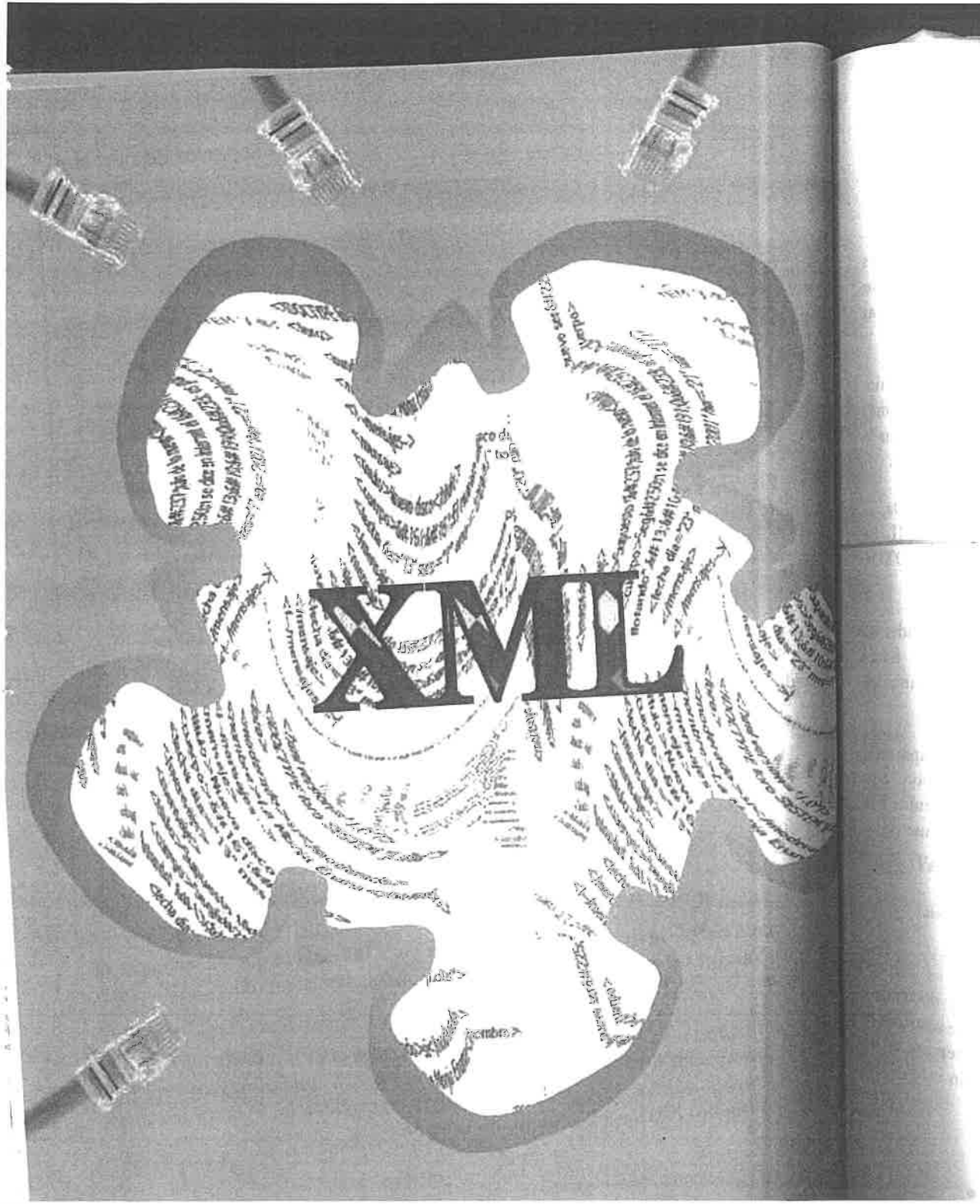


CAPÍTULO 6

XML Schema

En este capítulo se explicará el otra forma de especificar las reglas estructurales que deben cumplir los documentos para ser válidos. Son los esquemas y proporcionan un mayor control sobre los elementos y atributos que se utilizarán en el documento XML. Veremos las ventajas e inconvenientes de los esquemas respecto a las DTD.

6.1 XML SCHEMA

Un documento debe ser validado contra una serie de definiciones estructurales para obtener de él todo el rendimiento que se espera. Como ya hemos visto, las DTDs presentan algunos inconvenientes, ya que se pueden realizar pocas restricciones sobre los contenidos.

Como ejemplo de las limitaciones de una DTD podemos decir no permite definir el número máximo de veces que debe aparecer un elemento. Además presenta la desventaja de que la sintaxis con la que se crea no es la de XML, por lo que requiere el aprendizaje de otro lenguaje (aunque éste sea fácil de aprender).

Para solventar los problemas o inconvenientes que presentan las DTDs, se intentó encontrar una manera de escribir una DTD aplicando el estándar XML. Se definió XML Schema que, aunque tiene una sintaxis más compleja que las DTDs, proporciona un mayor control sobre la información que éstas contienen.

Con XML Schema se distingue entre elementos globales y locales, además no sólo permite especificar el tipo de dato que es, sino que permite hacer restricciones sobre el mismo, como podría ser el caso de necesitar determinar un máximo de caracteres en una cadena o que sean dígitos numéricos.

Es probable que el uso de las DTDs, al igual que el HTML, no desaparezca, al menos en mucho tiempo, debido fundamentalmente a su facilidad de uso y a su universalidad. Pero poco a poco y, conforme vayan apareciendo herramientas que permitan un mejor aprovechamiento del estándar XML, se hará un mayor uso de los Esquemas.

6.2 COMPARACIÓN DTD Y XML SCHEMA

Aunque sirven para obtener el mismo resultado (validar un documento XML), hay una serie de diferencias entre DTDs y los Esquemas. En este apartado se pretende que el lector tenga una idea clara de cuando tiene que hacer uso de las DTDs, y cuando tiene que utilizar los Esquemas. También se indicarán las ventajas e inconvenientes del uso de una u otra tecnología.

Ventajas e inconvenientes de la DTD

Las DTD presentan los siguientes inconvenientes:

- Poseen una sintaxis especializada no válida según XML.
- Es poco intuitiva su representación de los documentos.
- Utiliza caracteres especiales para dar significado a las secuencias ("+", "*", "?").
- No permite realizar restricciones avanzadas sobre el contenido de elementos y atributos.
- No soportan espacios de nombres.
- No soportan herencia.

Como ventajas, se pueden mencionar las siguientes:

- Es más compacta que las XML Schema.
- Fácil de aprender.

Ventajas e inconvenientes del Esquema

Las ventajas que presenta sobre las DTDs son:

- Soportan restricciones restricciones avanzadas sobre el contenido de elementos y atributos.
- Se puede analizar como cualquier otro documento XML puesto que cumplen con la sintaxis XML.
- Soportan una serie extensa de tipos de datos, mientras que las DTDs tratan todos los datos como cadenas.
- Permite ampliar vocabularios y soporta grupos de atributos.
- Admite y está basado en espacios de nombres.

Como desventajas, presenta las siguientes:

- Son más complicadas de aprender y desarrollar que las DTDs.
- Su uso en los navegadores es muy limitado, ya que hasta el momento solo las últimas versiones de Internet Explorer y NetScape permiten su uso.

