que el elemento `<grupoSanguineo>` tenga como único elemento descendiente a uno solo de los cuatro siguientes A, B, AB ó O. Indica cuál de las siguientes es una declaración correcta del citado elemento.

- `<!ELEMENT grupoSanguineo (A ? B ? AB ? O) >`
- `<!ELEMENT grupoSanguineo (A , B , AB , O) >`
- `<!ELEMENT grupoSanguineo (A | B | AB | O) >`
- `<!ELEMENT grupoSanguineo (A + B + AB + O) >`

---

**Actividad 4.5:**

¿Cuál de las siguientes afirmaciones es correcta respecto a la declaración del elemento?

a. `<!ELEMENT contenido (alfa | beta*) >`

Tanto el elemento alfa como el elemento beta pueden aparecer 0 o más veces como descendientes del elemento contenido.

b. `<!ELEMENT contenido (alfa , beta) >`

Tanto el elemento alfa como el elemento beta pueden aparecer una vez como descendientes del elemento contenido, sea cual sea el orden.

c. `<!ELEMENT contenido (alfa | beta) >`

El elemento alfa y el elemento beta deben aparecer una vez cada uno como descendientes del elemento contenido.

d. `<!ELEMENT contenido (alfa , beta*) >`

El elemento alfa debe aparecer una vez y a continuación el elemento beta debe aparecer 0 o más veces, ambos como descendientes del elemento contenido.

---

**! NOTA:** Los documentos HTML no necesitan estar bien formados para poderse visualizar en los navegadores. Por eso, en los DTD que validan estos documentos se usa una determinada nomenclatura para indicar que un elemento pueda ser opcionalmente abierto o cerrado. Así, si aparecen dos guiones después del nombre del elemento tanto la etiqueta inicial como la final son obligatorias. Un guion seguido por la letra "O" indica que puede omitirse la etiqueta final. Un par de letras "O" indican que tanto la etiqueta inicial como la final pueden omitirse.

### **<!ATTLIST>**

Es una declaración de tipo de atributo. Indica la existencia de atributos de un elemento en el documento XML. En general se utiliza un solo ATTLIST para declarar todos los atributos de un elemento (aunque se podría usar un ATTLIST para cada atributo).

Sintaxis general: `<!ATTLIST nombre_elemento`

```
    nombre_atributo tipo_atributo caracter
    nombre_atributo tipo_atributo caracter
```

`>`

El **nombre** del atributo tiene que ser un nombre XML válido.

El **carácter** del atributo puede ser:

- un valor textual entre comillas, que representa un valor por defecto para el atributo.
- `#IMPLIED`, el atributo es de carácter opcional y no se le asigna ningún valor por defecto.

- #REQUIRED, el atributo es de carácter obligatorio, pero no se le asigna un valor por defecto.
- #FIXED, el atributo es de carácter obligatorio y se le asigna un valor por defecto que además es el único valor que puede tener el atributo.

Los posibles **tipos de atributo** son:

- CDATA: caracteres que no contienen etiquetas
- ENTITY: el nombre de una entidad (que debe declararse en el DTD)
- ENTITIES: una lista de nombres de entidades (que deben declararse en el DTD), separadas por espacios
- Enumerado: una lista de valores de entre los cuales, el atributo debe tomar uno

Ejemplo:

Se quiere definir una regla que valide la existencia de un elemento <semaforo>, de contenido vacío, con un único atributo color cuyos posibles valores sean rojo, naranja y verde. El valor por defecto será verde.

```
<!ELEMENT semaforo EMPTY>
<!ATTLIST semaforo
          color (rojo | naranja | amarillo) "verde">
```

- ID: un identificador único. Se usa para identificar elementos, es decir, caracterizarlos de manera única. Por ello, dos elementos no pueden tener el mismo valor en atributos de tipo ID. Además, un elemento puede tener a lo sumo un atributo de tipo ID. El valor asignado a un atributo de este tipo debe ser un nombre XML válido.
- IDREF: representa el valor de un atributo ID de otro elemento, es decir, para que sea válido, debe existir otro elemento en el documento XML que tenga un atributo de tipo ID y cuyo valor sea el mismo que el del atributo de tipo IDREF del primer elemento.

Ejemplo:

Se quiere representar un elemento <empleado> que tenga dos atributos, idEmpleado e idEmpleadoJefe. El primero será de tipo ID y carácter obligatorio, y el segundo de tipo IDREF y carácter optativo.

```
<!ELEMENT semaforo (nombre, apellido)>
<!ATTLIST empleado
          idEmpleado ID #REQUIRED
          idEmpleadoJefe IDREF #IMPLIED>
***
```

Aquí tenemos un fragmento de XML válido con respecto a las reglas recién definidas. Cada elemento <empleado> tiene un atributo idEmpleado, con un valor válido (nombre XML válido) y el segundo elemento <empleado> tiene también un atributo idEmpleadoJefe, cuyo valor ha de ser el mismo que el del atributo de tipo ID de otro

elemento existente en el documento. En este caso, este atributo `idEmpleJefe` del segundo `<empleado>` vale igual que el atributo `idEmpleado` del primer `<empleado>`:

```
<empleados>
    <empleado idEmpleado="e_111">...</empleado>
    <empleado idEmpleado="e_222" idEmpleadoJefe="e_111"> ✓
    ...
</empleado>
</empleados>
```

Un fragmento de XML no válido con respecto a las reglas recién definidas sería aquél en el cual el atributo `idEmpleadoJefe` del primer `<empleado>` tenga un valor que no exista para ningún atributo de tipo ID de otro elemento del documento:

```
<empleados>
    <empleado idEmpleado="e_111" idEmpleadoJefe="e_333"> ✗
    ...
</empleado>
    <empleado idEmpleado="e_222" idEmpleadoJefe="e_111">
    ...
</empleado>
</empleados>
```

- **IDREFS:** representa múltiples IDs de otros elementos, separados por espacios.
- **NMTOKEN:** cualquier nombre sin espacios en blanco en su interior. Los espacios en blanco anteriores o posteriores se ignorarán.

#### Ejemplo:

Se quiere declarar un atributo de tipo NMTOKEN de carácter obligatorio.

En el siguiente DTD, la declaración del elemento `<rio>` y su atributo `pais` es:

```
<!ELEMENT rio (nombre)>
<!ATTLIST rio pais NMTOKEN #REQUIRED>
```

En el siguiente fragmento XML, el valor del atributo `pais` es válido con respecto a la regla anteriormente definida:

```
<rio pais="EEUU">
    <nombre>Misisipi</nombre>
</rio>
```



En el siguiente documento XML, el valor del atributo `pais` no es válido con respecto a la regla anteriormente definida por contener espacios en su interior:

```
<rio pais="Estados Unidos"> ✗
    <nombre>Misisipi</nombre>
</rio>
```



- NMOKENS: una lista de nombres, sin espacios en blanco en su interior (los espacios en blanco anteriores o posteriores se ignorarán), separados por espacios.
- NOTATION: un nombre de notación, que debe estar declarada en el DTD.

### <!ENTITY>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración de tipo de entidad. Hay diferentes tipos de entidades y, en función del tipo, la sintaxis varía:

- Referencia a entidades generales (internas o externas).
- Referencia a entidades parámetro (internas o externas).
- Entidades no procesadas (*unparsed*).
- Referencia a entidades generales, se utilizarán dentro del documento XML.

Sintaxis general: <! ENTITY nombre\_entidad definición\_entidad>

#### Ejemplo:

En primer lugar, se declara una entidad en el DTD

```
<!ENTITY rsa "República Sudafricana">
```

A continuación, se usa en el XML anteponiendo al nombre de la entidad el carácter ampersand (&) y a continuación un carácter punto y coma (;). El programa analizador del documento realizará la sustitución.

```
<país>
  <nombre>&rsa;</nombre>
  ...
</país>
```

#### Ejemplo:

En el DTD de HTML se declaran diversas entidades para referenciar caracteres con acentos diversos, como &eacute; para referenciar el carácter é.

- Referencia entidades generales externas, ubicadas en otros archivos:

Sintaxis general: <! ENTITY nombre\_entidad tipo\_uso url\_archivo>

Siendo el tipo de uso privado (SYSTEM) o público (PUBLIC).

#### Ejemplo:

Se dispone de un archivo de texto, *autores.txt*, que contiene el siguiente texto plano “Juan Manuel y José Ramón”.

Se crea un documento XML que hace referencia a ese archivo de texto, en forma de entidad externa. Al visualizar el documento, la referencia a la entidad general externa se sustituirá por el texto contenido en el archivo.

```
<?xml version="1.0"?>
<!DOCTYPE escritores [
    <!ELEMENT escritores (#PCDATA)>
    <!ENTITY autores SYSTEM "autores.txt">
]>
<escritores>&autores;</escritores>
```

- Referencia a entidades parámetro, que se usarán en el propio DTD y funcionan cuando la definición de las reglas del DTD se realiza en un archivo externo.

Sintaxis general: `<!ENTITY % nombre_entidad definición_entidad>`

Ejemplo:

Se declara una entidad parámetro dimensiones y se referencia dentro del propio DTD.

```
<!ENTITY % dimensiones "alto CDATA #IMPLIED ancho CDATA #IMPLIED
profundo CDATA #IMPLIED">
<!ELEMENT objeto (nombre)>
<!ATTLIST objeto
    codigo ID #REQUIRED
    %dimensiones;>
...
```

Este código es equivalente a haber escrito:

```
<!ELEMENT objeto (nombre)>
<!ATTLIST objeto
    nombre CDATA #REQUIRED
    alto CDATA #IMPLIED
    ancho CDATA #IMPLIED
    profundo CDATA #IMPLIED>
...
```

- Referencia entidades parámetro externas, ubicadas en otros archivos:

Sintaxis general: `<!ENTITY % nombre_entidad tipo_uso fpi url_archivo>`

Siendo el tipo de uso privado (SYSTEM) o público (PUBLIC). Si es PUBLIC, es necesario definir el FPI que, recordemos, es el nombre por el cual se identifica públicamente un determinado elemento (sea un DOCTYPE, un ELEMENT o una ENTITY), en este caso la entidad.

- Entidades no procesadas, referencian a datos que no deben ser procesados por el analizador XML, sino por la aplicación que lo use (como una imagen).

Sintaxis general: `<!ENTITY % nombre_entidad tipo_uso fpi valor_entidad NDATA tipo>`

Ejemplo:

Se declara una notación (se comentará más adelante) de nombre JPG para el tipo MIME (Multipurpose Internet Mail Extensions - Extensiones Multipropósito de Correo de Internet). Se declara una entidad no procesada de nombre mediterraneo, asociada al archivo de

imagen *mediterraneo.jpg*. Por último, se declara un elemento *<mar>*, que cuenta con atributo *imagen* que es del tipo ENTITY recién declarado.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg" NDATA JPG>
<!ELEMENT mar ... >
<!ATTLIST mar
    imagen ENTITY #IMPLIED>
```

Y en el XML, el valor del atributo *imagen* del elemento *<mar>* es la entidad no procesada declarada en el DTD:

```
<mares>
    <mar imagen="mediterraneo">
        <nombre>Mediterráneo</nombre>
    </mar>
    ...
</mares>
```

**! NOTA:** En ocasiones se utiliza otra tecnología, llamada XLink, en sustitución de las entidades no procesadas.

#### Ejemplo:

Una extensión del ejemplo anterior sería el permitir incluir múltiples imágenes como valor del atributo *imagen* del elemento *<mar>*.

```
<!NOTATION JPG SYSTEM "image/jpeg">
<!ENTITY mediterraneo1 SYSTEM "mediterraneo1.jpg" NDATA JPG>
<!ENTITY mediterraneo2 SYSTEM "mediterraneo2.jpg" NDATA JPG>
<!ELEMENT mar ... >
<!ATTLIST mar
    imagen ENTITIES #IMPLIED>
```

Y en el documento XML, el valor del atributo *imagen* del elemento *<mar>* son las dos entidades no procesadas declaradas en el DTD:

```
<mares>
    <mar imagen="mediterraneo1 mediterraneo2">
        <nombre>Mediterráneo</nombre>
    </mar>
    ...
</mares>
```

#### <!NOTATION>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración del tipo de atributo NOTATION. Una notación se usa para especificar un formato de datos que no sea XML. Se usa con frecuencia para describir tipos MIME, como *image/gif* o *image/jpg*.

Se utiliza para indicar un tipo de atributo al que se le permite usar un valor que haya sido declarado como notación en el DTD.

Sintaxis general de la notación:

```
<!NOTATION nombre_notación SYSTEM "identificador_externo">
```

Sintaxis general del atributo que la usa:

```
<!ATTLIST nombre_elemento
            nombre_atributo NOTATION valor_defecto>
```

Ejemplo:

Se declaran tres notaciones que corresponden a otros tantos tipos MIME de imágenes (gif, jpg y png). También se declara un elemento <mar> y sus atributos `imagen` y `formato_imagen`, este último referenciando a las notaciones recién creadas. Por último, se declara una entidad no procesada que se asocia a un archivo de imagen.

```
<!NOTATION GIF SYSTEM "image/gif">
<!NOTATION JPG SYSTEM "image/jpeg">
<!NOTATION PNG SYSTEM "image/png">
<!ELEMENT mar ... >
<!ATTLIST mar
            imagen ENTITY #IMPLIED
            formato_imagen NOTATION (GIF | JPG | PNG) #IMPLIED>
<!ENTITY mediterraneo SYSTEM "mediterraneo.jpg">
```

Y en el documento XML, el valor del atributo `imagen` del elemento <mar> es la entidad no procesada declarada en el DTD, y el valor del atributo `formato_imagen` el de una de sus alternativas válidas, la notación JPG:

```
<mares>
  <mar imagen="mediterraneo" formato_imagen="JPG">
    <nombre>Mediterráneo</nombre>
  </mar>
  ...
</mares>
```

## Secciones condicionales

Permiten incluir o excluir reglas en un DTD en función de condiciones. Sólo se pueden ubicar en DTDs externos. Su uso tiene sentido al combinarlas con referencias a entidades parámetro. Las secciones condicionales son `IGNORE` e `INCLUDE`, teniendo la primera precedencia sobre la segunda.

Ejemplo:

Supónganse dos estructuras diferentes para un mensaje:

- una sencilla que incluye emisor, receptor y contenido
- otra extendida que incluye los datos de la estructura sencilla más el título y el número de palabras.

Se quiere diseñar un DTD que, en función del valor de una entidad parámetro, incluya una estructura de mensaje o la otra. Por defecto se incluirá la larga.

El DTD, que ha de ser externo, tendrá la siguiente estructura:

```

<!-- Mensaje corto -->
<! [IGNORE[
    <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo -->
<! [INCLUDE[
    <!ELEMENT mensaje (titulo, emisor, receptor, contenido, palabras)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT palabras (#PCDATA)>
]]>

<!-- Declaración de elementos y atributos comunes -->
<!ELEMENT emisor (#PCDATA)>
<!ELEMENT receptor (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>

```

Al añadir las referencias a entidad parámetro, el DTD quedaría:

```

<!ENTITY %corto "IGNORE">
<!ENTITY %largo "INCLUDE">

<!-- Mensaje corto -->
<! [%corto[
    <!ELEMENT mensaje (emisor, receptor, contenido)>
]]>

<!-- Mensaje largo -->
<! [%largo[
    <!ELEMENT mensaje (titulo, emisor, receptor, contenido, palabras)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT palabras (#PCDATA)>
]]>
<!-- Declaración de elementos y atributos comunes -->
<!ELEMENT emisor (#PCDATA)>
<!ELEMENT receptor (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>

```

Para cambiar el tipo de mensaje que se va a incluir, basta con modificar la asociación de valores de %corto a INCLUDE y de %largo a IGNORE, de la forma:

```

<!ENTITY %corto "INCLUDE">
<!ENTITY %largo "IGNORE">

```

Se podrían dejar la declaración de las entidades en la DTD interna al documento XML, donde se determinaría qué tipo de mensaje se quiere incluir de manera específica para ese documento:

```

<?xml verion="1.0"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd" [
    <!ENTITY %corto "INCLUDE">
    <!ENTITY %largo "IGNORE">
] >

```

---

```
<mensaje>
  ...
</mensaje>
```

---

#### 4.7.2. Poniendo todo junto

Se quiere construir un DTD que valide el siguiente documento XML, *persona.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE persona SYSTEM "persona.dtd"]>
<persona dni="12345678-L" estadoCivil="Casado">
  <nombre>María Pilar</nombre>
  <apellido>Sánchez</apellido>
  <edad>60</edad>
  <enActivo/>
</persona>
```

El esquema XML asociado se quiere que cumpla ciertas restricciones semánticas:

- El atributo *dni* es un identificador obligatorio.
- El estado civil puede ser: *Soltero*, *Casado* o *Divorciado*. Por defecto es *Soltero*.
- El elemento *<enActivo>* es optativo.

