


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.

1.8. 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 estos.

- 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 valide:

```
<?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:

```
<xs: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. Así mismo, 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.

Posibles valores: *unqualified*, *qualified*

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

Posibles 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.

```
<xs:element name="semaforo" type="xs: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 solamente 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 <xs: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.

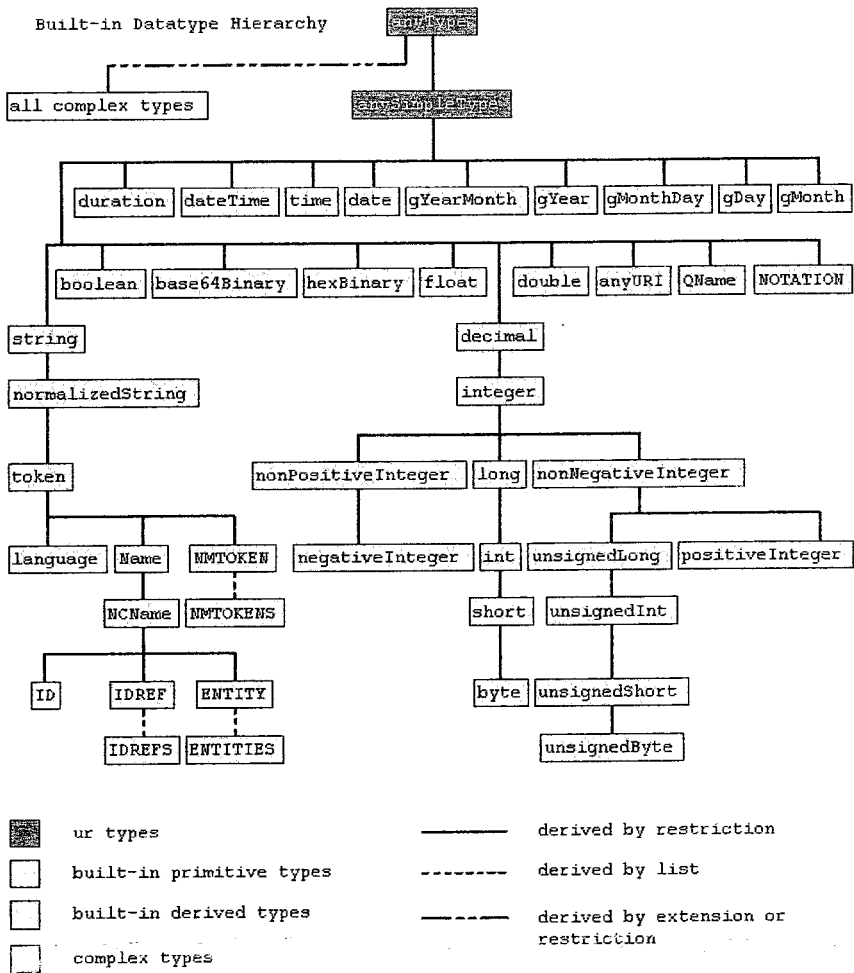


Figura 4.6: Tipos de datos predefinidos (primitivos, derivados y especiales)

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

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

Tabla 4.2: Tipos de datos numéricos

De 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	Sólo la 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 \rightarrow Gregoriano)
gYear	Sólo el año, en formato yyyy (g \rightarrow Gregoriano)
gYearMonth	Sólo el año y el mes, en formato yyyy-mm (g \rightarrow Gregoriano)
gMonthDay	Sólo el mes y el año, en formato -mm-dd (g \rightarrow 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;

$$\underbrace{-?d\{4\}-d\{2\}d\{2\}}_{\text{date}} \underbrace{Td\{2}:\underbrace{d\{2\}:d\{2\}}_{\text{time}}(\underbrace{(\backslash d+)?((\backslash +)\backslash d\{2}:\backslash d\{2})Z}_{\text{común a date y time}})?}$$

Patrón	Significado
-?	Un signo negativo (opcional). Para representar fechas anteriores al año 0
$\backslash d\{4\}-\backslash d\{2\}\backslash d\{2\}$	La parte de la fecha (obligatoria). Equivale a yyyy-mm-dd
$\backslash d\{2}:\backslash d\{2}:\backslash d\{2\}$	La parte de la hora (obligatoria). Equivale a hh:mm:ss
T	Indicador de hora local
$(\backslash d+)?$	La parte decimal de la fracción de segundo (opcional)
$((\backslash +)\backslash d\{2}:\backslash d\{2})Z)?$	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
normalizedString	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 para atributo según XML 1.0, compatible con DTD
NMTOKENS	Lista separada por espacios 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 para atributo según XML 1.0, compatible con DTD. Han de ser valores únicos en el documento XML
IDREF	Tipo de datos para atributo según XML 1.0, compatible con DTD
IDREFS	Lista separada por espacios de IDREF, compatible con DTD
ENTITY	Tipo de datos para atributo en XML 1.0, compatible con DTD
ENTITIES	Lista separada por espacios de ENTITY, compatible con DTD
NOTATION	Tipo de datos para atributo 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:

```
<xs:element name="elementoA" type="date"/> ❶
<xs:element name="elementoB" type="w3c:date"/> ❷
<xs:element name="elementoC" type="xsd:date"/> ❸
<xs:element name="elementoD" type="xs:date"/> ❹
```

❶ El tipo de datos `date` no se puede resolver como ninguno de los componentes de definición de tipos del esquema.

❷ El tipo de datos `w3c:date` tiene un prefijo, `w3c`, que no ha sido declarado.

❸ 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.

❹ 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 redefinidos.

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"> ❶
  <xs:restriction base="xs:float" />
</xs:simpleType>

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

- ❶ 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`.
- ❷ 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="TipoContraseña">
  <xs:restriction base="xs:string"> ❶
    <xs:minLength value="6" /> ❷
    <xs:maxLength value="12" /> ❸
  </xs:restriction>
</xs:simpleType>
```

- ❶ Se está construyendo un nuevo tipo simple a base de restringir el rango de valores permitidos del tipo base.
- ❷ Se fuerza a que el nuevo tipo sea de cadenas de longitud mínima de 6. En una faceta.
- ❸ 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:

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

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

```
<clave>claveValida!</clave>      ✓
<clave>mala</clave>              ✗
```

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
xs:minInclusive	Especifica el límite inferior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
xs:maxInclusive	Especifica el límite superior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas
xs:minExclusive	Especifica el límite inferior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
xs:maxExclusive	Especifica el límite superior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas
xs:enumeration	Especifica una lista de valores aceptables.	Todos
xs:pattern	Especifica un patrón o expresión regular que deben cumplir los valores válidos.	Texto
xs:whiteSpace	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
xs:length	Especifica el número exacto de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto

xs:minLength	Especifica el mínimo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
xs:maxLength	Especifica el máximo número de caracteres (o elementos de una lista) permitidos. Ha de ser mayor o igual que 0.	Texto
xs:fractionDigits	Especifica el número máximo de posiciones decimales permitidas en números reales. Ha de ser mayor o igual que 0.	Números con parte decimal
xs:totalDigits	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 quisiéramos **definir un tipo de datos al que poder referenciar repetidas veces** y declarar el 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 sólo 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 espaciado (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:

- a. Un número real con tres decimales que represente las temperaturas posibles en la Tierra, suponiendo que van desde -75° a 75°, ambas inclusive.
- b. Un `xs:token` que sólo pueda valer las siglas de los países vecinos de España, incluyendo a la propia España: ES, PR, FR, AN.
- c. Un número real que represente salarios, con 5 dígitos enteros y 2 decimales.
- d. Un mensaje de la red social Twitter, 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 plique.