Tabla resumen comparando DTD Y Schemas

OBJETO	*DTD*	*ESQUEMA XML*
Sintaxis	EBNF + pseudo XML (no XML)	XML 1.0.
Herramientas	Específicas, aunque existen muchas.	Usa casi todas las herramientas XML, incluyendo DOM.
Soportado por DOM	No tiene	Se puede visualizar y manipular con DOM.
Modelos para el contenido	Poco soporte, apenas hay restricciones.	Mucho soporte; hay muchos tipos de restricciones sobre los elementos y atributos.
Tipos de datos	Cadenas, Tokens e ID.	Todos los tipos más frecuentes. Se incluyen fecha, hora y otras.
Ámbito de nombres	Sólo nombres globales.	Nombres locales y globales.
Herencia	No	Sí
Extensibilidad	Limitada.	Ilimitada.
Soporte para múltiples vocabularios	No, una DTD por documento.	Si, utilizando espacios de nombres.

Tabla 6.1. Comparación de DTD y Esquema

Selección de XML Schema

En principio, todo parece indicar que es mejor evolucionar y utilizar los esquemas en lugar de las DTDs que ofrecen pocas posibilidades. Hay que tener en cuenta que las DTDs existen desde el comienzo del estándar XML, mientras que los esquemas son relativamente nuevos, por lo que aunque en el futuro se haga uso de éstos, las DTD no desaparecerán rápidamente ya actualmente son utilizadas por un gran número de usuarios y gran cantidad de aplicaciones.

El método a utilizar es una elección personal:

- Si quiere facilidad de uso empleará DTD.
- Si prefiere mayor funcionalidad y control utilizará esquemas.

6.3 XML SCHEMA

Un esquema es un documento XML (en un fichero de extensión xsd) y su estructura es por tanto la de un documento como, por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Ejemplo de XML Schema-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
...
</xs:schema>
```

Tiene la siguiente estructura:

- La cabecera es igual a la de todos los documentos XML, además no incluye declaración DOCTYPE porque no se validará con una DTD.
- El cuerpo del documento está, como todos los documentos, contenido en un elemento raíz. Siempre será de nombre "*schema*".
- Todos los elementos para la creación del esquema se enmarcan en distintos espacios de nombres.

Dentro del esquema en sí (dentro del elemento "*schema*"), podemos distinguir dos focos de atención principales, el contenido del esquema y la declaración inicial del esquema que incluye los espacios de nombres.

6.3.1 Declaración de esquema

Esta declaración está comprendida en el marcado de apertura del elemento "*schema*". La declaración no es tal pero la llamaremos así por crear un cierto marco para el desarrollo de las explicaciones.

En esta declaración tenemos una serie de atributos que nos van a servir para establecer ciertas características de nuestro esquema que son:

- Espacios de nombres implicados.
- Información sobre de formato libre sobre el schema.
- Características internas del esquema.
- Características de los documentos que vamos a crear.
- Información del idioma.

De estos valores sólo uno es estrictamente obligatorio, pero los referidos a espacios de nombres será usados casi siempre.

Este es un ejemplo de atributos sobre espacios de nombres:

```
<xsd:schema -
        xsd:xmlns="http://www.w3.org/2001/XMLSchema" -
        xmlns:esp-a="http://www.ejemplo.como/01"
        xmlns:esp-b="http://www.ejemplo.como/02"
        targetNamespace="http://www.ejemplo.como/01 http://www.ejemplo.como/02">
```

En el ejemplo tenemos:

- En primer lugar el atributo obligatorio y de valor fijo que es "*xmlns*" que establece el espacio de nombres de los esquema. Puede llevar sinónimo o no según deseemos, pero lo corriente es que lleve "*xs*" o "*xsd*".
- Después declaramos dos espacios de nombres con sinónimo que servirán para usar nuestros nombres dentro de la declaración del esquema. podríamos haber declarado uno de ellos sin sinónimo y los nombres sin prefijo se asignarían a es espacio de nombres. Cuando hay un solo espacio de nombres de este tipo, se suele definir sin sinónimo.
- En tercer lugar tenemos "*targetNamespace*" que define el espacio de nombre sobre el que van a definir los elementos que creemos en este esquema. Este espacio de nombres será igual a los definidos anteriormente puesto que si cualificamos elementos con unos espacios de nombres, se deberán crear sobre ellos.

La información de formato libre para un esquema está representada por dos atributos, de nombre "ID" y "version" que sirven para poder información de formato libre, aunque el primero es de tipo ID.

La información interna para la creación del esquema está representada por un par de atributos de uso avanzado que son "*finalDefault*" y "*blockDefault*". Estos atributos establecen limitaciones a los tipos de construcciones válidas para elementos y atributos.

Y la información para la construcción de documentos instancia está representada por los atributos "*elementFormDefault*" y " *attributeFormDefault*" que establece si los elementos y atributos de dichos documentos deben o no conformar con el espacio de nombres declarado en "*targetNamespace*". Los posibles valores son "*qualified*" y "*unqualified*"

La información de idioma viene dada por el atributo xml:lang, al igual que en todos los demás elementos XML (ver apéndices).

6.3.2 Uso de un esquema en un documento

El uso de un esquema en un documento se realiza mediante atributos en el elemento raíz del documento.

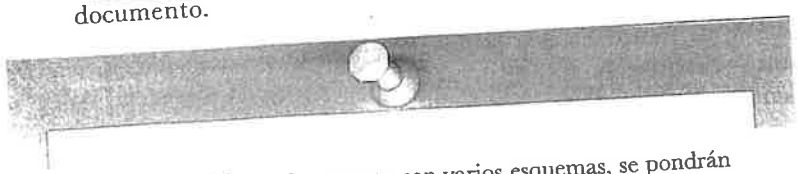
Por ejemplo:

```
<?xml version="1.0"?>
<raiz xmlns ="http://www.ejemplo.como" -
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ejemplo.como/ejemplo.xsd">
      ...
</raiz>
```

En este ejemplo vemos como se realiza la inclusión.

- Primero, se declara el espacio de nombres por defecto para el documento que coincidirá con el declarado en el esquema.
- Segundo, se declara el espacio de nombres para instancias de documentos de esquemas que es siempre el mismo y que se suele marcar con el sinónimo xsi aunque esto es opcional (cada uno puede elegir el que desee).

- Tercero, se usa el espacio de nombres xsi y el nombre "*schemaLocation*" o "*noNamespaceSchemaLocation*" para establecer donde reside el esquema que se va a usar para validar este documento.



Si se quiere validar el documento con varios esquemas, se pondrán todos los esquemas en el mismo atributo, separados por espacios.

También será preciso declarar más espacios de nombres para su uso y utilizar los elementos con los sinónimos necesarios.

Cada elemento se validará con un único esquema, dependiendo del espacio de nombres al que pertenezca. .

El siguiente ejemplo se valida con dos esquemas:

```
<?xml version="1.0"?>
<esq1:raiz xmlns:esq1 ="http://www.ejemplo1.como"
           xmlns:esq2 ="http://www.ejemplo2.como"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.ejemplo.como/ejemplo1.xsd
                               http://www.ejemplo.como/ejemplo2.xsd">