El DTD que cumple con esto, ubicado en el archivo *persona.dtd*, es:

```
<!ELEMENT persona (nombre, apellido, edad, enActivo?)>
<!ATTLIST persona dni ID #REQUIRED
            estadoCivil (Soltero | Casado | Divorciado) "Soltero">
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido (#PCDATA)>
<!ELEMENT edad (#PCDATA)>
<!ELEMENT enActivo (#PCDATA) EMPTY>
```

### Herramientas de generación automática de DTDs

Va a haber muchos documentos XML que puedan ser válidos por un mismo DTD. Imagínese simplemente un DTD que valide documentos XML que simulen un correo electrónico básico, es decir, que contengan elementos *<para>*, *<cc>* (optativo), *<ccos>* (optativo), *<asunto>* (optativo) y *<cuerpo>*.

Cualquier correo que cumpla con estas reglas será un documento válido con respecto a este DTD. Por ejemplo, igualmente válido sería un documento XML que contenga el elemento *<cc>* que el que no lo contenga, puesto que es optativo.

La cuestión aquí es que se va a tratar de inferir, a partir de un documento XML (que es un ejemplo concreto válido respecto a un DTD), el DTD genérico que lo valide.

Por ello, lo que se generará es un DTD que valide exactamente el XML que lo generó, pero si las reglas de negocio imponen restricciones adicionales, la herramienta de generación automática no será capaz de deducirlas, lo que nos obligará a añadirlas manualmente al DTD generado.

Entiéndase por **regla de negocio** a una restricción o característica propia del ámbito en el que se trabaja. Por ejemplo, una regla de negocio podría ser que un pago se haga en euros o dólares.

Ejemplo:

Se quiere generar un DTD a partir de un XML en el que hay un elemento <distancia>, el cual posee un atributo unidad, y se quiere que el valor de dicho atributo pueda ser *centímetro*, *metro* o *kilómetro*.

No hay manera de indicárselo al programa de generación automática. Éste sólo podrá suponer que existe un atributo unidad, perteneciente al elemento distancia, que contiene texto. Nada más. La enumeración de los posibles valores y, si se diera el caso, la existencia de uno de ellos por defecto, tendrá que indicarse “a mano” sobre el DTD generado.

A pesar de todo, son herramientas prácticas desde el punto de vista que realizan el trabajo mecánico inicial y, sobre el DTD que generan, resulta más cómodo introducir los ajustes necesarios para cumplir las restricciones semánticas (reglas de negocio).

**NOTA:** Para lograr una inferencia lo más precisa posible, se recomienda escribir un documento XML lo más completo posible, sin omitir elementos opcionales, ni atributos, etc.

## Limitaciones de los DTD

Algunas limitaciones de los DTD son:

1. Un DTD no es un documento XML, luego no se puede verificar si está bien formado.
2. No se pueden fijar restricciones sobre los valores de elementos y atributos, como su tipo de datos, su tamaño, etc.
3. No soporta espacios de nombres.
4. Sólo se puede enumerar los valores de atributos, no de elementos.
5. Sólo se puede dar un valor por defecto para atributos, no para elementos.
6. Existe un control limitado sobre las cardinalidades de los elementos, es decir, concretar el número de veces que pueden aparecer.

## 4.8. Validación de documentos XML con esquemas XML

Un esquema XML es un mecanismo para comprobar la validez de un documento XML. Se trata de una forma alternativa a los DTD, pero con ciertas ventajas sobre éstos:

- Es un documento XML, por lo que se puede comprobar si está bien formado.
- Existe un extenso catálogo de tipos de datos predefinidos para elementos y atributos que pueden ser ampliados o restringidos para crear nuevos tipos.
- Permiten concretar con precisión la cardinalidad de un elemento, es decir, las veces que puede aparecer en un documento XML.
- Permite mezclar distintos vocabularios (juegos de etiquetas) gracias a los espacios de nombres.

Como característica negativa, decir que los esquemas XML son más difíciles de interpretar por el ojo humano que los DTD.

En un esquema XML se describe lo que un documento XML puede contener: qué elementos, con qué atributos, de qué tipos de datos son tanto elementos como atributos, en qué orden aparecen los elementos, cuántas ocurrencias puede haber de cada elemento, etc.

Los documentos XML que se validan contra un esquema se llaman **instancias del esquema**. De hecho, se puede ver el esquema como un molde, y los documentos XML como objetos que tratan de ajustarse a ese molde. También se puede hacer una analogía con la programación orientada a objeto, los esquemas serían como las clases y los documentos XML como los objetos.

**NOTA:** Se pueden definir esquemas muy restrictivos o poco restrictivos. Los primeros fuerzan a los documentos XML validados a ser más precisos.

## Herramientas para validar esquemas

Todas las herramientas ya vistas que comprueban si un documento XML está bien formado son capaces de validar el documento frente a un esquema XML.

Por citar una herramienta de validación on-line:

- <http://www.corefiling.com/opensource/schemaValidate.html>

## Herramientas para generar esquemas automáticamente a partir de documentos XML

Sucede aquí lo mismo que con la generación automática de DTDs a partir de un documento XML: el generador no es capaz de conocer las reglas de negocio, sólo de crear un armazón básico a partir del documento XML.

- Herramientas de línea de comandos:
  - o Trang (<http://www.thaiopensource.com/relaxng/trang.html>): una herramienta de software libre instalable que permite inferir un esquema XML a partir de un documento XML. Igualmente, permite convertir esquemas XML en DTD y viceversa (así como entre otros mecanismos de validación de documentos XML, como RELAX NG).
- Editores de XML con esta funcionalidad:
  - o XMLSpy de Altova.
  - o <oXygen/> de
- Herramientas on-line:
  - o <http://www.flame-ware.com/products/xml-2-xsd>
  - o <http://www.freeformatter.com/xsd-generator.html>

#### 4.8.1. Estructura de un esquema XML. Componentes

Un esquema XML es un documento XML que ha de cumplir una serie de reglas:

- Su elemento raíz se llama `<schema>`.
- Su espacio de nombres debe ser `http://www.w3.org/2001/XMLSchema`. Se podría no definir un prefijo o usar uno de los más comunes: `xs` o `xsd`. En este libro se usará `xs`.

Así, el esquema XML más sencillo será:

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

- Puesto que un documento XML bien formado contiene al menos un elemento raíz, el esquema XML dispondrá de, al menos, un componente de declaración de elemento que defina el elemento raíz del documento XML. Este componente de declaración de elemento se identifica con el elemento `<xs:element>`, que tendrá un atributo `name` cuyo valor será el nombre del elemento raíz del documento XML.

Ejemplo:

Un documento XML muy simple:

```
<?xml version="1.0"?>
<simple>Es difícil escribir un documento más simple</simple>
```

Un posible esquema que lo valida:

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

**! NOTA:** Este esquema analizará como válido cualquier documento XML cuyo elemento raíz sea `<simple>`. De esta manera se ha obtenido un esquema XML completamente laxo en la validación de documentos XML. Cuantas más reglas y restricciones le añadamos, más restrictivo (y por tanto preciso) será.

- Puede haber un esquema con múltiples elementos raíz.

Ejemplo:

Un esquema con dos elementos raíz declarados, `<simple>` y `<complejo>`. Cualquier documento XML cuyo elemento raíz sea bien simple, bien complejo, se considerará conforme con este esquema.

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

- Un esquema es un conjunto de declaraciones de elementos y atributos y definición de tipos que sirve para fijar la estructura que deben tener sus documentos instancia XML. El orden en que se declaran los elementos, llamados componentes, en un esquema no es significativo ni afecta al funcionamiento del mismo.
- La vinculación de un esquema al documento XML se realiza en el documento XML, insertando `xmlns:xsi` y `xsi:noNamespaceSchemaLocation` como atributos del elemento raíz, en el caso de esquemas no asociados a espacios de nombres. Si el esquema estuviera asociado a un espacio de nombres se usaría el atributo `xsi:schemaLocation`. En este libro se usará sobre todo el primer caso.

Ejemplo:

En el documento XML se declara un espacio de nombres de prefijo `xsi`, propio de las instancias de esquemas XML, y se indica la ubicación del documento del esquema, en este caso de nombre `simple.xsd`.

```
<?xml version="1.0"?>
<simple xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="simple.xsd">
    ...
</simple>
```

## Atributos de los documentos instancia del esquema o documento XML

En la terminología de los esquemas, los documentos XML son instancias de los mismos, al fin y al cabo, son objetos salidos del molde que es el esquema.

Por eso, en los documentos XML aparece un espacio de nombres que tiene sentido cuando se trabaja con esquemas asociados. Se ha visto en el elemento raíz del documento XML del ejemplo anterior, la declaración del espacio de nombres `http://www.w3.org/2001/XMLSchema-instance` asociada al prefijo `xsi`.

A continuación, y en el mismo elemento raíz, se declara un atributo de ese espacio de nombres, `xsi:noNamespaceSchemaLocation="simple.xsd"`, que permite vincular el documento XML con un esquema de nombre `simple.xsd`.

Existen otros atributos de este espacio de nombres que se usan ocasionalmente en documentos XML (o instancias de esquema). Son:

- `xsi:nil`: asociado a un elemento indica que debe ser vacío de contenido. Puede valer `false` o `true`. Por defecto vale `false`, y para poderlo poner a `true`, se debe activar el atributo `nillable` en la definición del elemento en el esquema asociado. Se verá más adelante.

Ejemplo:

En el esquema se declararía un elemento de la forma:

```
<x:element name="fechaCompra" type="xs:date" nillable="true"/>
```

En el documento instancia XML se explicitaría que el contenido de `<fechaCompra>` debe ser vacío:

```
<fechaCompra xsi:nil="true"></fechaCompra>
```

- `xsi:type`: puede ser la definición de un tipo de un elemento, como `xs:string` o existente en un esquema asociado.
- `xsi:schemaLocation`: localización de un esquema con un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.
- `xsi:noNamespaceSchemaLocation`: localización de un esquema sin un espacio de nombres asociado. Se pueden indicar varios esquemas separados por espacios.

#### 4.8.2. Componentes básicos de un esquema

Se van a introducir los componentes que van a permitir construir un esquema XML que valide documentos XML. Como se ha comentado, a los documentos XML se les llama instancias del esquema. Los componentes imprescindibles son:

- `xs:schema`
- `xs:element`
- `xs:attribute`

**AVISO:** Se van a describir diferentes componentes que tienen atributos. Cuando el valor de un atributo pertenezca a una lista cerrada de posibles opciones, se indicarán explícitamente. Asimismo, aparecerá en negrita en valor por defecto.

##### **xs:schema**

Es el componente de declaración de esquema y elemento raíz de todo esquema XML.

###### Atributos optativos principales:

- `xmlns`: una referencia URI que indica uno o más espacios de nombres a usar en el esquema. Si no se indica ningún prefijo, los componentes del esquema del espacio de nombres pueden usarse de manera no cualificada.

###### Otros atributos optativos:

- `id`: identificador único para el elemento.
- `targetNamespace`: una referencia URI al espacio de nombres del esquema.
- `elementFormDefault`: formato de los nombres de los elementos en el espacio de nombres del esquema. Puede ser *unqualified*, que indica que los elementos no llevan prefijo, o *qualified*, indica que los elementos deben llevar el prefijo.

Potenciales valores: ***unqualified, qualified***

- `attributeFormDefault`: formato de los nombres de los atributos en el espacio de nombres del esquema. Funciona igual `elementFormDefault`.

Potenciales valores: ***unqualified, qualified***

- `version`: la versión del esquema.

Ejemplo:

Estructura de cualquier esquema:

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

**xs:element**

Es el componente de declaración de elemento y representa la existencia de un elemento en el documento XML.

Atributos optativos principales:

- **name:** indica el nombre del elemento. Es un atributo obligatorio si el elemento padre es `<xs:schema>`.
- **ref:** indica que la descripción del elemento se encuentra en otro lugar del esquema, es decir, referencia a la descripción del elemento. Este atributo no se puede usar si el elemento padre es `<xs:schema>`.
- **type:** indica el tipo del elemento.
- **default:** es el valor que tomará el elemento al ser procesado por alguna aplicación (habitualmente un analizador de XML o un navegador) cuando en el documento instancia XML no había recibido ningún valor. Sólo se puede usar si el contenido del elemento es únicamente textual.
- **fixed:** indica el único valor que puede contener el elemento en el documento instancia XML. Sólo se puede usar si el contenido del elemento es únicamente textual.
- **minOccurs:** indica el número mínimo de ocurrencias que del elemento puede haber en el documento XML. No se puede usar este atributo si el componente padre es `<xs:schema>`. Va desde 0 hasta ilimitado (*unbounded*).

Posibles valores: *0, 1, 2, ..., unbounded*.

- **maxOccurs:** indica el número máximo de ocurrencias que del elemento puede haber en el documento XML. No se puede usar este atributo si el componente padre es `<xs:schema>`. Va desde 0 hasta ilimitado (*unbounded*).

Posibles valores: *0, 1, 2, ..., unbounded*.

Otros atributos optativos:

- **id:** identificador único para el elemento.
- **form:** especifica el formato del nombre del atributo. Puede ser cualificado, es decir, con el espacio de nombres como prefijo del nombre, o no cualificado, sin el espacio de nombres.

Posibles valores: *unqualified* y *qualified*. El valor por defecto es el del atributo `elementFormDefault` del componente `<xs:schema>`.

- **substitutionGroup**: indica el nombre de otro elemento que puede ser sustituido por este elemento. Sólo se puede usar este atributo si el componente padre es `<xs:schema>`.
- **nillable**: especifica si puede aparecer el atributo de instancia `xsi:nil`, asociado al elemento en el documento instancia XML. Por defecto es falso, lo que significa que no se puede asignar a un elemento el atributo de instancia `xsi:nil`. Más adelante se comentarán los **atributos de instancia**.

Posibles valores: *false* y *true*.

- **abstract**: indica si el elemento puede ser usado en un documento XML instancia. Si vale cierto, indica que el elemento no puede aparecer en una instancia.

Posibles valores: *false* y *true*.

- **final**: indica si el elemento se puede derivar de alguna manera: por extensión o por restricción o ambas. Posibles valores: *extension*, *restriction* y *#all*.

#### Ejemplo:

Se quiere indicar que en el documento instancia XML aparecerá un elemento de nombre `<autor>`. La manera más genérica es:

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

El problema de esta descripción es que es demasiado general y no es útil, ya que no detalla si el elemento `<autor>` es de un tipo determinado, si tiene descendientes, contenido textual...