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	;
[Cualquier letra	M
[0-9]	Un dígito	7
[^0-9]	Cualquier carácter no dígito	j
[\t]	Cualquier carácter de espaciado (tabulador, espacio...)	
[^\s]	Cualquier carácter de no espaciado	C
{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 ocurrencias de una cadena	B
l*	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="primerApellido" 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="primerApellido" type="xs:string" />
    <xs:element name="fechaNacimiento" type="xs:date" />
  </xs:sequence>
</xs:complexType>

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

1.8.6. 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
 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 declara 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:extension>
    </xs:complexType>
  </xs:element>
```

```

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

```

❶ El componente de definición de contenido simple especifica que el elemento complejo no debe tener elementos descendientes.

❷ 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`, no derivado de `xs:string` que sólo permita los valores *Celsius*, *Kelvin* o *Fahrenheit*.

Declaración de elementos sólo con atributos

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

Ejemplo:

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

```

Declaración de elementos sólo con elementos descendientes

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

Ejemplo:

El elemento `` de HTML, que tiene elementos descendientes `` 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 lea 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 declara 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 ``, 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:

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

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

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

Ejemplo:

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

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

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

Existe otra declaración alternativa más compleja que se desaconseja.

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

```
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 finidos, 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 finidos, 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>
```

✗

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="io-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"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- los elementos de tipo simple se referencian con el
  atributo ref -->
  <!-- la definición de la cardinalidad se hace cuando el
  elemento es referenciado -->
  <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"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

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

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

      <!-- definición del elemento personaje, usando el tipo
      complejo TipoPersonaje -->
      <xs:element name="personaje" type="TipoPersonaje"
                    minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="TipoISBN" use="required"/>
  </xs: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:
 - o `##any`: se permiten elementos de cualquier espacio de nombres. Es el valor por defecto.
 - o `##other`: se permiten elementos de cualquier espacio de nombres que no sea el del elemento padre.
 - o `##local`: se permiten elementos sin espacio de nombres.
 - o `##targetNamespace`: se permiten elementos del espacio de nombres del elemento padre.
 - o 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:
 - o `strict`: el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
 - o `lax`: igual que el `strict`, pero si no se obtiene el esquema no se producirá un error.
 - o `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:
 - o `##any`: se permiten atributos de cualquier espacio de nombres. Es el valor por defecto.
 - o `##other`: se permiten atributos de cualquier espacio de nombres que no sea el del elemento padre.
 - o `##local`: se permiten atributos sin espacio de nombres.
 - o `##targetNamespace`: se permiten atributos del espacio de nombres del elemento padre.
 - o 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:
 - o `strict`: el procesador XML debe obtener el esquema para los espacios de nombres requeridos y validar los elementos. Es el valor por defecto.
 - o `lax`: igual que el `strict`, pero si no se obtiene el esquema no se producirá un error.
 - o `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 opcionales:

- `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 opcionales:

- `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 opcionales:

- `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 HTML, <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>
```

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 productos, 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>
```

Control de la integridad referencial

Mediante una serie de instrucciones se va a poder simular el comportamiento de las relaciones 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 elemento no pueda estarlo con un elemento inexistente.

Ejemplo:

En una empresa los empleados pertenecen a departamentos que se identifican por números: 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 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.

Un esquema muy detallado que valida este XML es:

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

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>)

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"?>
<?xml-stylesheet 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"
          cy1="150" cx2="450" cy2="100">
    <curve x1var="x1" y1var="y1"
           x2var="x2" y2var="y2"
           cx1var="cx1" cy1var="cy1"
           cx2var="cx2" cy2var="cy2"
           style="stroke:green; fill:none; stroke-width:4;"/>
    <cline x1var="x1" y1var="y1"
          x2var="cx1" y2var="cy1" 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="cy1"/>
    <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
.

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 item</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 XMLMind 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="capitulo1">
    <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 procesamiento 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-converter.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, país, 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:

```
<países>
  <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>
</países>
```

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: $49 / 255 = 19,21\%$

11.2. YAML

YAML (acrónimo recursivo de **Y**AML **A**in't **M**arkup **L**anguage – YAML no es otro lenguaje e marcas, aunque también se refiere como **Y**et **A**nother **M**arkup **L**anguage), es un formato de Imacenamiento 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 **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 pontos “:” y el valor; y en **formato lineal**, donde los pares etiqueta/valor se rodean por una llave de apertura “{” y una de cierre “}” 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...