  ...
     <esq2:elemento> ...
  ...
     <esq1:elemento> ...
  ...
</esq1:raiz>
```

6.3.3 Contenido de esquema

El contenido en sí de un esquema son las definiciones estructurales que permitan definir cómo van a ser los documentos XML. Ya que los documentos representados son los definidos por el estándar, que ya representamos con DTD, sólo tendremos que establecer los elementos y atributos que van a contener.

Las piezas disponibles para crear el esquema se representan con elementos XML y disponen de opciones modeladas mediante atributos, ya que un esquema es un documento XML. Son las siguientes:

- Definiciones de elementos y grupos
- Definiciones de atributos y grupos de atributos
- Declaraciones de tipos, simples y complejos, nuevos y derivados de otros por restricción.

Una distinción interesante es la de definición, que equivale a la creación de tipos y elementos del esquema, y declaración, que significa establecer, mediante el uso de los lo definido, la estructura que tendrá el documento, sus elementos y atributos.

A continuación veremos estos tres tipos de componentes del esquema por separado e interactuando para crear los esquemas, y otros menos relevantes.

6.4 COMPONENTES DE UN ESQUEMA

Los componentes de un esquema sean del tipo que sean, se separan en dos grupos, locales y globales. La forma de identificar unos y otros es muy sencilla, cuando un componente está dentro de otro, es local a ese segundo componente y no puede usarse en ningún otro lugar. Cuando un componente es global, se coloca en el primer nivel de anidamiento dentro del elemento "*schema*" y puede ser usado a lo largo de todo el documento.

Este es un ejemplo de elementos globales y locales:

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

  <!--elemento global-->
  <xs:complexType name="CantidadType">

    ...

  </xs:complexType>

  <!--elemento global-->
  <xs:element name="Ciudad" type="xs:string"/>

  <!--elemento global-->
  <xs:complexType name="DatosCiudad">
    <xs:sequence>
                <!--elemento local-->
      <xs:element ref="Ciudad"/>      <!--uso de elemento global-->
      <xs:element name="Cantidad" type="CantidadType"/>
    </xs:sequence>
```

```
</xs:complexType>
```

En este ejemplo se utilizan los elementos que forman un esquema (algunos), y que aún no se ha presentado. El lector debe evitar fijarse en ella y quedarse sólo con el importante concepto de definición global y local.

Por último, el diagrama de la Figura 6.1 representa de forma gráfica el ámbito de aplicación o la visibilidad de una definición de tipos global y local.

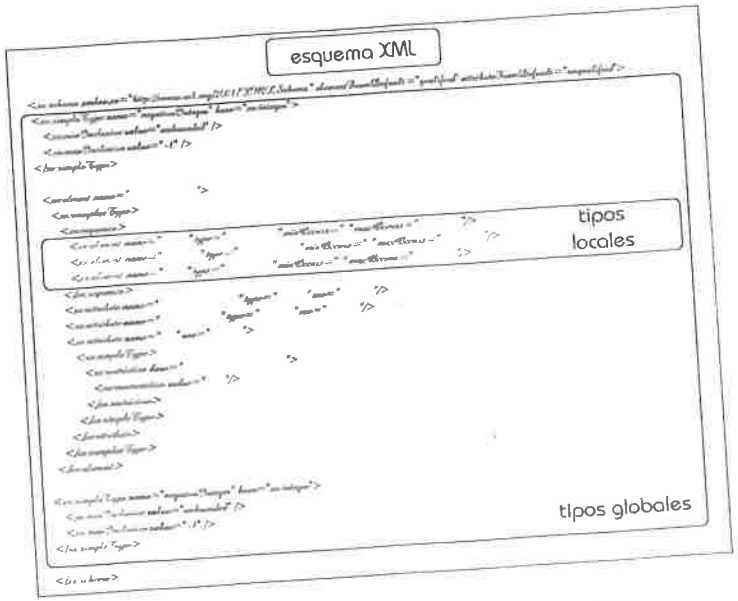


Figura 6.1: Declaración global y local en esquemas

Estos son los elementos que vamos a manejar en todos los esquemas que creemos. Los presentamos acompañados de la sintaxis inicial que usaremos para ellos:

- definición y declaración de elementos

```
<xs:element ... />

<xs:element ... >
    ...
</xs:element>
```


- atributos

```
<xsd:attribute ... />
```

- tipos simples

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

- tipos complejos

```
<xs:complexType ... >
    ...
</xs:complexType ... >
```

- documentación, comentarios e información para aplicaciones.

```
<xs:annotation>
  <xs:documentation>
                    Comentario
  </xs:documentation>
  <xs:appInfo>
          ...
  </xs:appInfo>
</xs:annotation>
```

En sucesivos apartados describiremos cada uno de estos elementos.

6.5 DOCUMENTACIÓN Y COMENTARIOS DE ESQUEMAS

En un esquema se pueden escribir comentarios que ayuden a una mejor comprensión del mismo. Los elementos se definirán global o localmente.

Para ello se aplica la siguiente sintaxis:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-Ejemplo de XML Schema-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">

...
<xs:annotation>
   <xs:documentation>
                        Comentario
   </xs:documentation>