Para precisar más, indicaremos que el elemento `<autor>` es del tipo de datos predefinido `xs:string` (cadena de texto), deberá aparecer un mínimo de una vez y un máximo de tres.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="autor" type="xs:string" maxOccurs="3"/>
</xs:schema>
```

**AVISO:** Todos los ejemplos que se muestran deben aparecer como contenido del componente de declaración de esquema `<xs:schema>`, sin embargo, y por evitar repetición de código, se omitirá a partir de ahora. De igual forma, se omitirá la instrucción de procesamiento `<?xml ... ?>`.

#### Ejemplo:

Se le asigna un **valor por defecto**, *verde*, al elemento `<semaforo>`.

```
<xs:element name="semaforo" type="xs:string" default="verde"/>
```

Lo que representa esto es:

- a. Si no se le da ningún valor al elemento, la aplicación que procese el documento XML le asignará de manera automática el valor por defecto.

```
<semaforo></semaforo> ➔ <semaforo>verde</semaforo>
```

- b. Si se le da un valor al elemento, que puede ser distinto al valor por defecto, se quedará con ese valor:

```
<semaforo>amarillo</semaforo>
```

#### Ejemplo:

Se le asigna un **valor fijo**, *rojo*, al elemento `<semaforo>` y no se le podrá asignar ningún otro valor.

```
<xss:element name="semaforo" type="xss:string" fixed="rojo"/>
```

Lo que representa esto es:

- a. Si no se le da ningún valor al elemento, la aplicación que procese el documento XML le asignará de manera automática el valor fijo.

```
<semaforo></semaforo> → <semaforo>rojo</semaforo>
```

- b. Se le da un valor al elemento semáforo, que sólamente puede ser el valor fijo. Si se le asignara otro valor distinto, produciría un error de documento no válido:

```
<semaforo>rojo</semaforo> ✓
```

```
<semaforo>amarillo</semaforo> ✗
```

#### **xs:attribute**

Es el componente de declaración de atributo y representa la existencia de un atributo de un elemento en el documento XML.

#### Atributos optativos principales:

- **name:** nombre del atributo. Este atributo no puede aparecer simultáneamente que **ref**.
- **ref:** referencia a la descripción del atributo que se encuentra en otro lugar del esquema. Si aparece este atributo no aparecerán los atributos **type**, ni **form**, ni podrá contener un componente `<xss:simpleType>`.
- **type:** tipo del elemento.
- **use:** indica si la existencia del atributo es opcional, obligatoria o prohibida.

Posibles valores: *optional*, *required*, *prohibited*.

- **default:** valor que tomará el atributo al ser procesado por alguna aplicación (habitualmente un analizador de XML o un navegador) cuando en el documento XML no había recibido ningún valor. No puede aparecer simultáneamente con **fixed**.
- **fixed:** único valor que puede contener el atributo en el documento XML. No puede aparecer simultáneamente con **default**.

#### Otros atributos optativos:

- **id:** indica un identificador único para el atributo.

- **form**: especifica el formato del nombre del atributo. Puede ser cualificado, es decir, con el espacio de nombres como prefijo del nombre, o no cualificado, sin el espacio de nombres.

Posibles valores: *unqualified* y *qualified*. El valor por defecto es el del atributo **attributeFormDefault** del componente `<xs:schema>`.

#### Ejemplo:

Atributo de nombre **moneda**, tipo de datos textual y con **valor por defecto Euro**. Este valor por defecto se le asigna al atributo si no se le ha asignado otro o no aparece.

```
<xs:attribute name="moneda" type="xs:string" default="Euro"/>
```

#### Ejemplo:

Atributo de nombre **unidad**, tipo de datos textual y **valor fijo Minutos**.

```
<xs:attribute name="unidad" type="xs:string" fixed="Minutos"/>
```

#### Ejemplo:

Atributo de nombre **idEmple**, tipo de datos entero positivo y **existencia obligatoria**.

```
<xs:attribute name="idEmple" type="xs:positiveInteger"
use="required"/>
```

### 4.8.3. Tipos de datos

Se han visto ya en algunos ejemplos que en las declaraciones de elementos y atributos, a estos se les puede asignar un tipo de datos como `xs:string`, `xs:positiveInteger`...

En los esquemas XML, en la declaración de elementos y atributos se concreta el valor que puedan tomar mediante un tipo de datos. Los tipos de datos se organizan según diversos criterios. Una de las divisiones es en predefinidos y construidos:

- Los **predefinidos** son los que vienen integrados en la especificación de los esquemas XML.
- Los **construidos** son tipos generados por el usuario basándose en un tipo predefinido o en un tipo previamente construidos.

### Tipos de datos predefinidos

Existen 44 tipos predefinidos (del inglés *built-in types*, podría traducirse como “tipos que vienen de serie”). Se organizan en forma de relación jerárquica, a la manera de una jerarquía de clases en programación orientada a objeto, de forma que cada tipo será igual que su tipo padre más alguna particularidad que lo distinga.

Existe un tipo predefinido especial, `xs:anyType`, que se encuentra en la raíz de la jerarquía de tipos, y del que se derivan todos los demás tipos, tanto predefinidos como complejos. Este tipo es el más genérico de todos y de cualquier elemento se podría decir que es de ese tipo. Este tipo, en sí mismo, es un tipo complejo, por lo que no se podrían declarar atributos del mismo. A un elemento del cual no se indica tipo se le asociará el tipo `xs:anyType`.

Cuanto más se desciende en la jerarquía de tipos, más restrictivos (y por tanto precisos) van siendo los tipos. La recomendación al declarar elementos y atributos es asignarles el tipo que más se ajuste a lo que se quiere definir. Por ejemplo, si queremos determinar el tipo para un elemento edad, podríamos optar por `xs:anyType`, pero es una declaración demasiado genérica. En este caso sería más adecuado usar `xs:nonNegativeInteger`, que representa a todos los números enteros más el 0.

Existe **otro tipo predefinido especial**, `xs:anySimpleType`, que representa a cualquier tipo simple sin particularizar. Se puede usar para cualquier atributo, y también para elementos que sean de tipo simple pero, de nuevo, es demasiado genérico.

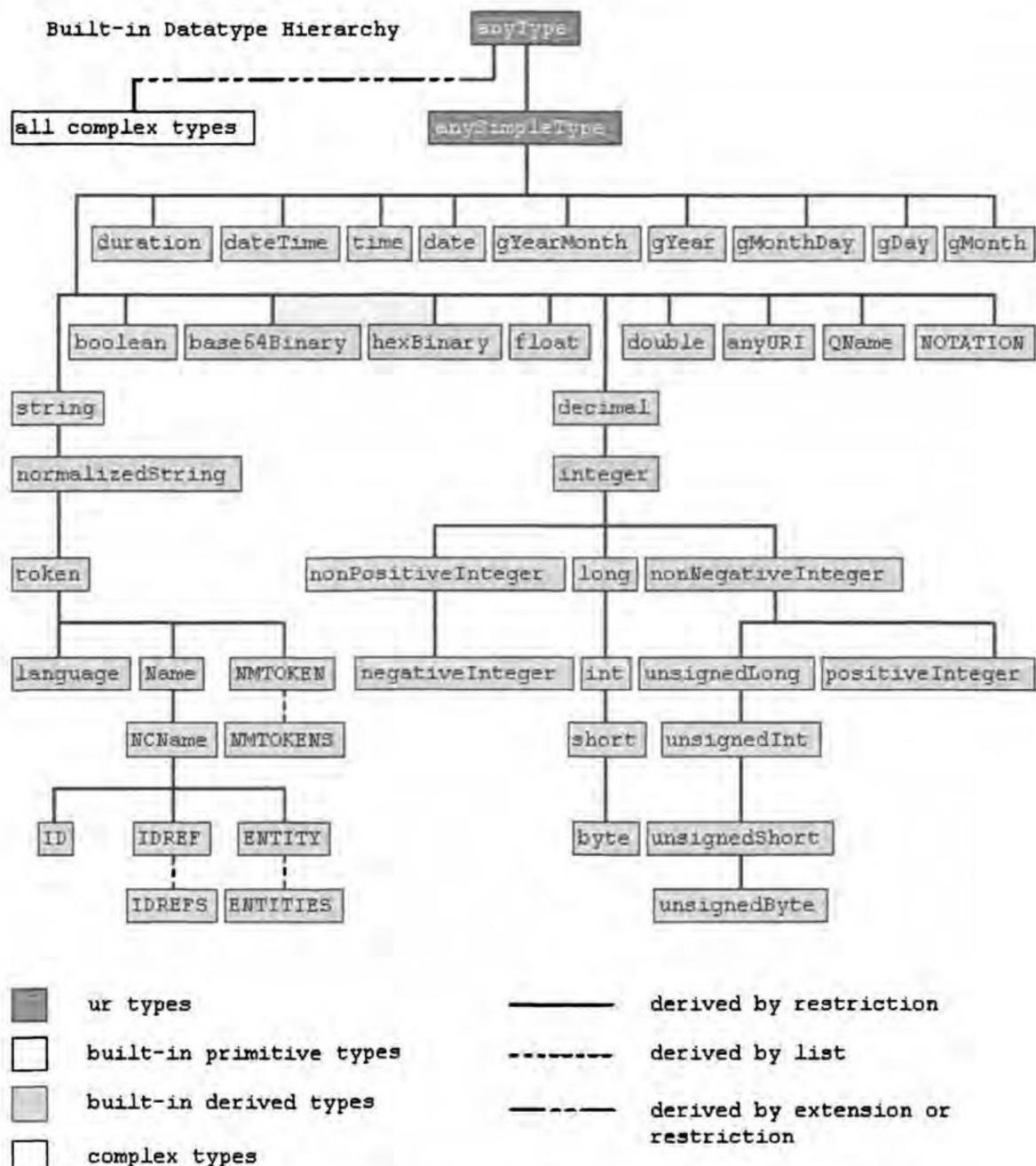


Figura 4.6: Tipos de datos predefinidos (primitivos, derivados y especiales)

Los tipos predefinidos se dividen también en **primitivos** y **no primitivos** (derivados de los primitivos).

- Los primitivos descienden directamente de `xs:anySimpleType`. Hay 19. Véase la *Figura 4.6*.
- Los no primitivos derivan de alguno de los primitivos. Hay 25. Véase la *Figura 4.6*.

Los tipos de datos predefinidos se agrupan en cinco categorías, en función de su contenido:

- Numéricos (hay 16)
- De fecha y hora (hay 9)
- De texto (hay 16)
- Binarios (hay 2)
- Booleanos (hay 1)

### Numéricos

Tipo de datos	Descripción
<code>float</code>	Número en punto flotante de precisión simple (32 bits)
<code>double</code>	Número en punto flotante de precisión doble (64 bits)
<code>decimal</code>	Números reales que pueden ser representados como $i \cdot 10^{-n}$
<code>integer</code>	Números enteros, de $-\infty$ a $+\infty$ ( $-\infty, -3, -2, -1, 0, 1, 2, 3, \dots, +\infty$ )
<code>nonPositiveInteger</code>	Números enteros negativos más el 0
<code>negativeInteger</code>	Números enteros menores que 0
<code>nonNegativeInteger</code>	Números enteros positivos más el 0
<code>positiveInteger</code>	Números enteros mayores que 0
<code>unsignedLong</code>	Números enteros positivos desde 0 hasta 18.446.744.073.709.551.615 (64 bits de representación → $2^{64}$ combinaciones)
<code>unsignedInt</code>	Números enteros positivos desde 0 hasta 4.294.967.295 (32 bits de representación → $2^{32}$ combinaciones)
<code>unsignedShort</code>	Números enteros positivos desde 0 hasta 65.535 (16 bits de representación → $2^{16}$ combinaciones)
<code>unsignedByte</code>	Números enteros positivos desde 0 hasta 255 (8 bits de representación → $2^8$ combinaciones)
<code>long</code>	Números enteros representados con 64 bits → $2^{64}$ combinaciones, desde -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807
<code>Int</code>	Números enteros representados con 32 bits → $2^{32}$ combinaciones, desde -2.147.483.648 hasta 2.147.483.647
<code>short</code>	Números enteros representados con 16 bits → $2^{16}$ combinaciones, desde -32.768 hasta 32.767

byte	Números enteros representados con 8 bits → $2^8$ combinaciones, desde -128 hasta 127
------	--------------------------------------------------------------------------------------

**Tabla 4.2:** Tipos de datos numéricosDe fecha y hora

Tipo de datos	Descripción
duration	Duración en años + meses + días + horas + minutos + segundos
dateTime	Fecha y hora, en formato aaaa-mm-dd T hh:mm:ss
date	Sólomente fecha en formato aaaa-mm-dd
time	Sólo la hora, en formato hh:mm:ss
gDay	Sólo el día, en formato -dd (la g viene de calendario Gregoriano)
gMonth	Sólo el mes (g → Gregoriano)
gYear	Sólo el año, en formato yyyy (g → Gregoriano)
gYearMonth	Sólo el año y el mes, en formato yyyy-mm (g → Gregoriano)
gMonthDay	Sólo el mes y el año, en formato -mm-dd (g → Gregoriano)

**Tabla 4.3:** Tipos de datos de fecha y hora

El tipo de datos dateTime representa un valor de fecha y hora según las especificaciones de la norma ISO 8601. Es, en sí mismo, la combinación de los otros dos tipos predefinidos time y date.

El patrón de este valor es;

$\underbrace{-?\d{4}-\d{2}\d{2}T\d{2}:\d{2}:\d{2}}_{\text{date}}$ 
 $\underbrace{(\.\d+)?(([+-]\d{2}:\d{2})|Z)}_{\text{time}}$ 
 $\underbrace{\text{común a date y time}}$

Patrón	Significado
-?	Un signo negativo (opcional). Para representar fechas anteriores al año 0
\d{4}-\d{2}\d{2}	La parte de la fecha (obligatoria). Equivale a yyyy-mm-dd
\d{2}:\d{2}:\d{2}	La parte de la hora (obligatoria). Equivale a hh:mm:ss
T	Indicador de hora local
(\.\d+)?	La parte decimal de la fracción de segundo (opcional)
(([+-]\d{2}:\d{2}) Z)?	La parte de la zona horaria (opcional) La Z permite expresar las horas en formato UTC (Universal Time Coordinated – Tiempo Universal Coordinado), anteriormente GMT (Greenwich Meridian Time – Hora del Meridiano de Greenwich)

**Tabla 4.4:** Estructura del tipo de datos xs:dateTime

Ejemplo:

Para representar una hora que sea las 3:40 PM del horario estándar de la costa este de Estados Unidos, que va 5 horas por detrás del UTC, se escribiría 15:40:00-05:00. Esa misma hora, pero de la zona central de Australia, que va 9 horas y media por delante del UTC, se escribiría como 15:40:00+09:30.

Ejemplo:

Aunque se verá más adelante, se introduce el concepto de **faceta**, que es una restricción que permite generar nuevos tipos de datos a partir de uno existente, limitando su rango de valores posibles.

Un tipo derivado de `xs:date` que representa la fecha en la que se ha entregado una determinada práctica, siendo el valor de la faceta `xs:maxInclusive` la fecha límite.

```
<!-- Se usa maxInclusive para indicar una límite final de fecha -->
<xs:simpleType name="TipoFechaEntregaPractica">
  <xs:restriction base="xs:date">
    <xs:maxInclusive value="2009-12-31"/>
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Un tipo derivado de `xs:time` que representa la hora en la que un pasajero ha embarcado en un determinado vuelo, siendo los valores de las facetas `xs:minInclusive` y `xs:maxInclusive` la hora de comienzo y fin del embarque.

```
<!-- Se usa minInclusive y maxInclusive para indicar límites
    iniciales y finales de horas -->
<xs:simpleType name="TipoHorarioEmbarque">
  <xs:restriction base="xs:time">
    <xs:minInclusive value="05:00:00"/>
    <xs:maxInclusive value="05:30:00"/>
  </xs:restriction>
</xs:simpleType>
```

De texto

Tipo de datos	Descripción
string	Cadenas de texto
normalizadaString	Cadenas de texto en las que se convierten los caracteres tabulador, nueva línea, y retorno de carro en espacios simples
token	Cadenas de texto sin los caracteres tabulador, ni nueva línea, ni retorno de carro, sin espacios por delante o por detrás, y con espacios simples en su interior
language	Valores válidos para <code>xml:lang</code> (según la especificación de XML)

NMTOKEN	Tipo de datos <b>para atributo</b> según XML 1.0, compatible con DTD
NMTOKENS	<b>Lista separada por espacios</b> de NMTOKEN, compatible con DTD
Name	Tipo de nombre según XML 1.0
QName	Nombre cualificado de espacio de nombres XML
NCName	QName sin el prefijo ni los dos puntos
anyURI	Cualquier URI
ID	Tipo de datos <b>para atributo</b> según XML 1.0, compatible con DTD. Han de ser valores únicos en el documento XML
IDREF	Tipo de datos <b>para atributo</b> según XML 1.0, compatible con DTD
IDREFS	<b>Lista separada por espacios</b> de IDREF, compatible con DTD
ENTITY	Tipo de datos <b>para atributo</b> en XML 1.0, compatible con DTD
ENTITIES	<b>Lista separada por espacios</b> de ENTITY, compatible con DTD
NOTATION	Tipo de datos <b>para atributo</b> en XML 1.0, compatible con DTD

Tabla 4.5: Tipos de datos textuales

Ejemplos:

Language	en-US
Name	horaSalida
QName	cliente:nombre
NCName	Nombre
anyURI	http://www.w3c.com
NMTOKENS	SP FR PR IT

Binarios

Tipo de datos	Descripción
hexBinary	Secuencia de dígitos hexadecimales (0..9,A,B,C,D,E,F)
base64Binary	Secuencia de dígitos en base 64

Tabla 4.6: Tipos de datos binarios

Booleano

Tipo de datos	Descripción
boolean	Puede tener 4 valores: 0, 1, true y false

Tabla 4.7: Tipos de datos booleanos

Todos los tipos de datos predefinidos:

- Pueden ser asignados tanto a elementos como a atributos.
- Pertenecen al espacio de nombres <http://www.w3.org/2001/XMLSchema>.

Ejemplo:

Se declararán tres elementos cuyos tipos de datos, supuestamente predefinidos, están incorrectamente declarados. Finalmente, se declara un elemento con un tipo de datos válido:

```
<xss:element name="elementoA" type="date"/> ①
<xss:element name="elementoB" type="w3c:date"/> ②
<xss:element name="elementoC" type="xsd:date"/> ③
<xss:element name="elementoD" type="xs:date"/> ④
```

- ① El tipo de datos date no se puede resolver como ninguno de los componentes de definición de tipos del esquema.
- ② El tipo de datos w3c:date tiene un prefijo, w3c, que no ha sido declarado.
- ③ El tipo de datos xsd:date se encuentra en el espacio de nombres <http://www.w3.org/2001/XMLSchema-datatypes>, pero los componentes existentes en este espacio de nombres no son referenciables desde el esquema. Se ha indicado un espacio de nombres erróneo.
- ④ El tipo de datos xs:date se encuentra en el espacio de nombres <http://www.w3.org/2001/XMLSchema>, que es el adecuado para los tipos de datos predefinidos.

#### 4.8.4. Tipos de datos simples vs. complejos

Aparece aquí una nueva catalogación de tipos de datos, en simples y complejos.

- **Tipos de datos simples:**

- Todos los tipos de datos predefinidos son simples.
- En general representan valores **atómicos** (el número 7, el texto "Hola"), no listas. Excepcionalmente, se construyen **no atómicos**, como una lista de números primos (1, 2, 3, 5, 7, 11...). Todos los tipos predefinidos son atómicos.
- Se pueden asignar tanto a elementos que sólo tengan contenido textual como a atributos.
- También se pueden construir, derivando de un tipo base predefinido al que se le introducen restricciones.

Ejemplo:

Un tipo de datos simple derivado es el que represente un correo electrónico. Se basa en el tipo de datos predefinido xs:string, que representa cualquier cadena de texto, pero fuerza a que cumpla una serie de reglas (contener el carácter "@", el carácter "...." ).

- **Tipos de datos complejos:**

- Sólo se pueden asignar a elementos que tengan elementos descendientes y/o atributos. Estos elementos pueden tener contenido textual.
- Por defecto, un elemento con un tipo de datos complejo contiene contenido complejo, lo que significa que tiene elementos descendientes.
- Un tipo de datos complejo se puede limitar a tener contenido simple, lo que significa que sólo contiene texto y atributos. Se diferencia de los tipos de datos simples precisamente en que tiene atributos.
- Se pueden limitar para que no tengan contenido, es decir, que sean vacíos, aunque pueden tener atributos.
- Pueden tener contenido mixto: combinación de contenido textual con elementos descendientes.

## Definición de tipos simples

Se usará el componente de definición de tipos simples, `<xs:simpleType>`. Se podrán limitar el rango de valores con el componente `<xs:restriction>`.

Se pueden asignar a elementos y a atributos.

### **xs:simpleType**

Es el componente de definición de tipos simples.

#### Atributos optativos principales:

- `name`: nombre del tipo. Este atributo es obligatorio si el componente es hijo de `<xs:schema>`, de lo contrario no está permitido.
- `id`: identificador único para el componente.

Hasta ahora sólo se ha visto cómo asignar un tipo de datos predefinido (que siempre es simple) a un elemento.

#### Ejemplo:

Asignar el tipo de datos predefinido `xs:float` al elemento `<ingresosAnuales>`.

```
<xs:element name="ingresosAnuales" type="xs:float"/>
```

La declaración anterior de elemento con un tipo simple asignado es una forma simplificada de la que se muestra a continuación. Se recomienda usar la simplificada.

```
<xs:element name="ingresosAnuales">
  <xs:simpleType>
    <xs:restriction base="xs:float" />
  </xs:simpleType>
</xs:element>
```

Ahora se puede construir un tipo de datos simple, al que se le asignará un nombre, y que será un sinónimo del tipo de datos predefinido xs:float. Se usará el componente de restricción de valores de un tipo simple, xs:restriction. Este tipo no aporta nada nuevo xs:float y se muestra a modo de ejemplo. No se recomienda definir tipos de datos que sean sinónimos literales de predefinidos.

```
<xs:simpleType name="TipoIngresosAnuales"> ①
    <xs:restriction base="xs:float" />
</xs:simpleType>

<xs:element name="ingresosAnuales" type="TipoIngresosAnuales"/> ②
```

① Se define el tipo simple de nombre TipoIngresosAnuales. A partir de ahora se podrá referenciar en todo el esquema y es absolutamente equivalente al predefinido xs:float.

② Se le asigna ese tipo al elemento <ingresosAnuales>.

#### **xs:restriction**

Es un componente que define restricciones en los valores de un tipo.

##### Atributos obligatorios:

- base: nombre del tipo base a partir del cual se construirá el nuevo tipo, sea predefinido o construido.

##### Atributos optativos principales:

- id: identificador único para el componente.

#### Ejemplo:

Se declara un tipo simple que se utilizará para elementos que sean una contraseña. Será de tipo base xs:string, con unas **facetas** (se comentan a continuación) que fuercen su tamaño mínimo a 6 y máximo a 12.

```
<xs:simpleType name="TipoContrasenia">
    <xs:restriction base="xs:string"> ①
        <xs:minLength value="6" /> ②
        <xs:maxLength value="12" /> ③
    </xs:restriction>
</xs:simpleType>
```

① Se está construyendo un nuevo tipo simple a base de restringir el rango de valores permitidos del tipo base.

② Se fuerza a que el nuevo tipo sea de cadenas de longitud mínima de 6. Es una faceta.

③ Se fuerza a que el nuevo tipo sea de cadenas de longitud máxima de 6. Es una faceta.

La declaración de un elemento de nombre <clave> como del tipo recién creado sería:

```
<xss:element name="clave" type="TipoContraseña">
```

Un par de ejemplos de uso, uno válido y uno inválido, en un documento instancia XML son:

<pre>&lt;clave&gt;claveValida!&lt;/clave&gt;</pre>	✓
<pre>&lt;clave&gt;mala&lt;/clave&gt;</pre>	✗

## Facetas para restringir rangos de valores

Se usan para limitar el rango de valores que tiene un tipo base, lo que produce la aparición de un tipo limitado o facetado.

En función del tipo base, existen unas posibles facetas que se pueden aplicar.

Faceta	Uso	Tipos de datos para donde se usa
<code>xs:minInclusive</code>	Especifica el límite inferior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
<code>xs:maxInclusive</code>	Especifica el límite superior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
<code>xs:minExclusive</code>	Especifica el límite inferior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
<code>xs:maxExclusive</code>	Especifica el límite superior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
<code>xs:enumeration</code>	Especifica una lista de valores aceptables.	Todos
<code>xs:pattern</code>	Especifica un patrón o expresión regular que deben cumplir los valores válidos.	Texto
<code>xs:whiteSpace</code>	Especifica cómo se tratan los espacios en blanco, entendiéndose como tales los saltos de línea, tabuladores y los propios espacios. Sus posibles valores son <i>preserve</i> , <i>replace</i> y <i>collapse</i> .	Texto
<code>xs:length</code>	Especifica el número exacto de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto

<b>xs:minLength</b>	Especifica el mínimo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
<b>xs:maxLength</b>	Especifica el máximo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
<b>xs:fractionDigits</b>	Especifica el número máximo de posiciones decimales permitidas en números reales. Ha de ser mayor o igual que 0.	Números con parte decimal
<b>xs:totalDigits</b>	Especifica el número exacto de dígitos permitidos en números. Ha de ser mayor que 0.	Numéricos

**Tabla 4.8:** Facetas**Ejemplo:**

El elemento `<edadLaboral>` es un entero no negativo, que debe tener un valor mínimo de 16, incluido, y máximo 70, no incluido. En notación matemática se emplean los corchetes para identificar los intervalos cerrados (los extremos están incluidos) y paréntesis para identificar los intervalos abiertos (los extremos no están incluidos). En nuestro caso sería [16, 70)

```

<xs:element name="edadLaboral">
  <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:minInclusive value="16"/>
      <xs:maxExclusive value="70"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

```

Si quisieramos **definir un tipo de datos al que poder referenciar repetidas veces** y declarar un elemento `<edadLaboral>` de ese tipo, sería:

```

<xs:simpleType name="TipoEdadLaboral">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="16"/>
    <xs:maxExclusive value="70"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="edadLaboral" type="TipoEdadLaboral">

```

**!** NOTA: Se recomienda llevar a cabo esta práctica, es decir, definir tipos de datos con nombre que puedan ser referenciados posteriormente. Se podría crear una librería de tipos de datos.

Ejemplo:

Se quiere definir un tipo de dato simple, `TipoEstaciones`, basado en el tipo predefinido `xs:token`, y que sólamente permita como valores los nombres de las cuatro estaciones.

```
<xs:simpleType name="TipoEstaciones">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Primavera" />
    <xs:enumeration value="Verano" />
    <xs:enumeration value="Otoño" />
    <xs:enumeration value="Invierno" />
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Se quiere un tipo de datos, `TipoCantidad`, basado en `xs:decimal`, que permita números con 2 dígitos decimales y 11 dígitos totales. El rango de valores implícito sería desde -999.999.999,99 a 999.999.999,99.

```
<xs:simpleType name="TipoCantidad">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="11"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

Ejemplo:

Se define un tipo basado en `xs:string`, usado para registrar direcciones en el que se quiere que todos los caracteres de espacio (espacio, tabulador, nueva línea y retorno de carro) se colapsen, lo que significa que si estos caracteres aparecen al principio o final de la cadena se eliminan, y si lo hacen en su interior, formando una hilera, se sustituyen en un solo espacio.

La definición de este tipo coincide con el tipo predefinido `xs:token`, derivado de `xs:string`.

```
<xs:simpleType name="TipoDireccionFormateada">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="replace" />
  </xs:restriction>
</xs:simpleType>
```

### Actividad 4.6:

Define diferentes tipos simples a partir de las siguientes especificaciones. A continuación, crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos. Por último, escribe un documento instancia XML que sea válido con respecto a ese esquema. Repítelo para los distintos tipos definidos:

- Un número real con tres decimales que represente las temperaturas posibles en la Tierra, suponiendo que van desde -75° a 75°, ambas inclusive.
- Un xs:token que sólo pueda valer las siglas de los países vecinos de España, incluyendo a la propia España: ES, PR, FR, AN.
- Un número real que represente salarios, con 5 dígitos enteros y 2 decimales.
- Un mensaje de la red social Tweeter, conocido como tweet (trino), es una cadena de texto de una longitud máxima de 140 caracteres.

Una faceta que permite usos muy precisos es xs:pattern, que se analiza en detalle.

#### **xs:pattern**

Representa un patrón o expresión regular que debe de cumplir la cadena de texto a la que se aplique.

#### Ejemplo:

Se quiere definir un tipo de datos, basado en xs:string, que se caracterice por cadenas de tres letras mayúsculas.

```
<xs:simpleType name="TipoTresLetrasMayusculas">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][A-Z][A-Z]" />
  </xs:restriction>
</xs:simpleType>
```

La sintaxis usada para estos patrones se basa en expresiones regulares. Las más usadas son:

Patrón	Significado	Ejemplo
.	Cualquier carácter	;
\w	Cualquier letra	M
\d	Un dígito	7
\D	Cualquier carácter no dígito	j
\s	Cualquier carácter de espaciado (tabulador, espacio...)	
\S	Cualquier carácter de no espaciado	C
\d{n}	n dígitos exactamente (Ej. cuatro dígitos exactamente)	3856

\d{n,m}	De n a m dígitos (Ej. de dos o tres dígitos)	91
\d{n,}	n o más dígitos (Ej. Tres o más dígitos)	3500
[xyz]	Uno de los caracteres x, y o z (en minúscula)	Y
[A-Z]	Uno de los caracteres de la A a la Z (en mayúscula)	
[^abc]	Negación de un grupo de caracteres	D
[F-J-[H]]	Sustracción de un carácter de un rango	G
(a b)	Alternativa entre dos expresiones	b
b?	Sucesión de 0 o una ocurrencia de una cadena	B
1*	Sucesión de 0 o más ocurrencias de una cadena	111
(cd)+	Sucesión de 1 o más ocurrencias de una cadena	cdcd

**Tabla 4.9:** Expresiones regulares

Como se ha visto en los patrones anteriores, en las facetas se utilizan cuantificadores (como también se vio en los DTD):

Cuantificador	Significado
?	0 ó 1 ocurrencias
*	0 o más ocurrencias
+	1 o más ocurrencias
{n}	n ocurrencias exactas
{n,m}	De n a m ocurrencias (siendo n < m)
{n,}	n o más ocurrencias

**Tabla 4.10:** Expresiones regulares para cuantificar

Hay ciertos caracteres que tienen un significado propio en las expresiones regulares: {, }, [ , ], (,), ?, \*, +, -, |, ^,,\

Para indicar en una expresión regular la aparición literal de uno de estos caracteres, hay que anteponerle el carácter de escape \.

#### Ejemplo:

La expresión regular para una cadena que contenga las letras a o b o el carácter + sería:

[ab\+]

#### Ejemplo:

Un número de teléfono que se quiere se represente como 3 dígitos, un punto, 3 dígitos, un punto, tres dígitos y un punto. Por ejemplo 912.345.678:

```

<xs:simpleType name="TipoTelefonoPunteado">
    <xs:restriction base="xs:string">
        <xs:pattern value="\d{3}.\d{3}.\d{3}" />
    </xs:restriction>
</xs:simpleType>

```

**Actividad 4.7:**

Define diferentes tipos simples a partir de las siguientes especificaciones.