   <xs:appInfo>
                ...
   </xs:appInfo>
</xs:annotation>
...
```

El comentario debe ir dentro del elemento "*xs:documentation*" que irá dentro de un elemento "*xs:annotation*". El texto que vaya aquí comprendido no será analizado por la aplicación y se entenderá que es documentación interna al desarrollo del esquema.

El elemento "*documentation*" puede llevar el atributo "xml:lang="es". Este atributo muestra el idioma en el que se basa el Esquema.

Además, dentro del elemento "*xs:annotation*" se puede incluir el elemento "*xs:appInfo*", que proporciona información del esquema para aplicaciones externas.

6.6 TIPOS DE DATOS

Los tipos de datos disponibles se pueden clasificar en tres categorías, primitivos, derivados y definidos por el usuario. Los tipos definidos por el usuario se pueden dividir en dos, simples y complejos.

Por otro lado, hay que distinguir entre tipos globales y locales, aunque sólo para los tipos definidos por el usuario, los tipos de datos incluidos en los esquemas, se consideran globales.

Las siguientes definiciones, aclaran el significado o características de cada uno de estos tipos:

- Tipos de datos primitivos: son aquellos que no se definen en función de otros y forman parte de la definición de los esquemas. Sirven para formar otros tipos de datos. Además nunca pueden

- Tipos de datos derivados: este tipo de datos se forman a partir de otros existentes y forman parte de la definición de esquemas.

- Tipo simple: es el tipo definido por el usuario para aquellos elementos que únicamente pueden contener texto, es decir, no pueden contener otros elementos.

- Tipo complejo: es el tipo definido por el usuario para aquellos elementos que pueden contener otros elementos. Tienden a describir la estructura de un documento más que su contenido.

- Tipo global: es aquel que se declara como hijo inmediato del elemento "*chema*": Se declara una vez y puede ser utilizado por todos los elementos que se creen.

- Tipo local: es el que se declara dentro de algún elemento y sólo tiene validez dentro de dicho elemento.

- Tipos de datos atómicos: Son aquellos cuyo contenido no se puede dividir. Por ejemplo los tipos primitivos, derivados y simples.

- Tipos de datos no atómicos: Formado por un conjunto de datos atómicos, no puede estar formado por otros datos no atómicos.

El diagrama de las Figura 6.2 representa gráficamente la relación existente entre las clases de tipos de datos. En este diagrama, las flechas indican que unos ciertos tipos de datos se definen en base a otros.

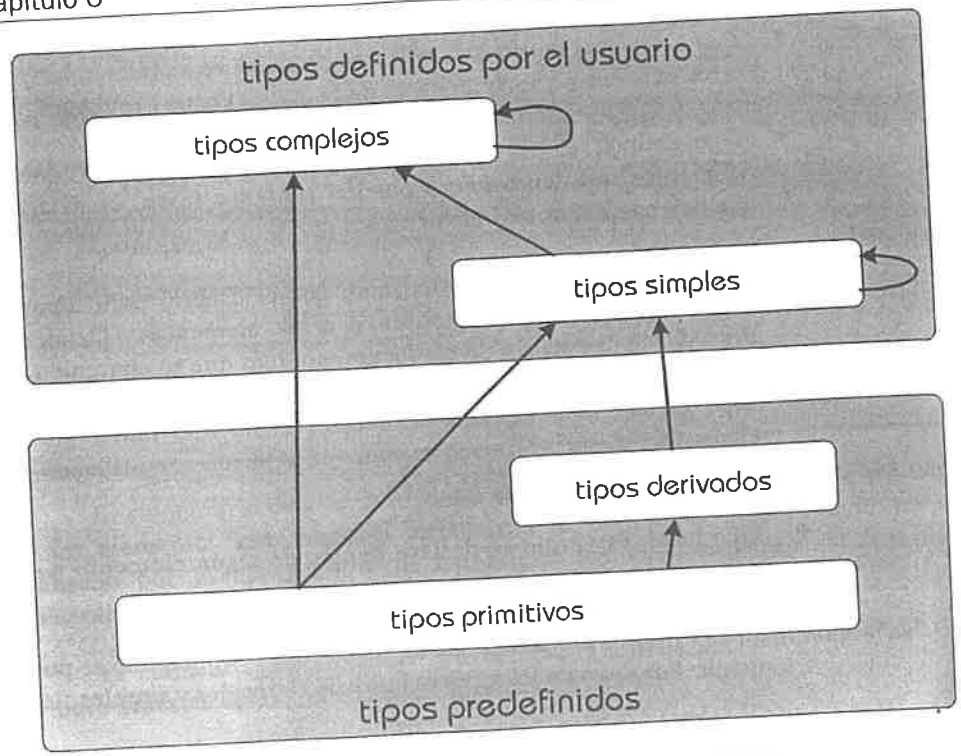


Figura 6.2: Genealogía de tipos de datos en esquemas

Los tipos de datos primitivos, forman parte de la especificación de XML Schema y no pueden cambiar si no se cambia dicha especificación.

Los tipos de datos primitivos definidos en XML esquema son los siguientes:

string	boolean	decimal	float	double
duration	time	date	gYearMonth	gYear
gMonthDay	gDay	gMonth	hexBinary	base64Binary
dateTime	anyURI	QName	NOTATION	

Los tipos de datos derivados, también forman parte de la especificación de XML Schema y no pueden cambiar si no se cambia dicha especificación.

Los tipos de datos derivados definidos en XML esquema son los siguientes:

normalizedString	token	language	NMTOKEN
NMTOKENS	Name	NCName	ID
IDREF	IDREFS	ENTITY	ENTITIES
integer	nonPositiveInteger	negativeInteger	long
int	short	byte	nonNegativeInteger
unsignedLong	unsignedInt	unsignedShort	unsignedByte
positiveInteger			

6.6.1 Estructura de un tipo de datos

Los tipos de datos se componen de las siguientes partes:

- Espacio de valores → posibles valores de dicho tipo de dato.
- Espacio léxico → representaciones de los valores del espacio de valores.
- Facetas → propiedades del tipo.

La diferencia entre espacio léxico y espacio de valores, es que en el espacio de valores, tenemos el concepto representado, y en el espacio léxico la representación que usamos en XML Schema. Las representaciones que se han hecho son no binarias, para facilitar la lectura directa. Así, por ejemplo el valor 34 tiene una representación léxica como la cadena de caracteres "34". Esto encaja perfectamente con las normas XML y con el contenido de los elementos que, en el fondo, son siempre cadenas de caracteres.

Las facetas muestran las propiedades de un tipo de datos concreto. Existen cinco fundamentales y doce opcionales o de parametrización. Las facetas dependen del tipo de datos al que vaya referido.

En los esquemas tenemos estas cinco facetas fundamentales:

- Igualdad → Se pueden comparar los valores (numéricos o no) para determinar si son iguales o no. Hay que tener en cuenta que se distingue entre mayúsculas y minúsculas, esto es importante para valores no numéricos (XML no es lo mismo que Xml).

- Orden → Se puede aplicar a valores numéricos y no numéricos (se toma el valor numérico codificado). Sirve para determinar la relación existente entre varios valores. Es importante determinar el sistema de codificación ya que cambia su valor numérico de un sistema de codificación a otro.

- Límites → Sirve para determinar los valores superiores e inferiores de un tipo de dato. Las posibilidades son:

  - minInclusive: Valor inferior incluido.
  - maxInclusive: Valor superior incluido.
  - minExclusive: Valor inferior no incluido.
  - maxExclusive: Valor superior no incluido.

- Cardinalidad → Cantidad de valores que se permiten en un tipo de dato. Dicho valor puede ser finito o infinito (unbounded), 0 ó 1.

- Numérico / No numérico → Un tipo de dato será numérico si su valor pertenece a algún sistema numérico.

Además tenemos estas facetas de limitación:

- length, minlength, maxlength
- pattern
- enumeration
- minExclusive, maxExclusive, minInclusive, maxInclusive.
- whiteSpace
- totalDigits, fractionDigits

6.6.1.1 TIPOS DE DATOS PREDEFINIDOS DISPONIBLES

La siguiente tabla incluye tanto los tipos primitivos como los tipos derivados disponibles en la especificación de XML esquema.

Elemento	Contenido
string	El elemento contiene una cadena de caracteres.
normalizedString	El elemento contiene una cadena de caracteres, pero los caracteres retorno línea, tabulador y retorno de carro son convertidos a caracteres de espacio en blanco antes del procesamiento del Esquema.
Token	Igual que "normalizedString", pero los espacios en blanco adyacentes son comprimidos a un sólo espacio en blanco, y los espacios en blanco de delante y detrás son eliminados.
Byte	El elemento va a contener un byte, el valor comprende entre -1 y 126. Además se pueden utilizar distintas expresiones léxicas. Por ejemplo: se podría expresar 100 ó 10E2.
unsignedbyte	Igual que el anterior, pero los valores vienen comprendidos entre 0 y 126.
base64Binary	El elemento contiene un número base 64.
hexbinary	El elemento puede contener un número hexadecimal (base 16).
integer	El elemento contiene un número entero, el valor comprende entre -126789, -1, 0, 1, 126789.
positiveInteger	El contenido de este elemento puede contener un número entero positivo, el rango de valores comprende entre 1 y 126789.
negativeInteger	El contenido del elemento puede ser un número entero negativo, el rango de valores está entre -126789 y -1.
nonNegativeInteger	El contenido de este elemento es un número entero positivo ó 0, el valor comprende entre 0, 1 y 126789.
nonPositiveInteger	El contenido de este elemento es un número entero negativo, incluyendo el 0. El rango corresponde entre -126789 y 0.
int	El contenido de este elemento es un número entero, el rango cmprende entre -1 y 126789675.
unsignedInt	El contenido de este elemento es un número entero positivo cuyo rango comprende entre 0 a 1267896754.
long	El contenido del elemento a a ser un dígito comprendido entre -1 y 12678967543233.
unsignedlong	El contenido del elemento corresponde entre 0 y 12678967543233.
short	El contenido del elemento puede estar entre 0 y 12678.
unsignedshort	El contenido del elemento puede comprender entre 0 y 12678.
decimal	El contenido de este tipo de elemento es un número decimal,

	con dos dígitos en la parte fraccionaria (en caso de que no sea entero).
float	El contenido de este elemento es un número en punto flotante icluyendo el 0 y -0. Además existe el concepto de "INF" para infonito positivo, "-INF" para el infinito negativo, y "NaN" para indicar que no es un número.
double	Igual que el anterior pero para una precisión de 64 bytes.
boolean	El contenido de este tipo de elementos puede ser 0 ó 1.
time	El elemento va a ser la hora del día.
dateTime	El elemento va a indicar una hora del día. El formato es: 1999-05-31T13:20:00.000-05:00 para expresar la fecha 31 de Mayo de 1999 a las 1.20pm Hora Estándar de la Costa Este que va 5 horas por detrás de la Hora Universal.
duration	Indica un periodo de tiempo, por ejemplo P1Y2M3DT10H30M12.3S para expresar 1 año, 2 meses, 3 días, 10 horas, 30 minutos, 2 meses, 3 días, 10 horas, 30 minutos, y 12.3 segundos.
date	Indica una fecha, el formato es "yyyy-mm-dd".
gMonth	El contenido de este elemento indica un mes, el formato es --MM--
gyear	El contenido de este elemento va as er un año. El formato es el siguiente: YYYY.
gYearMonth	El contenido de este elemento va a ser mes y año. El formato es el siguiente: YYYY-MM.
gDay	El contenido de este elemento es un día del mes. El formato es: --- DD.
GMonthDay	El contenido de este elemento es un dia y un mes. El formato es: --MM-DD.
name	El contenido de este elemento es un tipo de nombre de XML 2.0.
anyURI	El contenido de este elemento será una dirección http.
language	Valores válidos para xml:lang según está definido en XML 1.0.
ID	El contenido de este elemento será un atributo ID de XML 1.0.
IDREF	El contenido de este elemento será un atributo IDREF de XML 1.0.

IDREFS	El contenido de este elemento será un atributo IDREFS de XML 1.0.
ENTITY	El contenido de este elemento será un atributo ENTITY de XML 1.0.
ENTITIES	El contenido de este elemento será un atributo ENTITIES de XML 1.0.
NOTATION	El contenido de este elemento será un atributo NOTATION de XML 1.0.
NMTOKEN	El contenido de este elemento será un atributo NMTOKEN de XML 1.0.
NMTOKENS	El contenido de este elemento será un atributo NMTOKENS de XML 1.0.
AnySimpleType	
QName	Debe ser un nombre calificado según la recomendación del estándar XML 1.0.
NCName	Debe ser un nombre que cumpla con los requisitos de un elemento NCName de la especificación del Espacio de Nombres. Es un nombre XMl sin dos puntos (":")
AnyType	No restringe el contenido en modo alguno.

Tabla 6.3: Tipos de datos disponibles en esquemas

Notas:

- Los caracteres nueva línea, tabulador y retorno de carro en un tipo "normalizedString" son convertidos a caracteres de espacio en blanco antes del procesamiento del Esquema.
- Los espacios en blanco adyacentes son comprimidos a un solo espacio en blanco, y los espacios en blanco primero y último son eliminados.
- Un número se puede representar como en los ejemplos siguientes: 100 ó 10E2
- El prefijo "g" indica calendario Gregoriano.
- Para mantener la compatibilidad entre Schema y DTD los tipos simples "ID", "IDREF", "IDEREFS", "NMTOKEN" y "NMTOKENS" deberán utilizarse únicamente en atributos.

6.6.2 Definición de tipos simples por restricción

A continuación veremos, una por una, la utilización de las facetas de los tipos simples. Dicha utilización implica la creación de un nuevo tipo de datos, un tipo de datos simple definido por el usuario.

Para ello tendremos siempre la sintaxis:

```
<xs:simpleType name="nuevo_tipo" final="restricciones">
   <xs:restriction base="tipo_base">
      < ... facetas que se modifican ... />
   </xs:restriction>
</xs:simpleType>
```

En esta sintaxis tendremos siempre el nombre que le damos a nuestro nuevo tipo, las restricciones que imponemos que incluyen el tipo a partir del cual derivamos y las facetas que modificamos para crear nuestro tipo.

También tenemos un atributo en el elemento "*simpleType*" que sirve para evitar posteriores derivaciones de tipos a partir de este. El atributo es "*final*" y puede tomar los valores #all, restriction, list y union. Cada uno de estos valores indica el tipo de derivación que prohibimos, salvo #all que prohíbe todas. Podemos enumerar separadas por espacio si queremos prohibir dos de ellas.

6.6.2.1 LENGTH, MINLENGTH, MAXLENGTH

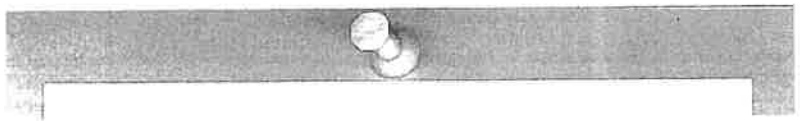
El valor tiene que ser un valor entero no negativo. Determina el número de caracteres o dígitos de un tipo de dato.

Por ejemplo, para declarar un tipo de dato "string" "abecedario" de cinco caracteres.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:simpleType name="abecedario">
      <xs:restriction base="xs:string">
         <xs:minLength value="5" fixed="false"/>
         <xs:maxLength value="5" fixed="true">
            <xs:annotation>aquí hemos usado la opción de fijar el valor del
atributo de longitud máxima </xs:annotation>
         </xs:maxLength>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="abecedario">
      <xs:restriction base="xs:string">
         <xs:length value="5" fixed="true"/>
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```


Este ejemplo también ilustra el uso de comentarios para diversos elementos (se pueden incluir para todo elemento) y el uso del atributo fixed que establece que una faceta ya modificada no lo pueda ser en posteriores derivaciones.



La utilidad de prohibir la posterior modificación de una faceta ya modificada es fundamental.

Si defino un tipo de tamaño máximo 10, no tiene sentido derivar de él para cambiar ese tamaño. Para eso se debería derivar del original.

Si derivo de un tipo ya modificado es para añadirle más modificaciones, no para cambiar las que estableció en su momento

6.6.2.2 PATTERN

Restricciones sobre los posibles valores que puede tener un tipo de datos mediante patrones con los que debe conformar el valor candidato.

Las expresiones que se permiten en los esquema se construyen con los elementos que se resumen en la siguiente tabla:

^	Indica el comienzo de una línea
$	Indica el final de una línea
*	Repite carácter precedente 0 o más veces

+	Repite carácter precedente 1 o más veces
?	Repite carácter precedente 0 o 1 vez
.	Acepta cualquier carácter simple excepto *newline*
(x)	Acepta 'x' y recuerda el contenido para posteriores sustituciones
X|y	Acepta con 'x' o 'y'
{n}	Cuando n es positivo, acepta exactamente n ocurrencias del carácter precedente
{n,}	Cuando n es positivo, acepta al menos n ocurrencias del carácter precedente
{n,m}	Cuando n y m son positivos, exige al menos n y máximo m ocurrencias del carácter precedente
[xyz]	Un conjunto de caracteres. Acepta cualquiera de los caracteres incluidos entre paréntesis. Se pueden también incluir rangos usando el *hyphen (-)*
[^xyz]	La negación o el complemento de un conjunto de caracteres. Acepta cualquier carácter que no esta incluido en el conjunto.
[\b]	Acepta un backspace
\b	Acepta los limites de una palabra, tal como espacios o *newline*
\B	Acepta otros caracteres, no los limites de una palabra
\cX	Donde X es un carácter de control, Acepta un carácter de control en un string
\d	Acepta dígito. Equivalente a [0-9]
\D	Acepta cualquier carácter que no sea un dígito. Equivalente a [^0-9]
\f	Acepta un *formfeed*)
\n	Acepta un avance de línea
\r	Acepta un retorno de carro
\s	Acepta un carácter espacio en blanco, incluye espacio, tabulador, form feed y linefeed
\S	Acepta un carácter que no sea un espacio en blanco
\t	Acepta un tabulador
\v	Acepta un tabulador vertical
\w	Acepta un carácter alfabético. Equivale a [A-Za-z0-9_]
\W	Acepta caracteres no alfabéticos. Equivale a [Â-Za-z0-9_]
\n	Donde n es positivo. Es una referencia al ultimo string que calzo en n-ésimo paréntesis.

Tabla 6.2. Expresiones permitidas en el estándar XML

El siguiente ejemplo ilustra la utilización de los patrones para aceptar un número IP válido. Se puede observar la flexibilidad y la complejidad de estas expresiones.