- Una cadena de texto que represente las matrículas españolas anteriores a la normalización del año 2000. El formato era: una o dos letras mayúsculas + un guion + cuatro dígitos + un guion + una o dos letras mayúsculas. (ej. Z-1234-AB)
- Una cadena de texto que represente las cuatro posibles formas de pago: con tarjeta VISA, MasterCard, American Express y en Efectivo.

A continuación, crea un esquema en el que sólo haya un elemento al que se le asigne un tipo simple de los definidos.

Por último, escribe un documento instancia XML que sea válido con respecto a ese esquema. Repítelo para los distintos tipos definidos.

**Más sobre derivación de tipos simples: uniones y listas**

Otra forma de derivar un tipo de datos consiste en definir uniones o listas de tipos de datos base:

- Unión:** es un tipo de datos creado a partir de una colección de tipos de datos base. Un valor es válido para una unión de tipos de datos si es válido para al menos uno de los tipos de datos que forman la unión. Se construye con el componente `<xs:union>`.

**Ejemplo:**

Se quiere reflejar que las tallas de ropa se pueden identificar por números (38, 40, 42) o por letras, que son las iniciales en inglés (S, M, L).

```

<xs:element name="talla">
    <xs:simpleType>
        <xs:union memberTypes="TipoTallaNumerica TipoTallaTextual" />
    </xs:simpleType>
</xs:element>

<xs:simpleType name="TipoTallaNumerica">
    <xs:restriction base="xs:positiveInteger">
        <xs:enumeration value="38"/>
        <xs:enumeration value="40"/>
        <xs:enumeration value="42"/>
    </xs:restriction>
</xs:simpleType>

```

```

<xs:simpleType name="TipoTallaTextual">
  <xs:restriction base="xs:string">
    <xs:enumeration value="S"/>
    <xs:enumeration value="M"/>
    <xs:enumeration value="L"/>
  </xs:restriction>
</xs:simpleType>

```

Cualquiera de estos dos fragmentos XML serían válidos:

```
<talla>42</talla>
```

Y también:

```
<talla>M</talla>
```

- **Lista:** es un tipo de datos compuesto por listas de valores de un tipo de datos base. La lista de valores debe aparecer separada por espacios. Se construye con el componente `<xs:list>`.

#### Ejemplo:

Se define un tipo `TipoListaNumerosBingo` como una lista de valores enteros positivos que empieza en el 1 y acaba en el 90.

```

<xs:simpleType name="TipoListaNumerosBingo">
  <xs:list>
    <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="90"/>
    </xs:restriction>
  </xs:list>
</xs:simpleType>

<xs:element name="numerosGanan" type="TipoListaNumerosBingo"/>

```

Un fragmento XML que sería válido para la definición precedente será:

```
<numerosGanadores>1 7 19 34 42 60 63 73</numerosGanadores>
```

#### Actividad 4.8:

Dado un tipo simple, `TipoColoresSemaforo`, basado en `xs:string` y que sólo pueda tomar los valores *Rojo*, *Amarillo* y *Verde*, define un nuevo tipo `ListaColoresSemaforo` que sea una lista cuyos elementos sean del tipo `TipoColoresSemaforo`. Por último, crea otro tipo que sea una lista de sólo tres colores de semáforo, `Tipo3ColoresSemaforo`.

Crea un esquema que contenga estos tipos y declarar un elemento del tipo recién construido `Tipo3ColoresSemaforo`.

Crea un documento XML instancia del esquema, que contenga un único elemento de ese tipo. Comprueba que es válido respecto al esquema.

#### 4.8.5. Definición de tipos de datos complejos

Se usará el componente de definición de tipos complejos, `<xs:complexType>`. Sólo se pueden asignar a elementos. Un elemento se considera de tipo complejo si tiene atributos y/o descendientes.

El **contenido** de un elemento es aquello que va entre sus etiquetas de apertura y de cierre. Un elemento puede tener el siguiente **contenido**:

- **Contenido simple** (`<xs:simpleContent>`): el elemento declarado sólo tiene contenido textual, sin elementos descendientes.
- **Contenido complejo** (`<xs:complexContent>`): el elemento declarado tiene elementos descendientes. Puede tener o no contenido textual.

#### Modelos de contenido

En el momento en el que se habla de elementos descendientes, aparece otro elemento en los esquemas que son los **compositores** o **modelos de contenido**.

Indican la distribución que tienen entre sí los elementos descendientes de uno dado. Siempre aparecen conformando un tipo de datos complejo. Hay tres posibilidades distintas, cada una de ellas declarada con un componente:

- **Secuencia:**

Los elementos aparecen en fila, unos detrás de otros, en un orden determinado. Se declara con el componente `<xs:sequence>`. Al igual que sucedía con los elementos, la secuencia puede aparecer un número variable de veces, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

Ejemplo:

El `<mensaje>` con elementos descendientes `<emisor>`, `<receptor>` y `<contenido>`.

```
<xs:sequence>
  <xs:element name="emisor" type="xs:string"/>
  <xs:element name="receptor" type="xs:string"/>
  <xs:element name="contenido" type="xs:string"/>
</xs:sequence>
```

- **Alternativa:**

Los elementos aparecen como alternativa unos de los otros. Sólo se elige uno. Se declara con el componente `<xs:choice>`. La alternativa puede aparecer un número variable de veces, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

Ejemplo:

Al comienzo de una novela podrá haber un elemento `<prologo>`, o un elemento `<prefacio>` o un elemento `<introduccion>`, sólo uno de los tres, todos textuales.

```
<xs:choice>
  <xs:element name="prologo" type="xs:string"/>
  <xs:element name="prefacio" type="xs:string"/>
  <xs:element name="introduccion" type="xs:string"/>
</xs:choice>
```

- **Todos:**

Los elementos aparecen en cualquier orden. Se declara con el componente `<xs:all>`. El uso de este modelo de contenido no se recomienda por la pérdida de control sobre el orden de los elementos. La aparición de todos los elementos puede aparecer 0 o 1 vez, configurable con los atributos `minOccurs` y `maxOccurs` (ambos valen 1 por defecto).

Ejemplo:

Supóngase que existen dos elementos `<alfa>` y `<omega>`, que tienen que aparecer como descendientes de `<griego>`, pero el orden es indiferente. Sería:

```
<xs:all>
  <xs:element name="alfa" type="xs:string"/>
  <xs:element name="omega" type="xs:string"/>
</xs:all>
```

### **xs:complexType**

Es el componente de definición de tipos complejos.

Atributos optativos principales:

- `name`: nombre del tipo. Este atributo es obligatorio si el componente es hijo de `<xs:schema>`, de lo contrario no está permitido.
- `mixed`: indica si se intercala contenido textual con los elementos descendientes:  
Posibles valores: *false, true*.
- `id`: identificador único para el componente.
- `abstract`: indica si se puede usar directamente como tipo de un elemento de un documento instancia XML. Si vale cierto, significa que no puede usarse directamente y el tipo debe derivarse para generar otro tipo que sí se pueda usar.  
Posibles valores: *false, true*.
- `final`: indica si el tipo puede ser derivable por extensión por restricción o por ambas.  
Posibles valores: *extension, restriction, #all*.

Ejemplo:

Un tipo complejo para un elemento `<persona>` que contenga una secuencia de elementos descendientes `<primerApellido>`, `<segundoApellido>` y `<fechaNacimiento>`, de los tipos esperables.

```

<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="primerApellido" type="xs:string" />
      <xs:element name="segundoApellido" type="xs:string" />
      <xs:element name="fechaNacimiento" type="xs:date" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Como ya se ha comentado previamente, se podría definir un **TipoPersona** y declarar un elemento de ese tipo:

```

<xs:complexType name="TipoPersona">
  <xs:sequence>
    <xs:element name="primerApellido" type="xs:string" />
    <xs:element name="segundoApellido" type="xs:string" />
    <xs:element name="fechaNacimiento" type="xs:date" />
  </xs:sequence>
</xs:complexType>

<xs:element name="persona" type="TipoPersona">

```

#### 4.8.6. Diferentes declaraciones de elementos

Para tipos simples y complejos, combinando las posibilidades de contenido (simple y complejo) y los elementos que condicionan el tipo y el contenido (atributos, elementos descendientes, contenido textual) tendríamos la siguiente tabla:

Tipo	Contenido	Elementos descendientes	Atributos	Contenido textual
Simple	Simple			✓
Complejo	Simple		✓	✓
Complejo	Complejo			
Complejo	Complejo		✓	
Complejo	Complejo	✓		
Complejo	Complejo	✓	✓	
Complejo	Complejo mixto	✓		✓
Complejo	Complejo mixto	✓	✓	✓

Tabla 4.11: Tipos de elementos

## Declaración de elementos vacíos

Un elemento es vacío (EMPTY en los DTD) si no tiene contenido textual, ni elementos descendientes (aunque podría tener atributos).

Hay diversas maneras de representarlo:

- Como un tipo simple derivado de `xs:string` con longitud 0.
- Como un tipo complejo sin contenido, que no contiene elementos descendientes. Se aconseja esta.

### Ejemplo:

Para el elemento `<br>` de HTML, se declara un tipo complejo sin contenido.

```
<xs:element name="br">
  <xs:complexType />
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xs:element name="br">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```

## Declaración de elementos con contenido textual y atributos

Se declarara con un tipo de datos complejo con contenido simple que contenga algún atributo.

Lo que se hace realmente es tomar un tipo de datos simple y, mediante un componente de declaración de extensión, `<xs:extension>`, se le añade uno o más atributos, lo que hace que el tipo se convierta en complejo.

### Ejemplo:

El elemento `<textarea>` de HTML, que acepta contenido textual y tiene atributos (`name`, `cols`, `rows`...). Un fragmento de un documento HTML sería:

```
<textarea name="comentarios" cols="10" rows="5">
  Introduzca aquí sus comentarios
</textarea>
```

La definición del tipo complejo para un elemento `<textarea>` sería:

```
<xs:element name="textarea">
  <xs:complexType>
    <xs:simpleContent> ①
      <xs:extension base="xs:string"> ②
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="cols" type="xs:positiveInteger"/>
```

```

<xs:attribute name="rows" type="xs:positiveInteger"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>

```

- ① El componente de definición de contenido simple especifica que el elemento complejo no debe tener elementos descendientes.
- ② El componente de definición de extensión, <xs:extension>, extiende el tipo de datos indicado como base para que el contenido incluya declaración de atributos.

#### Actividad 4.9:

Define un tipo de datos que valide el siguiente fragmento de un documento instancia XML:

```
<temperatura escala="celsius">37</temperatura>
```

El tipo de datos del elemento <temperatura> es xs:integer, y el del atributo escala, uno derivado de xs:string que sólamente permita los valores *Celsius*, *Kelvin* o *Farenheit*.

#### Declaración de elementos sólo con atributos

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún atributo.

##### Ejemplo:

El elemento <img> de HTML, que tiene atributos (src, width, height...) pero no tiene elementos descendientes ni contenido textual.

```

```

La declaración sería:

```

<xs:element name="img">
  <xs:complexType>
    <xs:attribute name="src" type="xs:string"/>
    <xs:attribute name="alt" type="xs:string"/>
  </xs:complexType>
</xs:element>

```

Existe otra declaración alternativa más compleja que se desaconseja.

```

<xs:element name="img">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="src" type="xs:string"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

```

```

<xs:attribute name="alt" type="xs:string"/>
</xs:restriction>
</xs:complexContent>
</xs:complexType>
</xs:element>

```

## Declaración de elementos sólo con elementos descendientes

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún elemento.

### Ejemplo:

El elemento `<ul>` de HTML, que tiene elementos descendientes `<li>` pero no tiene atributos ni contenido textual. Un fragmento en un documento HTML sería:

```

<ul>
    <li>Primer elemento</li>
    <li>Segundo elemento</li>
</ul>

```

La declaración del elemento es:

```

<xs:element name="ul">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="li" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

Existe otra declaración alternativa más compleja que se desaconseja.

```

<xs:element name="ul">
    <xs:complexType>
        <xs:complexContent>
            <xs:restriction base="xs:anyType">
                <xs:sequence>
                    <xs:element name="li" type="xs:string"
                               maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

```

## Declaración de elementos con atributos y elementos descendientes

Se declara con un tipo de datos complejo con contenido complejo que contenga algún elemento descendiente y algún atributo.

### Ejemplo:

El elemento `<table>` de HTML, que tiene atributos, como `border`, y elementos descendientes, como `<tr>`, pero no tiene contenido textual. Un fragmento de un documento HTML podría ser:

```
<table border="1">
  <tr>
    <td>Primera celda</td>
    ...
  </tr>
  ...
</table>
```

La declaración será:

```
<xs:element name="table">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tr" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="border" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="tr">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="td" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Existe otra declaración alternativa más compleja que se desaconseja. En este ejemplo, no se llega a declarar el elemento `<td>`.

```
<xs:element name="table">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element name="tr" type="xs:anyType"
                     maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="border" type="xs:string" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

## Declaración de elementos con contenido textual y elementos descendientes

Se declarara con un tipo de datos complejo con contenido complejo que contenga algún elemento descendiente y contenido textual.

### Ejemplo:

El elemento `<p>` de HTML, suponiendo que no tuviese atributos, puede contener texto directamente, o texto en negrita `<b></b>`, cursiva `<i></i>`, etc.

```
<p>
    El perro de <b>San Roque</b> no tiene rabo porque <i>Ramón
    Rodríguez</i> se lo ha robado.
</p>
```

Para indicar que el contenido es mixto, textual y elementos descendientes, se fija atributo `mixed="true"`. La declaración será:

```
<xss:element name="p">
    <xss:complexType mixed="true"> ①
        <xss:choice minOccurs="0" maxOccurs="unbounded"> ②
            <xss:element name="b" type="xss:string" />
            <xss:element name="i" type="xss:string" />
        </xss:choice>
    </xss:complexType>
</xss:element>
```

① Aparece el atributo `mixed="true"` para indicar un contenido mixto.

② Aparece el modelo de contenido `<xss:choice>`, fijando el valor de sus atributos `minOccurs="0"` y `maxOccurs="unbounded"`, para permitir un número indefinido de ocurrencias del elemento `<b>` o del elemento `<i>`, en el orden que sea.

### Ejemplo:

El mismo que el anterior, pero asegurando que los elementos `<b>` e `<i>` aparecen sólo una vez y en ese orden. Se usará el modelo de contenido `<xss:sequence>`.

```
<xss:element name="p">
    <xss:complexType mixed="true">
        <xss:sequence>
            <xss:element name="b" type="xss:string" />
            <xss:element name="i" type="xss:string" />
        </xss:sequence>
    </xss:complexType>
</xss:element>
```

## Declaración de elementos con atributos, elementos descendientes y contenido textual

Se declara con un tipo de datos complejo con contenido complejo que contenga elementos, atributos y contenido textual.

### Ejemplo:

El elemento `<form>` de HTML, que tiene atributos (`name`, `method...`), elementos descendientes como `<input>` y contenido textual.

```
<form name="f1" method="post">
    Apellido: <input type="text" name="apellido" />
    ...
</form>
```

Para indicar que el contenido es mixto, textual y elementos descendientes, se fija el atributo `mixed="true"`. La declaración será:

```
<xss:element name="form">
    <xss:complexType mixed="true">
        <xss:choice minOccurs="0" maxOccurs="unbounded">
            <xss:element ref="input"/>
        </xss:choice>
        <xss:attribute name="name" type="xs:string"/>
        <xss:attribute name="method" type="xs:string"/>
    </xss:complexType>
</xss:element>
<xss:element name="input">
    <xss:complexType>
        <xss:attribute name="type" type="xs:string" use="required"/>
        <xss:attribute name="name" type="xs:string" use="required"/>
    </xss:complexType>
</xss:element>
```

Existe otra declaración alternativa más compleja que se desaconseja.

```
<xss:element name="form">
    <xss:complexType>
        <xss:complexContent mixed="true">
            <xss:restriction base="xs:anyType">
                <xss:sequence>
                    <xss:element name="input" type="xs:anyType"
                        maxOccurs="unbounded"/>
                </xss:sequence>
                <xss:attribute name="name" type="xs:string"/>
                <xss:attribute name="method" type="xs:string"/>
            </xss:restriction>
        </xss:complexContent>
    </xss:complexType>
</xss:element>
```

## Extensión de un tipo complejo

La extensión de tipos complejos se asemeja a la herencia en la programación orientada a objeto: se pueden añadir nuevos elementos y atributos a tipos complejos ya existentes. Los elementos añadidos aparecerán al final de todos los elementos del tipo base.

### Ejemplo:

Se dispone de un tipo complejo `TipoAlumno` con la siguiente definición:

---

```
<xs:complexType name="TipoAlumno">
  <xs:sequence>
    <xs:element name="Apellido" type="xs:string" />
    <xs:element name="Ciclo" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

---

Se le quiere añadir un elemento numérico `<Curso>` y un atributo que sea el número de Matrícula.

Se tratará un tipo complejo que llamaremos `TipoAlumnoExtendido`:

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

El contenido del elemento que tenga ese tipo es de tipo complejo:

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    ...
  </xs:complexContent>
</xs:complexType>
```

A continuación se describe la extensión, usando el componente de extensión, cuyo atributo `base` permite indicar el tipo de datos que sirve de base, en este caso `TipoAlumno`:

---

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    <xs:extension base="TipoAlumno">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

---

Por último, se define el elemento `<Curso>` y el atributo `Matrícula` que queremos añadir. El atributo o atributos a añadir deberán ubicarse después de la definición de secuencia, alternativa u otros elementos hijo.

---

```
<xs:complexType name="TipoAlumnoExtendido">
  <xs:complexContent>
    <xs:extension base="TipoAlumno">
      <xs:sequence>
        <xs:element ref="Curso" />
      </xs:sequence>
      <xs:attribute name="Matrícula" type="xs:positiveInteger" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

---

```

        use="required" />
    </xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:element name="Curso" type="xs:positiveInteger" />

```

La definición completa del tipo base y el tipo extendido es:

```

<xs:complexType name="TipoAlumno">
    <xs:sequence>
        <xs:element name="Apellido" type="xs:string" />
        <xs:element name="Ciclo" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="TipoAlumnoExtendido">
    <xs:complexContent>
        <xs:extension base="TipoAlumno">
            <xs:sequence>
                <xs:element name="Curso" type="xs:positiveInteger" />
            </xs:sequence>
            <xs:attribute name="Matrícula" type="xs:positiveInteger"
                use="required" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

La declaración de un elemento `<Alumno>` del tipo `TipoAlumnoExtendido` será:

```
<xs:element name="Alumno" type="TipoAlumnoExtendido" />
```

Se concluye este ejemplo mostrando la descripción de elementos que serían válidos de acuerdo al tipo recién definido y otros que serían inválidos.

Ejemplo:

El elemento es válido con respecto a su declaración. Tiene los elementos descendientes definidos, Apellido, Ciclo y Curso, en el orden adecuado. Igualmente tiene el atributo Matrícula.

```

<Alumno Matrícula="123">
    <Apellido>Marín</Apellido>
    <Ciclo>ASIR</Ciclo>
    <Curso>1</Curso>
</Alumno>

```



El elemento no es válido con respecto a su declaración. Tiene los elementos descendientes definidos, Apellido, Ciclo y Curso, pero en un orden distinto al de la declaración.

```

<Alumno Matrícula="123">
    <Apellido>Marín</Apellido>
    <Curso>1</Curso>
    <Ciclo>ASIR</Ciclo>
</Alumno>

```



El elemento no es válido con respecto a su declaración. No existe el atributo Matrícula, cuya aparición es requerida (obligatoria).

```
<Alumno>
  <Apellido>Marín</Apellido>
  <Ciclo>ASIR</Ciclo>
  <Curso>1</Curso>
</Alumno>
```

x

#### 4.8.7. Modelos de diseño de esquemas XML

Existen varios modelos de estructuración de las declaraciones al construir esquemas, si bien conviene recordar que el orden en que aparecen los componentes en un esquema no es representativo.

- Diseño anidado o de muñecas rusas: se llama así porque se anidan declaraciones unas dentro de otras, como las muñecas Matrioskas. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Esto produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones.

Con esta técnica los esquemas son más cortos pero su lectura es más compleja para el ojo humano.

##### Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"/>
        <xs:element name="personaje" minOccurs="0"
                    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="amigoDe" type="xs:string"
                          minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="desde" type="xs:date"/>
              <xs:element name="calificacion" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- **Diseño plano:** se declaran los elementos y los atributos y se indica una referencia a su definición, que se realiza en otro lugar del documento. Es una técnica que recuerda a los DTD.

Ejemplo:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <!-- definición de elementos de tipo simple -->
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="autor" type="xs:string"/>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="amigoDe" type="xs:string"/>
    <xs:element name="desde" type="xs:date"/>
    <xs:element name="calificacion" type="xs:string"/>

    <!-- definición de atributos -->
    <xs:attribute name="isbn" type="xs:string"/>

    <!-- definición de elementos de tipo complejo -->
    <xs:element name="personaje">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nombre"/>
                <xs:element ref="amigoDe" minOccurs="0"
                           maxOccurs="unbounded"/>
                <xs:element ref="desde"/>
                <xs:element ref="calificacion"/>

                <!-- los elementos de tipo simple se referencian con el
                    atributo ref -->
                <!-- la definición de la cardinalidad se hace cuando el
                    elemento es referenciado -->
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="titulo"/>
                <xs:element ref="autor"/>
                <xs:element ref="personaje" minOccurs="0"
                           maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="isbn"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- **Diseño con tipos con nombre reutilizables:** se definen tipos de datos simples o complejos a los que se identifica con un nombre (son plantillas, como las clases en la programación orientada a objeto). Al declarar elementos y atributos, se indica que son de alguno de los tipos con nombre previamente definidos (aquí elementos y atributos son instancias de las plantillas ya definidas, como objetos en la programación orientada a objeto).

Ejemplo:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">

    <!-- definición de tipos simples -->
    <xss:simpleType name="TipoNombre">
        <xss:restriction base="xss:string">
            <xss:maxLength value="32"/>
        </xss:restriction>
    </xss:simpleType>
    <xss:simpleType name="TipoDesde">
        <xss:restriction base="xss:date"/>
    </xss:simpleType>
    <xss:simpleType name="TipoDescripcion">
        <xss:restriction base="xss:string"/>
    </xss:simpleType>
    <xss:simpleType name="TipoISBN">
        <xss:restriction base="xss:string">
            <xss:pattern value="[0-9]{10}" />
        </xss:restriction>
    </xss:simpleType>

    <!-- definición of tipos complejos -->
    <xss:complexType name="TipoPersonaje">
        <xss:sequence>
            <xss:element name="nombre" type="TipoNombre"/>
            <xss:element name="amigoDe" type="TipoNombre"
                         minOccurs="0"
                         maxOccurs="unbounded"/>
            <xss:element name="desde" type="TipoDesde"/>
            <xss:element name="calificacion" type="TipoDescripcion"/>
        </xss:sequence>
    </xss:complexType>
    <xss:complexType name="TipoLibro">
        <xss:sequence>
            <xss:element name="titulo" type="TipoNombre"/>
            <xss:element name="autor" type="TipoNombre"/>

            <!-- definición del elemento personaje, usando el tipo
                 complejo TipoPersonaje -->
            <xss:element name="personaje" type="TipoPersonaje"
                         minOccurs="0"/>
        </xss:sequence>
        <xss:attribute name="isbn" type="TipoISBN" use="required"/>
    </xss:complexType>

```

```

<!-- definición del elemento libro usando el tipo complejo
    TipoLibro -->
<xs:element name="libro" type="TipoLibro"/>
</xs:schema>

```

#### 4.8.8. Poniendo todo junto

Se va a desarrollar un ejemplo de esquema para recopilar los elementos más usados.

Ejemplo:

Se dispone del siguiente documento XML, *persona.xml*:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<persona dni="12345678-L" xsi:noNamespaceSchemaLocation="persona.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <nombre>Juan Antonio</nombre>
    <apellido>Abascal</apellido>
    <estadoCivil>Soltero</estadoCivil>
    <edad>60</edad>
    <enActivo/>
</persona>

```

El esquema XML asociado se quiere que cumpla ciertas restricciones semánticas:

- El atributo *dni* es obligatorio y su formato es de 8 dígitos, un guion y una letra mayúscula.
- El estado civil puede ser: *Soltero*, *Casado*, *Divorciado* o *Viudo*. Por defecto es *Soltero*.
- La edad debe ser de 0 a 150.
- El elemento *<enActivo>* es optativo.

El esquema que cumple con esto, ubicado en el archivo *persona.xsd*, es:

```

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

    <!-- Declaración de elementos -->
    <xs:element name="persona">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellido" type="xs:string"/>
                <xs:element name="estadoCivil" type="TipoEstadoCivil"
                           default="Soltero"/>
                <xs:element name="edad" type="TipoEdadHumana"/>
                <xs:element name="enActivo" minOccurs="0">
                    <xs:complexType/>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="dni" type="TipoDni" use="required"/>
        </xs:complexType>
    </xs:element>

```

```

<!-- Definición de tipos -->
<xs:simpleType name="TipoDni">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{8}\-[A-Z]" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TipoEstadoCivil">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Soltero"/>
        <xs:enumeration value="Casado"/>
        <xs:enumeration value="Divorciado"/>
        <xs:enumeration value="Viudo"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TipoEdadHumana">
    <xs:restriction base="xs:nonNegativeInteger">
        <xs:maxInclusive value="150" />
    </xs:restriction>
</xs:simpleType>

</xs:schema>

```

#### 4.8.9. Construcción avanzada de esquemas

Se verán a continuación elementos complementarios en la construcción de esquemas, que permiten alcanzar altos niveles de sofisticación en su diseño.

##### Grupos de elementos y grupos de atributos

Se pueden definir grupos de elementos y grupos de atributos para poder referenciarlos en definiciones posteriores de tipos compuestos.

Estos grupos no son tipos de datos sino contenedores que albergan un conjunto de elementos o atributos que pueden ser usados en la definición de tipos complejos. Se utilizarán los componentes `<xs:group>` y `<xs:attributeGroup>`.

##### Ejemplo:

```

<!-- definición de un grupo de elementos -->

<xs:group name="ElementosPrincipalesLibro">
    <xs:sequence>
        <xs:element name="titulo" type="TipoNombre"/>
        <xs:element name="autor" type="TipoNombre"/>
    </xs:sequence>
</xs:group>

<!-- definición de un grupo de atributos -->

```

```

<xs:attributeGroup name="AtributosLibro">
  <xs:attribute name="isbn" type="TipoISBN" use="required"/>
  <xs:attribute name="disponible" type="xs:string"/>
</xs:attributeGroup>

<!-- Los grupos se usan en la definición de un tipo complejo -->
<xs:complexType name="TipoLibro">
  <xs:sequence>
    <xs:group ref="ElementosPrincipalesLibro"/>
    <xs:element name="personaje" type="TipoPersonaje"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attributeGroup ref="AtributosLibro"/>
</xs:complexType>

```

## Redefinición de tipos

Este elemento nos va permitir redefinir en un esquema un tipo de datos, simple o complejo, un grupo (de elementos) o un grupo de atributos, que hubieran sido ya definidos en otro documento de esquema al que se referencia. Es semejante a una extensión o herencia de una clase en programación orientada a objeto.

### **xs:redefine**

Es un componente de redefinición de tipos de datos existentes en un documento de esquema que se incluirá. Permite redefinir tipos de datos y grupos de elementos y atributos. Funciona de manera parecida al componente de inclusión de esquemas, **xs:include**.

Elemento padre: **xs:schema**

Atributos obligatorios:

- **schemaLocation:** URI del documento de esquema al que se referencia.

Atributos optativos principales:

- **id:** especifica un identificador único para el elemento.

Ejemplo:

Se define en un esquema un tipo de datos complejo, **TipoTrayecto**, que contiene los elementos **origen** y **destino**. Posteriormente, en otro esquema se quiere hacer uso de este tipo como base para definir otro tipo que, además, contiene un elemento **duracion**.

El primer esquema, almacenado en el documento **esquemaInicial.xsd**, contendrá:

```

<xs:complexType name="TipoTrayecto">
  <xs:sequence>
    <xs:element name="origen"/>
    <xs:element name="destino"/>
  </xs:sequence>
</xs:complexType>

```

El segundo esquema, donde se redefine (reusa) el tipo de datos **TipoTrayecto**, se almacena en otro archivo, por ejemplo **esquemaAmpliado.xsd**. Al nuevo tipo de datos se le llamará como al que extiende, **TipoTrayecto**:

---

```

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

<xs:redefine schemaLocation="esquemaInicial.xsd">
    <xs:complexType name="TipoTrayecto">
        <xs:complexContent>
            <xs:extension base="TipoTrayecto">
                <xs:sequence>
                    <xs:element name="duracion"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:redefine>
<xs:element name="ruta" type="TipoTrayecto"/>

</xs:schema>

```

---

## Grupos de sustitución

Permiten definir elementos que son sinónimos. Un uso muy habitual es para definir los mismos elementos en varios idiomas.

### Ejemplo:

Datos de un cliente en castellano y en inglés. El atributo substitutionGroup en un elemento indica que es del mismo tipo que el valor de ese atributo. En el ejemplo, <name> es equivalente a <nombre> y <customer> a <cliente>.

---

```

<xs:element name="nombre" type="xs:string"/>
<xs:element name="name" substitutionGroup="nombre"/>

<xs:complexType name="TipoInfoCliente">
    <xs:sequence>
        <xs:element ref="nombre"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="cliente" type="TipoInfoCliente"/>
<xs:element name="customer" substitutionGroup="cliente"/>

```

---

Dos fragmentos XML igualmente válidos:

<cliente><nombre>Isabel Sánchez</nombre></cliente>

Y también:

<customer><name>José R. Rodríguez</name></customer>

### **xs:any**

Permite que en el documento XML aparezcan elementos que no han sido explícitamente declarados en el esquema.

Elemento padre: xs:choice, xs:sequence.

Atributos opcionales:

- **id:** identificador único del elemento.
- **maxOccurs:** indica el número máximo de ocasiones en las que el elemento puede aparecer como descendiente del elemento padre. El valor por defecto es 1, pero puede tomar cualquier valor mayor o igual que cero. Si no se quiere poner límite se usará *unbounded* (ilimitado).
- **minOccurs:** indica el número mínimo de ocasiones en las que el elemento puede aparecer como descendiente del elemento padre. El valor por defecto es 1, pero puede tomar cualquier valor mayor o igual que cero.
- **namespace:** especifica los espacios de nombres que contienen los elementos que pueden usarse. Puede valer:
  - **##any:** se permiten elementos de cualquier espacio de nombres. Es el valor por defecto.
  - **##other:** se permiten elementos de cualquier espacio de nombres que no sea el del elemento padre.
  - **##local:** se permiten elementos sin espacio de nombres.
  - **##targetNamespace:** se permiten elementos del espacio de nombres del elemento padre.
  - Una lista de {URI que referencien espacios de nombres, **##targetNamespace**, **##local**}: se permiten elementos de una lista (delimitada por espacios) de espacios de nombres.
- **processContents:** indica cómo debe realizar la validación el procesador de XML frente a los elementos especificados por este **anyElement**. Puede valer:
  - **strict:** el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
  - **lax:** igual que el **strict**, pero si no se obtiene el esquema no se producirá un error.
  - **skip:** el procesador XML no intenta validar ningún elemento de los espacios de nombres especificados.

Ejemplo:

Se declara un elemento **<coche>**, con dos atributos marca y modelo. Al usar el elemento **<xs:any>** en los documentos XML validados por este esquema se podrán añadir otros elementos, colocados detrás de **modelo**, al elemento **coche**.