```
<xs:simpleType name="IP">
```



```
      <xs:pattern value="((([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-
5])\.){3}([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])"/>
    </xs:restriction>
  </xs:simpleType>
```

6.6.2.3 ENUMERATION

Limita un tipo de datos a un conjunto de valores. Si se repite un valor en la enumeración, éste no será válido.

Por ejemplo, la siguiente declaración, establece un tipo que únicamente soporte algunos países europeos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified"/>
  <xs:simpleType name="paises_europeos">
    <xs:restriction base="xs:string">
      <xs:enumeration value="España"/>
      <xs:enumeration value="Portugal"/>
      <xs:enumeration value="Italia"/>
      <xs:enumeration value="Francia"/>
    </xs:restriction>
  </xs:simpleType>
</schema>
```

6.6.2.4 MINEXCLUSIVE, MAXEXCLUSIVE, MININCLUSIVE, MAXINCLUSIVE

Valores máximos y mínimos permitidos en un tipo de dato. "Inclusive", indica que dicho valor puede ser utilizado, mientras que si se declara "Exclusive" no se podrá utilizar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified"/>
  <xs:simpleType name="rango_numero1">
    <xs:restriction base="xs: integer ">
      <xs:minExclusive value="1"/>
      <xs:maxExclusive value="10"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="rango_numero2">
```

```
      <xs:restriction base="xs:integer">
         <xs:minInclusive value="1"/>
         <xs:maxInclusive value="10"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

6.6.2.5 WHITESPACE

Esta faceta establece para un tipo de texto el tratamiento a realizar con los espacios en blanco sobrantes. Toma tres valores: preserve (no cambiar espacios, ni retornos, ni tabuladores), replace (cambia cada tabulador y retorno por un espacio), collapse (cambia series sucesivas de espacios por uno sólo y elimina los espacios sobrantes por izquierda y derecha del contenido).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified"/>
   <xs:simpleType name="espacios">
      <xs:restriction>
         <xs:whiteSpace value="preserve"/>
      </xs:restriction>
   </xs:simpleType>
 </xs:schema>
```

6.6.2.6 TOTALDIGITS, FRACTIONDIGITS

"*totalDigits*" indica el número máximo de dígitos para un número. "*fractionDigits*" hace referencia al número máximo de dígitos que se puede utilizar en la parte decimal.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified"/>
   <xs:simpleType name="rango_numero3">
      <xs:restriction base="xs:decimal">
         <xs:totalDigits value="3"/>
         <xs:fractionDigits value="2"/>
         <xs:whiteSpace value="preserve"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```


6.6.2.7 FACETAS APLICABLES A CADA TIPO

No todas las facetas se pueden aplicar a todos los tipos. la siguiente tabla presenta un resumen de las facetas que se pueden modificar para cada tipo.

Tipo Simple	*Lng*	*mnL*	*mxL*	*ptr*	*Enm*	*whS*	*mxI*	*mxE*	*mnI*	*mnE*	*tID*	*frD*
String	■	■	■	■	■	■						
normalizedString	■	■	■	■	■	■						
Token	■	■	■	■	■	■						
Byte				■	■	■	■	■	■	■	■	■
unsignedByte				■	■	■	■	■	■	■	■	■
base64Binary	■	■	■	■	■	■						
hexBinary	■	■	■	■	■	■						
integer				■	■	■	■	■	■	■	■	■
positiveInteger				■	■	■	■	■	■	■	■	■
negativeInteger				■	■	■	■	■	■	■	■	■
NonNegativeInteger				■	■	■	■	■	■	■	■	■
NonPositiveInteger				■	■	■	■	■	■	■	■	■
Int				■	■	■	■	■	■	■	■	■
unsignedInt				■	■	■	■	■	■	■	■	■
long				■	■	■	■	■	■	■	■	■
unsignedLong				■	■	■	■	■	■	■	■	■
short				■	■	■	■	■	■	■	■	■
unsignedShort				■	■	■	■	■	■	■	■	■
decimal				■	■	■	■	■	■	■		
float				■	■	■	■	■	■	■		
double				■	■	■	■	■	■	■		
boolean												
time				■	■	■	■	■	■	■		
dateTime				■	■	■	■	■	■	■		

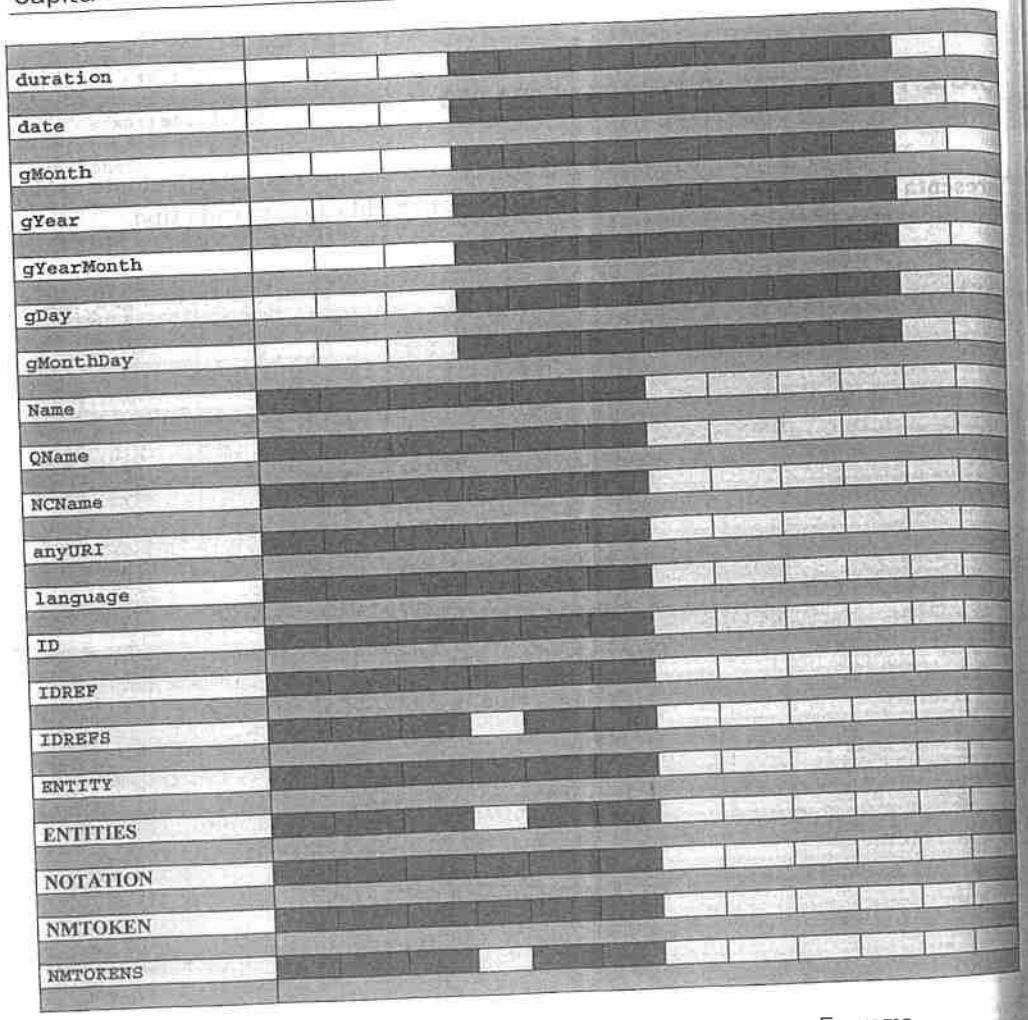


Tabla 6.4: Relación de facetas para cada tipo de elemento en un Esquema

Donde:

- lng. Es la abreviatura de Length.
- mnL. Es la abreviatura de MinLength.
- mxL. Es la abreviatura de MaxLength.
- ptr. Es la abreviatura de Pattern.

- Enm. Es la abreviatura de Enumaration.
- whS. Es la abreviatura de WhiteSpace.
- mxI. Es la abreviatura de MaxInclusive.
- mxE. Es la abreviatura de MaxExclusive.
- mnI. Es la abreviatura de MinInclusive.
- mnE. Es la abreviatura de MinExclusive.
- tID. Es la abreviatura de TotalDigits.
- frD. Es la abreviatura de FractionDigits.

6.6.3 Definición de tipos simples lista y unión

Estos tipos también se consideran simples, se definen a partir de otros tipos simples que no sean unión ni lista. El motivo de que estos tipos sean simples es que están pensados para contener un valor, aunque este es complejo, pero no a otros elementos, que es la definición que manejamos para simples y complejos.

Creación de tipos de datos lista

Este tipo de dato permite a un elemento que lo utilice contener varios elementos del mismo tipo.

Se utiliza la sintaxis:

```
<xsd:list base="xsd:tipo"/>
```

donde "tipo" corresponde al tipo de datos base para realizar la lista.

Este es un ejemplo de creación de una tipo lista:

```
<xs:simpleType name="lista">
```

```
    <xs:list itemType="xs:int"/>
  </xs:simpleType>

  <xs:simpleType name="lista2">
     <xs:restriction base="lista">
        <xs:length value="5"/>
     </xs:restriction>
  </xs:simpleType>
```

En este caso definimos dos tipos. El primero es una lista de enteros, y el segundo redefine la lista anterior en un nuevo tipo que debe tener 5 elementos.

Cuando se redefine un tipo lista ya definido, las facetas que se puede modificar son:

- "*length*" → Número de componentes de la lista.
- "*minlength*" → Número mínimo de componentes de la lista.
- "*maxlength*" → Número máximo de componentes de la lista.
- "*enumeration*" → Limitación de valores que puede contener dicho elemento.
- "*whiteSpace*" → Gestión de los espacios sobrantes.
- "*pattern*" → Plantilla a cumplir por los elementos

Creación de tipos de datos unión

Este tipo de dato permite contener una o más instancias de distintos tipos. La idea es permitir que el contenido de un elemento sea indeterminado e tipo, que pueda ser de varios tipos, los que forman la unión.

Para ello se utiliza el formato:

```
<xsd:union memberTypes="elemento1 elemento2 ..."/>
```

O, como vemos en este ejemplo doble, uno más complejo que permite declarar los tipos que vamos a usar en su interior (esto también se puede hacer con listas):


```
<xs:simpleType name="unaunion">
   <xs:union memberTypes="paises_europeos rango_numero2"/>
</xs:simpleType>

<xs:simpleType name="otraunion">
   <xs:union>
      <xs:simpleType name="espacios">
         <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="rango_numero3">
         <xs:restriction base="xs:decimal">
            <xs:totalDigits value="3"/>
            <xs:fractionDigits value="2"/>
            <xs:whiteSpace value="preserve"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="string">
         <xs:restriction base="xs:string"/>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>
```

Las tres definiciones de tipos realizadas dentro de la propia unión son definiciones locales.

La tercera definición no establece restricción sobre el tipo xs:string, tan sólo sirve para poder utilizarlo una vez que se están usando estas definiciones locales.

6.7 DECLARACIÓN DE ELEMENTOS

Ya hemos descrito la creación de tipos simples y los tipos existentes. La creación de tipos complejos implica que un elemento contiene subelementos, por tanto implica la definición previa de esos elementos. Ahora veremos como se declaran elementos.

Toda declaración de elementos, local o global, simple o complejo tiene el mismo formato que es:

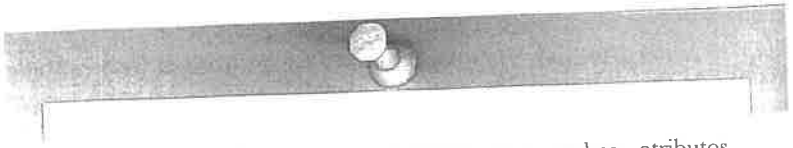
```
<xs:element name="nombre" type="xs:tipo" atributos/>
```

Si se trata de un elemento que usa un tipo ya definido, donde:

- "nombre" → Nombre del elemento.
- "xsd:tipo" → Es el tipo de datos que va a contener dicho elemento. Atributos → Se pueden añadir atributos para fijar un valor por defecto, o para indicar el número de veces que puede aparecer dicho elemento.

Las posibilidades son las siguientes:

- minOccurs. Indica el número de ves que debe aparecer el elemento como mínimo. Si se pone "0" será opcional.
- maxOccurs. Indica el número de veces que debe aparecer el elemento como máximo. Debe ser mayor o igual que minOccurs, ilimitado se pone "unbounded".
- fixed. Valor por defecto que se le pone a un elemento cuando está vacío.
- default. Asegurar que el elemento tiene un valor fijado.
- ref. Establece que esta definición es una referencia a otra de un elemento global.
- Y otras: abstract, block, nilllable, form, final y sustitutionGroup; cuyos significados veremos más adelante.



No se puede declarar un elemento con ambos atributos simultáneamente.

La diferencia entre ambos es la siguiente:

- "fixed" sólo establece su contenido si el atributo aparece en el documento XML. Si se omite, no se establece el contenido.

- "default" establece su valor cuando se omite del documento XML

Ejemplo de declaración de un elemento "*texto*" que tenga como valor fijo "Esto es un texto predeterminado".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
<xsd:element name="texto" type="xsd:string"
          fixed="Esto es un texto predeterminado"/>
</xs:schema>
```

Ejemplo de declaración de un elemento "*texto*" que ponga como valor por defecto "Esto es un texto predeterminado".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
<xsd:element name="texto" type="xsd:string"
          default="Esto es un texto predeterminado"/>
</xs:schema>
```

Aunque también se puede utilizar un tipo que se defina en el mismo lugar, denominado tipo personalizado anónimo, porque no se le da nombre y no puede ser utilizado en ningún otro lugar, se define, se usa y se olvida. Por ejemplo:

```
<xs:element name="nombre" atributos>
  <xs:simpleType name="espacios">
     <xs:restriction>
        <xs:whiteSpace value="preserve"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

En este caso se define el tipo dentro del elemento y el atributo "*type*" de "*element*" no se puede usar pues sería una declaración doble de tipo. El tipo usado puede ser simple o complejo.

6.7.1 Declaración local y global, raíz de documento

La diferencia ya conocida entre definición global y local, tiene matices especiales en el caso de elementos. Un esquema, a diferencia de una DTD no está

obligado a establecer una raíz de documento como tal. En un esquema, todo elemento global es susceptible de ser raíz de un documento, ya que un validador, lo que hará será buscar en el esquema la existencia de la raíz del documento que está procesando. Si la encuentra no se preocupará de que haya más o no.