```

<xs:element name="coche">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo" type="xs:string"/>
      <xs:any minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

**xs:anyAttribute**

Permite que en el documento XML aparezcan asociados a un elemento atributos que no han sido explícitamente declarados en el esquema.

Elemento padre: xs:complexType, xs:restriction (xs:simpleContent y xs:complexContent), xs:extension (xs:simpleContent y xs:complexContent), xs:attributeGroup.

**Atributos opcionales:**

- id: identificador único del elemento.
- namespace: especifica los espacios de nombres que contienen los atributos que pueden usarse. Puede valer:
  - ##any: se permiten atributos de cualquier espacio de nombres. Es el valor por defecto.
  - ##other: se permiten atributos de cualquier espacio de nombres que no sea el del elemento padre.
  - ##local: se permiten atributos sin espacio de nombres.
  - ##targetNamespace: se permiten atributos del espacio de nombres del elemento padre.
  - Una lista de {URI que referencien espacios de nombres, ##targetNamespace, ##local}: se permiten atributos de una lista (delimitada por espacios) de espacios de nombres.
- processContents: indica cómo debe realizar la validación el procesador de XML frente a los atributos especificados por este anyAttribute. Puede valer:
  - strict: el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
  - lax: igual que el strict, pero si no se obtiene el esquema no se producirá un error.
  - skip: el procesador XML no intenta validar ningún elemento de los espacios de nombres especificados.

**Ejemplo:**

Se declara un elemento <coche>, con dos atributos marca y modelo. Al usar el elemento <xs:anyAttribute> en los documentos XML validados por este esquema se podrán añadir otros atributos al elemento coche.

```
<xs:element name="coche">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute />
  </xs:complexType>
</xs:element>
```

## Documentación de esquemas

Para asociar a un esquema comentarios que puedan ser utilizados para generar una documentación con alguna herramienta de autor, se utilizan una serie de instrucciones: `xs:annotation`, `xs:appinfo` y `xs:documentation`.

### **xs:annotation**

Permite especificar comentarios al esquema.

Elemento padre: cualquiera

Sintaxis: `xs:annotation ::= (xs:appinfo | xs:documentation)*`

Atributos optionales:

- `id`: identificador único del elemento.

### **xs:appinfo**

Especifica información que vaya a ser usada por la aplicación.

Elemento padre: `xs:annotation`

Sintaxis: `xs:appinfo ::= Contenido XML bien formado`

Atributos optionales:

- `source`: una URI que especifica el origen de la información sobre la aplicación.

### **xs:documentation**

Se usa para asociar comentarios textuales sobre el esquema.

Elemento padre: `xs:annotation`

Sintaxis: `xs:documentation ::= Contenido XML bien formado`

Atributos optionales:

- `source`: una URI que especifica el origen de la información sobre la aplicación.

**Ejemplo:**

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

    <xs:annotation>
        <xs:appInfo>Nota</xs:appInfo>
        <xs:documentation xml:lang="sp">
            Este esquema tiene una nota
        </xs:documentation>
    </xs:annotation>

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

**xs:include**

Es un componente de inclusión de esquemas externos. Añade las declaraciones y definiciones de un esquema externo al esquema actual. El documento de esquema externo debe tener el mismo espacio de nombres que el esquema actual. Se utiliza para reutilizar tipos ya definidos, en ocasiones en librerías de tipos.

Elemento padre: xs:schema

Atributos obligatorios:

- schemaLocation: URI del esquema a incluir en el espacio de nombres del esquema contenedor.

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

**Ejemplo:**

Se incluyen dos esquemas en otro que actúa como contenedor. El espacio de nombres de los esquemas incluidos debe ser el mismo, sino no funcionará la inclusión.

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

    <xs:include
        schemaLocation="http://www.esquemas.com/empleados.xsd">
    <xs:include
        schemaLocation="http://www.esquemas.com/departamentos.xsd">
    ...
</xs:schema>
```

**xs:import**

Es un componente de importación de esquemas externos. Ofrece la misma funcionalidad que xs:include, a excepción que el esquema importado tiene diferente espacio de nombres que el actual.

Elemento padre: xs:schema

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

- **namespace:** indica el URI del espacio de nombres a importar.
- **schemaLocation:** especifica el URI del esquema del espacio de nombres importado.

Ejemplo:

Se importa un espacio de nombres con el prefijo `xhtml`: para referenciar a un párrafo de XHTML, `<p>`, que aparece en <http://www.w3.org/1999/xhtml>.

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xhtml="http://www.w3.org/1999/xhtml"
            targetNamespace="http://www.w3.org/2001/XMLSchema">

    <xs:import namespace="http://www.w3.org/1999/xhtml"/>
    ...
    <xs:complexType name="UnTipoComplejo">
        <xs:sequence>
            <xs:element ref="xhtml:p" minOccurs="0"/>
            ...
        </xs:sequence>
    </xs:complexType>
    ...
<xs:schema>

```

## Control de la integridad referencial

Mediante una serie de instrucciones se va a poder simular el comportamiento de las restricciones de clave primaria, de clave ajena y de unicidad, que permiten preservar la integridad referencial en los sistemas gestores de bases de datos.

La integridad referencial asegura que un elemento que deba estar relacionado con otro, no pueda estarlo con un elemento inexistente.

Ejemplo:

En una empresa los empleados pertenecen a departamentos que se identifican por números: 10, 20, 30... La integridad referencial controlará que un empleado no pueda ser asignado a un departamento inexistente.

Para lograr este efecto usaremos las instrucciones `xs:key`, `xs:keyref`, `xs:unique`. Todas ellas se apoyan para su construcción en las instrucciones `xs:selector` y `xs:field`.

### `xs:key`

Permite definir un elemento o atributo como clave. El valor de la clave no puede ser nulo ni repetirse, de manera que identifica de manera única al elemento/atributo al que se asocia. Asimismo, el elemento o atributo que constituya la clave no podrá omitirse.

Es un comportamiento equivalente a una clave primaria en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con `xs:keyref`, que permitirá a otro elemento o atributo referenciar a la clave.

Sintaxis: xs:key ::= xs:selector (xs:field)+

Elemento padre: xs:element

Atributos obligatorios:

- name: nombre de la clave.

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

### Ejemplo:

Se tiene un documento XML que representa un pedido. El pedido está compuesto de varios producto, cada uno de los cuales tienen un identificador cuyo valor es único en el pedido. Cada línea de pedido está compuesta de una referencia a un producto, la cantidad de unidades pedidas y un color opcional. Una instancia XML será:

```
<?xml version="1.0"?>
<pedido xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="pedido.xsd">
  <codigo>123ABBCC123</codigo>
  <productos>
    <producto>
      <identificador>557</identificador>
      <nombre>Pantalón</nombre>
      <precio moneda="euro">35.99</precio>
    </producto>
    <producto>
      <identificador>563</identificador>
      <nombre>Abrigo</nombre>
      <precio moneda="euro">65.99</precio>
    </producto>
  </productos>

  <lineas>
    <linea identificador="557">
      <cantidad>2</cantidad>
      <color>Azul</color>
    </linea>
    <linea identificador="557">
      <cantidad>1</cantidad>
      <color>Gris</color>
    </linea>
    <linea identificador="563">
      <cantidad>1</cantidad>
    </linea>
  </lineas>
</pedido>
```

Un esquema muy detallado que valida este XML es:

```

<?xml version="1.0"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
  xsi:noNamespaceSchemaLocation="pedido.xsd">
  <xss:element name="pedido" type="TipoPedido">
    <!-- Definición de una referencia a un elemento clave -->
    <xss:keyref name="prodNumKeyRef" refer="prodNumKey">
      <xss:selector xpath="lineas/linea"/>
      <xss:field xpath="@identificador"/>
    </xss:keyref>
    <!-- Definición de un elemento clave -->
    <xss:key name="prodNumKey">
      <xss:selector xpath="productos/producto"/>
      <xss:field xpath="identificador"/>
    </xss:key>
  </xss:element>
  <xss:complexType name="TipoPedido">
    <xss:sequence>
      <xss:element name="codigo" type="xs:string"/>
      <xss:element name="productos" type="TipoProductos"/>
      <xss:element name="lineas" type="TipoLineas"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="TipoLineas">
    <xss:sequence>
      <xss:element name="linea" type="TipoLinea"
        maxOccurs="unbounded"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="TipoLinea">
    <xss:sequence>
      <xss:element name="cantidad" type="xs:integer"/>
      <xss:element name="color" type="TipoColor" minOccurs="0"/>
    </xss:sequence>
    <xss:attribute name="identificador" type="xs:integer"/>
  </xss:complexType>
  <xss:complexType name="TipoProductos">
    <xss:sequence>
      <xss:element name="producto" type="TipoProducto"
        maxOccurs="unbounded"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="TipoProducto">
    <xss:sequence>
      <xss:element name="identificador" type="xs:integer"/>
      <xss:element name="nombre" type="xs:string"/>
      <xss:element name="precio" type="TipoPrecio"/>
    </xss:sequence>
  </xss:complexType>
  <xss:simpleType name="TipoColor">
    <xss:restriction base="xs:string">
      <xss:enumeration value="Azul"/>
      <xss:enumeration value="Gris"/>
      <xss:enumeration value="Beige"/>
    </xss:restriction>
  </xss:simpleType>
</xss:schema>

```

```

</xs:restriction>
</xs:simpleType>
<xs:complexType name="TipoPrecio">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="moneda" type="xs:token"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:schema>

```

### **xs:keyref**

Permite referenciar a claves primarias (xs:key) o claves únicas (xs:unique), compuestas por elementos y/o atributos.

Es un comportamiento equivalente a una clave ajena en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con xs:key (con la que se define la clave primaria) o con xs:unique (con la que se define la clave única).

Sintaxis: xs:keyref ::= xs:selector (xs:field)+

Elemento padre: xs:element

Atributos obligatorios:

- name: nombre de la referencia a la clave.
- refer: nombre de la clave primaria o clave única a la que hace referencia

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

### **xs:unique**

Permite definir un elemento o atributo como de valor único.

Es un comportamiento equivalente a una clave única en una tabla de un sistema gestor de bases de datos relacional.

Esta instrucción se declara de manera conjunta con xs:keyref, que permitirá a otro elemento o atributo referenciar a la clave única.

Elemento padre: xs:element

Sintaxis: xs:unique ::= xs:selector (xs:field)+

Atributos obligatorios:

- name: nombre de la clave única.

Atributos optativos principales:

- id: especifica un identificador único para el elemento.

**xs:selector**

Especifica una expresión XPath que selecciona un conjunto de elementos para usarlos en la definición de una clave, sea primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: xs:key, xs:keyref, xs:unique

Sintaxis: xs:selector ::= (annotation)?

Atributos obligatorios:

- xpath: especifica una expresión XPath, relativa al elemento que se está declarando, que identifica los elementos hijos a los cuales se aplica la restricción de identidad.

Atributos opcionales:

- id: especifica un identificador único del elemento

**xs:field**

Especifica una expresión XPath que indica los elementos o atributos que formarán parte de la clave que se esté definiendo, sea primaria (xs:key), única (xs:unique) o ajena (xs:keyref).

Elemento padre: xs:key, xs:keyref, xs:unique

Sintaxis: xs:field ::= (annotation)?

Atributos obligatorios:

- xpath: identifica un único elemento o atributo cuyo contenido o valor se usa para la restricción.

Atributos opcionales:

- id: especifica un identificador único del elemento.

## 4.9. Otros mecanismos para validar XML

Existen otros mecanismos para validar documentos XML:

- Relax NG (<http://www.oasis-open.org/committees/relax-ng>)
- Schematron (<http://www.xml.com/pub/a/2003/11/12/schematron.html>)

## 4.10. Otros lenguajes basados en XML

Existen lenguajes basados en XML que se usan para propósitos específicos. Estos lenguajes tendrán su mecanismo de validación (DTD, esquema XML...) que fijará qué elementos y atributos concretos pueden aparecer, con qué valores, en qué orden...

## 4.10.1. SVG

SVG (Scalable Vector Graphics – Gráficos Vectoriales Escalables) es un lenguaje de representación de gráficos vectoriales bidimensionales. Desde el año 2001 es una recomendación del W3C, por lo que muchos navegadores son capaces de mostrar gráficos en este formato.

### Ejemplo:

```
<?xml version="1.0"?>
<?xmlstylesheet href="formas.css" type="text/css"?>
<doc>
<canvas>
  <shape x1="100" y1="100" x2="300" y2="200" x3="200" y3="400">
    <cline x1var="x1" y1var="y1"
           x2var="x2" y2var="y2" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <cline x1var="x1" y1var="y1"
           x2var="x3" y2var="y3" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <cline x1var="x3" y1var="y3"
           x2var="x2" y2var="y2" style="stroke:black; fill:none;
           stroke-width:3;"/>
    <controlpoint xvar="x1" yvar="y1"/>
    <controlpoint xvar="x2" yvar="y2"/>
    <controlpoint xvar="x3" yvar="y3"/>
  </shape>
  <shape x1="200" y1="200" x2="500" y2="300" cx1="100"
         cyl="150" cx2="450" cy2="100">
    <curve x1var="x1" y1var="y1"
           x2var="x2" y2var="y2"
           cx1var="cx1" cylvar="cyl"
           cx2var="cx2" cy2var="cy2"
           style="stroke:green; fill:none; stroke-width:4;"/>
    <cline x1var="x1" y1var="y1"
           x2var="cx1" y2var="cyl" style="stroke:blue; fill:none;
           stroke-width:1;"/>
    <cline x1var="x2" y1var="y2"
           x2var="cx2" y2var="cy2" style="stroke:blue; fill:none;
           stroke-width:1;"/>
    <controlpoint xvar="x1" yvar="y1"/>
    <controlpoint xvar="x2" yvar="y2"/>
    <controlpoint xvar="cx1" yvar="cyl"/>
    <controlpoint xvar="cx2" yvar="cy2"/>
  </shape>
</canvas>
</doc>
```

Nótese que utiliza etiquetas propias que describen elementos de dibujo, como `<canvas>` (lienzo), `<shape>` (forma) o `<curve>` (curva).

Por otra parte, igual que se ha visto en los ejemplos de XML, se aplica al documento una hoja de estilos CSS externa. Esto se ve en la segunda línea del código. Además, y de forma análoga a HTML, existen elementos que disponen del atributo `style`, que permitirá aplicarles estilos.

#### 4.10.2. WML

**WML** (Wireless Mark-up Language – Lenguaje de Marcado Inalámbrico) es un lenguaje de representación de la información que se visualiza en las pantallas de dispositivos móviles que utilicen el protocolo WPA (Wireless Application Protocol – Protocolo de Aplicaciones Inalámbricas). Un ejemplo de este lenguaje es:

```

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
  "http://www.wapforum.org/DTD/wml13.dtd">
<wml>
  <card id="card1" title="Formulario">
    <onevent type="onenterforward">
      <refresh>
        <setvar name="elNombre" value="" />
        <setvar name="elGenero" value="" />
        <setvar name="lasAficiones" value="" />
      </refresh>
    </onevent>
    <onevent type="onenterbackward">
      <refresh>
        <setvar name="elNombre" value="" />
        <setvar name="elGenero" value="" />
        <setvar name="lasAficiones" value="" />
      </refresh>
    </onevent>
    <p>
      Encuesta simple<br/>
      ¿Cuál es tu nombre?<br/><input name="elNombre"/><br/>
      ¿Eres hombre o mujer?<br/>
      <select name="elGenero">
        <option value="Hombre">Soy hombre</option>
        <option value="Mujer">Soy mujer</option>
      </select><br/>
      ¿Cuáles son tus aficiones?<br/>
      <select name="lasAficiones" multiple="true">
        <option value="Viajes">Viajes</option>
        <option value="Deporte">Deporte</option>
        <option value="Lectura">Lectura</option>
        <option value="Cine">Cine</option>
        <option value="Gastronomía">Gastronomía</option>
      </select><br/>
      <anchor>
        <go method="get" href="procesarFormulario.asp">
          <postfield name="nombre" value="$ (elNombre) " />
          <postfield name="genero" value="$ (elGenero) " />
          <postfield name="aficiones" value="$ (lasAficiones) " />
        </go>
        Enviar
      </anchor>
    </p>
  </card>
</wml>
```

Como se ve, hay etiquetas que son específicas de este lenguaje, como `<card>`, `<onevent>` o `<setvar>`. Otras son análogas a las de otros lenguajes de marcas como HTML, por ejemplo `<input>`, `<select>` o `<br />`.

### 4.10.3. RSS

RSS (Really Simple Syndication – Sindicación Realmente Simple) es una familia de formatos de semillas web usadas para publicar información que se actualiza con frecuencia.

Más adelante se dedicará un capítulo a cubrir en detalle esta tecnología.

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
  <channel>
    <title>Ejemplo RSS</title>
    <description>Esto es un ejemplo de una semilla RSS</description>
    <link>http://www.domain.com/link.htm</link>
    <lastBuildDate>Fri, 11 May 2012 11:30:50 -0400 </lastBuildDate>
    <pubDate>Sat, 12 May 2012 09:00:00 -0400 </pubDate>
    <item>
      <title>Ejemplo de item </title>
      <description>Esto es un ejemplo de un ítem</description>
      <link>http://www.domain.com/link.htm</link>
      <guid isPermaLink="false">1102345</guid>
      <pubDate>Sat, 12 May 2012 09:00:00 -0400 </pubDate>
    </item>
  </channel>
</rss>
```

### 4.10.4. Atom

Atom (Atom Syndication Format – Formato Atómico de Sindicación) es un lenguaje usado como semillas web. Es una alternativa a RSS.

### 4.10.5. DocBook

Es un lenguaje de definición de documentos. Una herramienta interesante es XMILMind XML Editor (<http://www.xmlmind.com/xmleditor>).

Ejemplo:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book lang="es" id="libro">
  <title>Un libro muy simple</title>
  <chapter id="capitulo01">
    <title>Capítulo 1</title>
    <para>Por algún sitio hay que comenzar</para>
    <para>Y este es un buen lugar</para>
```

```

</chapter>
<chapter id="capitulo2">
  <title>Capítulo 2</title>
  <para>Por algún sitio hay que continuar</para>
  <para>Y este parece otro buen lugar</para>
</chapter>
</book>

```

#### 4.10.6. XBRL

**XBRL** (Extensible Business Reporting Language – Lenguaje Extensible de Informes Financieros) es un estándar abierto que permite representar la información y la expresión de la semántica requerida en los informes financieros.

### 4.11. Otras formas de almacenar información

#### 4.11.1. JSON

**JSON** (JavaScript Object Notation – Notación de Objetos JavaScript) es, de igual manera que XML, un formato en modo texto para el almacenamiento y transmisión de información.

Se usa frecuentemente como alternativa a XML en el envío de datos, por ejemplo en **AJAX** (Asynchronous JavaScript And XML – JavaScript Asíncrono y XML). El motivo fundamental es que el procesado de un documento en formato JSON es más sencillo que el de un documento XML.

Aunque se suelen ver como antagónicas, en ocasiones se usan de manera conjunta ambas tecnologías.

Permite representar objetos en aplicaciones RIA (Rich Internet Application – Aplicaciones Ricas de Internet) que usan JavaScript.

El término JSON se sustituye en ocasiones por el de **LJS** (Literal JavaScript).

JSON se apoya en dos estructuras de datos:

- Una colección de pares nombre/valor. En algunos lenguajes de programación a esto se le denomina objeto, registro, diccionario, tabla hash o **array asociativo**.
- Una lista ordenada de valores, lo que en otros lenguajes se denomina array, vector, lista o secuencia.

Un objeto es un conjunto desordenado de pares etiqueta/valor. La definición de un objeto comienza con la llave de apertura, “{“ y termina con la llave de cierre, “}”. Cada nombre va seguido del carácter dos puntos “:” y los pares nombre/valor se separan por el carácter coma “,”.

Un array es una colección ordenada de valores. Comienza por el carácter corchete de apertura “[“ y termina por el carácter corchete de cierre “]”. Los valores se separan por el carácter coma “,”.

Un valor puede ser:

- una cadena de texto entre comillas dobles
- un número
- true, false o null
- un objeto
- un array

Una cadena de texto es una secuencia de cero o más caracteres Unicode, rodeados por dobles comillas. Se usa el carácter barra invertida (\) como marca de escape. Un carácter se representa como una cadena de un solo carácter. Es equivalente al *String* de Java.

Un número es como los números en Java. Puede tener parte entera, entera + decimal, entera + exponente y entera + decimal + exponente.

#### Sintaxis:

```

objeto ::= { } | { miembros }
miembros ::= par | par, miembros
par ::= string : valor
array ::= [ ] | [ elementos ]
elementos ::= valor | valor, elementos
valor ::= string | número | objeto | array | true | false | null

```

Existen herramientas de conversión automática entre JSON y XML:

- <http://json.online-toolz.com/tools/xml-json-convertor.php>
- <http://jsontoxml.utilities-online.info>

#### Ejemplo:

Se dispone de un array que contiene dos objetos (llámense país). Cada uno de ellos tiene varios pares nombre/valor (atributos con sus valores), a saber, pais, población y animales. A su vez, animales tiene como valor un array de valores.

---

```

[ {
    "pais" : "Nueva Zelanda",
    "poblacion" : 3993817,
    "animales" : ["Oveja", "Kiwi"]
},
{
    "pais" : "Singapur",
    "poblacion" : 4553893,
    "animales" : ["Tigre"]
}
]

```

---

Nº caracteres totales: 135 (se excluyen todos los espacios aparecidos para formatear)

Nº caracteres de información: 49

Porcentaje de caracteres de información respecto al total:  $49 / 135 = 36,29\%$

La información equivalente en formato XML sería:

```
<paises>
  <pais>
    <nombre>Nueva Zelanda</nombre>
    <poblacion>3993817</poblacion>
    <animales>
      <animal>Oveja</animal>
      <animal>Kiwi</animal>
    </animales>
  </pais>
  <pais>
    <nombre>Singapur</nombre>
    <poblacion>4553893</poblacion>
    <animales>
      <animal>Tigre</animal>
    </animales>
  </pais>
</paises>
```

Nº caracteres totales: 255 (se excluyen todos los espacios aparecidos para formatear)

Nº caracteres de información: 49

Porcentaje de caracteres de información respecto al total:  $49 / 255 = 19,21\%$

#### 4.11.2. YAML

**YAML** (acrónimo recursivo de **YAML Ain't Markup Language** – YAML no es otro lenguaje de marcas, aunque también se refiere como **Yet Another Markup Language**), es un formato de almacenamiento de información o serialización de datos, ligero y de fácil lectura para el ojo humano. Comparte con JSON ciertos elementos como las **listas** y los **arrays asociativos**.

Ejemplo:

Una lista de frutas en **formato de bloque** donde cada elemento se denota por un guion “-”; y en **formato lineal**, donde los elementos se rodean de un corchete de apertura “[” y uno de cierre “]” y se separan entre sí por comas “,”:

- Mandarina
- Fresa
- Sandía

[Mandarina, Fresa, Sandía]

Ejemplo:

Un array asociativo en **formato de bloque**, en el que aparece la etiqueta, el carácter dos puntos ":" y el valor; y en **formato lineal**, donde los pares etiqueta/valor se rodean por una llave de apertura "{" y una de cierra "}" y se separan entre sí por comas ",":

nombre: Casilda

apellido: Marín

{nombre: Casilda, apellido: Marín}

A partir de estos elementos básicos se pueden construir estructuras más complejas: listas de arrays asociativos, arrays asociativos de listas...

## Ejercicios propuestos

- Se quiere guardar en formato XML la información relativa a tickets de compra, según las siguientes especificaciones:
  - Datos del ticket
    - o Código del ticket, en un atributo requerido de tipo id
    - o Fecha y hora
    - o Precio total:
      - Precio sin IVA total
      - Cantidad IVA
      - PVP total
      - La moneda se guarda como atributo de tipo token
    - o Forma de pago: puede ser en efectivo o con tarjeta
      - Si es con tarjeta:
        - Tipo de tarjeta, en un atributo
        - 12 asteriscos y los cuatro últimos dígitos de la tarjeta, en un atributo
        - Nombre del cliente
  - Datos del comercio
    - o Nombre
    - o Dirección completa
    - o CIF
    - o Teléfono
  - Datos de la compra:
    - o Líneas de compra:
      - Nombre del artículo
      - Cantidad
      - Precio unitario (sin IVA)
      - IVA
      - PVP

Se ofrece esta instancia XML de prueba:

```
<?xml version="1.0"?>
<ticket código="t00037">
  <datos_ticket>
    <fecha>
```

```

<hora>
<precio_total moneda="euro">
  <cantidad_sin_iva></sin_iva>
  <cantidad_iva></cantidad_iva>
  <cantidad_con_iva></cantidad_con_iva>
</precio_total>
<forma_pago número="*****1234" tipo="Visa Clásica">
  Tarjeta
</forma_pago>
<cliente>Pedro Medario</cliente>
</datos_ticket>

<fecha>2000-02-02</fecha>
<estadio espectadores="49000">Lansdowne Road</estadio>
<local>Irlanda</local>
<visitante>Inglaterra</visitante>
<resultado local="17" visitante="6" />
</partido>
<partido numero="2">
  <fecha>2000-02-03</fecha>
  <estadio>
    El parque de los príncipes</estadio>
  <local>Francia</local>
  <visitante>Gales</visitante>
  <aplazado />
</partido>
...
</jornada>
<jornada numero="2">
...
</jornada>
...
</temporada>

```

Se pide:

- Escribir otro documento XML que se ajuste a las especificaciones dadas pero que contenga alguna variación en la estructura. Así se tienen dos instancias del mismo modelo.
2. Crea el DTD que valide las especificaciones dadas en el ejercicio anterior, de las cuales el documento XML generado no es más que un caso concreto.
- Se recomienda hacer el DTD incremental por secciones del documento, es decir, crear un DTD para un supuesto documento que incluyera sólo los datos del ticket y probarlo con un documento XML cuyo elemento raíz se sea `<datos_ticket>`. Haz lo mismo con los datos del comercio y con los datos de la compra. Por último, intégralos todos.
3. Crea el esquema XML equivalente al DTD del ejercicio anterior. También se puede hacer el diseño incremental.
  4. Dado el siguiente fragmento XML, que representa la información relativa a los partidos jugados en una temporada en el torneo de rugby de las seis naciones, genera el DTD que mejor lo valide.

```

<?xml version="1.0"?>
<temporada año="2000">
  <jornada numero="1">
    <partido numero="1">
      <fecha>2000-02-02</fecha>
      <estadio espectadores="49000">Lansdowne Road</estadio>
      <local>Irlanda</local>
      <visitante>Inglaterra</visitante>
      <resultado local="17" visitante="6" />
    </partido>
    <partido numero="2">
      <fecha>2000-02-03</fecha>
      <estadio>
        El parque de los príncipes</estadio>
      <local>Francia</local>
      <visitante>Gales</visitante>
      <aplazado />
    </partido>
    ...
  </jornada>
  <jornada numero="2">
    ...
  </jornada>
  ...
</temporada>

```

Se cumplen las siguientes condiciones:

- a. Una jornada tiene 3 partidos.
  - b. Una temporada tiene 5 jornadas.
  - c. Si un partido se juega, aparece la etiqueta `<resultado>`, con los puntos del equipo local y del visitante, así como el atributo `espectadores` del elemento `<estadio>`, que representa el público asistente; y si se suspende, aparece `<aplazado>` y no aparece `espectadores`.
  - d. El atributo `espectadores` tendrá un valor por defecto de 0 y como máximo puede ser 80.000.
  - e. Los países que juegan son siempre los mismos: Inglaterra, Francia, Irlanda, Escocia, Gales e Italia.
5. Genera el esquema XML que valide el documento del ejercicio anterior de una manera más precisa.
  6. Genera un documento XML que represente un currículum. Deberá contener las siguientes secciones:
    - a. Datos personales: nombre, apellidos, fecha y lugar de nacimiento, nacionalidad(es), número de identificación (nif o nie) y nombre de un archivo que represente la foto.

- b. Datos de contacto: tipo de vía, nombre de la vía, número (optativo), portal (optativo), escalera (optativo), piso, puerta (optativo), código postal, país, email, teléfono móvil, teléfono fijo (optativo).
- c. Datos de contacto adicionales: página web, cuentas de redes sociales: LinkedIn, Facebook, Twitter, etc.
- d. Formación: para cada estudio realizado, nombre del mismo, lugar de realización, fecha de inicio y fecha de fin.
- e. Idiomas: para cada idioma, nombre del mismo, nivel (alto, medio, bajo) de expresión oral (optativo), nivel de comprensión oral (optativo), nivel de expresión escrita (optativo), nivel de comprensión escrita (optativo). Hay que indicar de cada idioma si es materno.
- f. Experiencia laboral: para cada experiencia, lugar de la misma, puesto desempeñado (optativo), fecha de comienzo y fecha de fin.
- g. Competencias socio-profesionales: para cada competencia socio-profesional (liderazgo, trabajo en equipo, iniciativa, etc.), nombre de la misma y nivel de 1 (muy bajo) a 5 (muy alto).
- h. Datos adicionales: aficiones, disponibilidad para viajar, vehículo propio, licencia(s) de conducir, etc.

Rellena el documento con los datos personales y comprobar que está bien formado.

7. Construye un esquema que valide el documento XML del ejercicio anterior. Se recomienda usar alguna herramienta de inferencia para generar el armazón del esquema y luego refinarlo con las reglas de negocio adecuadas.
8. Genera un documento XML de ejemplo que sea válido con respecto al siguiente DTD.

```
<!ELEMENT mensaje ( email | carta ) >
<!ELEMENT email ( cabecera, asunto?, texto+ ) >
<!ATTLIST email respuesta ( si | no ) "no" >
<!ELEMENT carta ( encabezado, texto ) >
<!ATTLIST carta respuesta ( si | no ) "no" >
<!ELEMENT cabecera ( emisor, receptor*, fecha?) >
<!ELEMENT asunto ( #PCDATA ) >
<!ELEMENT texto ( #PCDATA | saludo )* >
<!ELEMENT encabezado ( emisor, receptor*, fecha ) >
<!ELEMENT emisor ( #PCDATA ) >
<!ELEMENT receptor ( #PCDATA ) >
<!ELEMENT fecha ( #PCDATA ) >
<!ELEMENT saludo ( #PCDATA ) >
```

9. Convierte el anterior DTD en un esquema.
10. Construye los tipos de datos de esquemas XML más adecuados para representar:

- a. Un número de teléfonos con prefijo internacional, como el +34.91.234.56.78.
- b. Una matrícula de coche actual española, como la 1234-BCD.
- c. Un dni o un nie, que sean alternativa el uno del otro.
- d. Un correo electrónico.