Si queremos que un esquema tenga una raíz definida por obligación, tendremos que definir un único elemento global.

Por otro lado, como los elementos globales son susceptibles de ser raíz de documento, no tiene sentido que puedan aparecer varias veces, por este motivo, algunas facetas están prohibidas como son minOccurs y maxOccurs. Si estos elementos globales no lo fueran, es decir, fueran usados por otros elementos, en ese momento se podrán especificar estos valores, de esta forma no se pierde flexibilidad.

Por otro lado, si queremos tener reutilización mediante definiciones globales pero un único elemento raíz, definiremos globalmente todos los tipos que necesitemos, simples y complejos, y un único elemento global. De esta forma tendremos un único elemento pero reutilización por definición de tipos. Esto está justificado porque la única forma de tener un elemento compuesto de otros es definir un tipo, ya sea global o anónimo (local).

Para ilustrar estos conceptos, tenemos en las Figuras 6.2, 6.4, 6.5 y 6.6 diagramas que representan las posibilidades de creación de esquemas combinando los tipos de configuraciones de elementos y tipos.

En cada una de las figuras, tenemos descrito un tipo de documento, presentando cuatro. Dos de ellos tienen un único elemento raíz, y dos de ellos varios elementos raíz. También dos de ellos tienen definiciones de tipos locales y dos de ellos definiciones globales. No se puede decir que ninguna de las estructuras sea mejor, cada una tiene sus características y utilidad:

- Una raíz + def. globales: único tipo de documento pero complejo que aconseja reutilizar definiciones de tipos globales.
- Una raíz + def. locales: único tipo de documento y sencillo, seguramente no será necesario redefinir tipos simples.
- Varias raíces + def. globales: admite varios tipos de documentos, complejos y con necesidad de reutilización de tipos para abordar la complejidad.
- Varias raíces + def. locales: varios tipos de documentos pero sencillos, existe reutilización pero poca. También puede ser un

tipo de documento principal y necesidad de usar fragmentos validables del mismo.

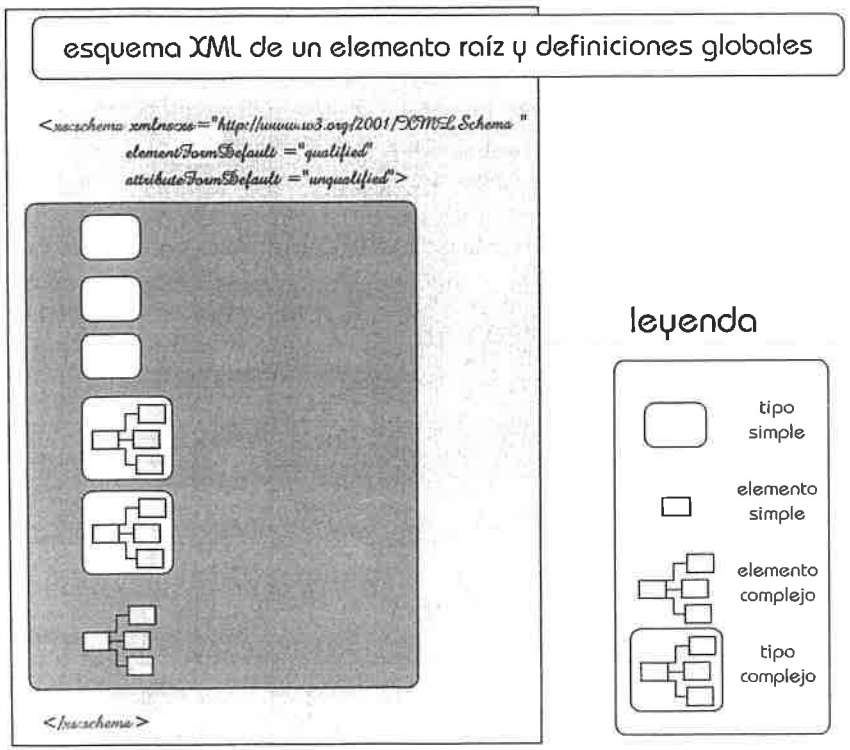


Figura 6.3: Esquema de un elemento raíz y definiciones globales

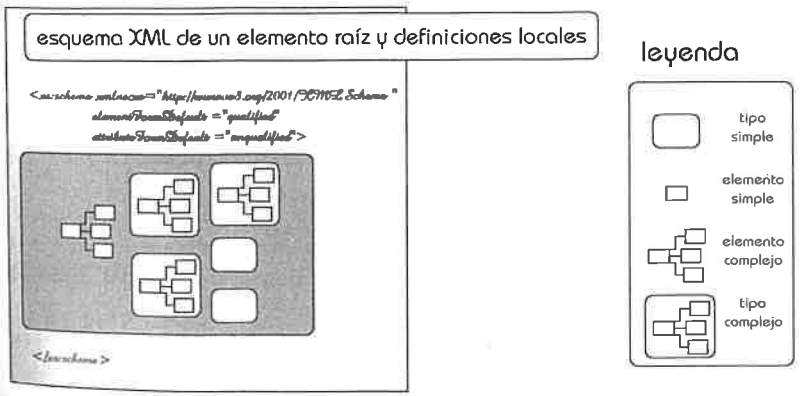


Figura 6.4: Esquema de un elemento raíz y definiciones locales

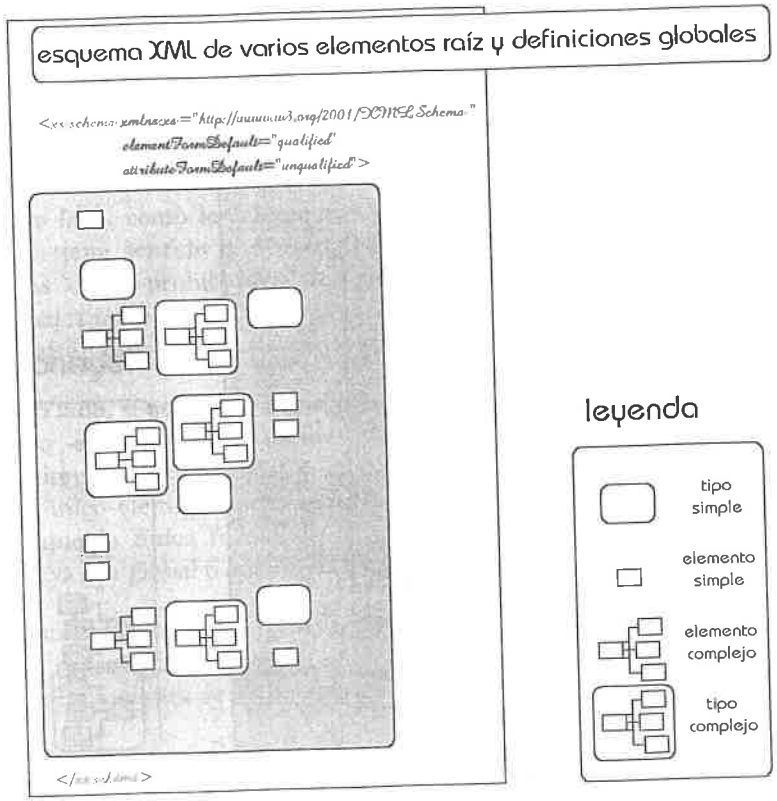


Figura 6.5: Esquema de varias raices definiciones globales

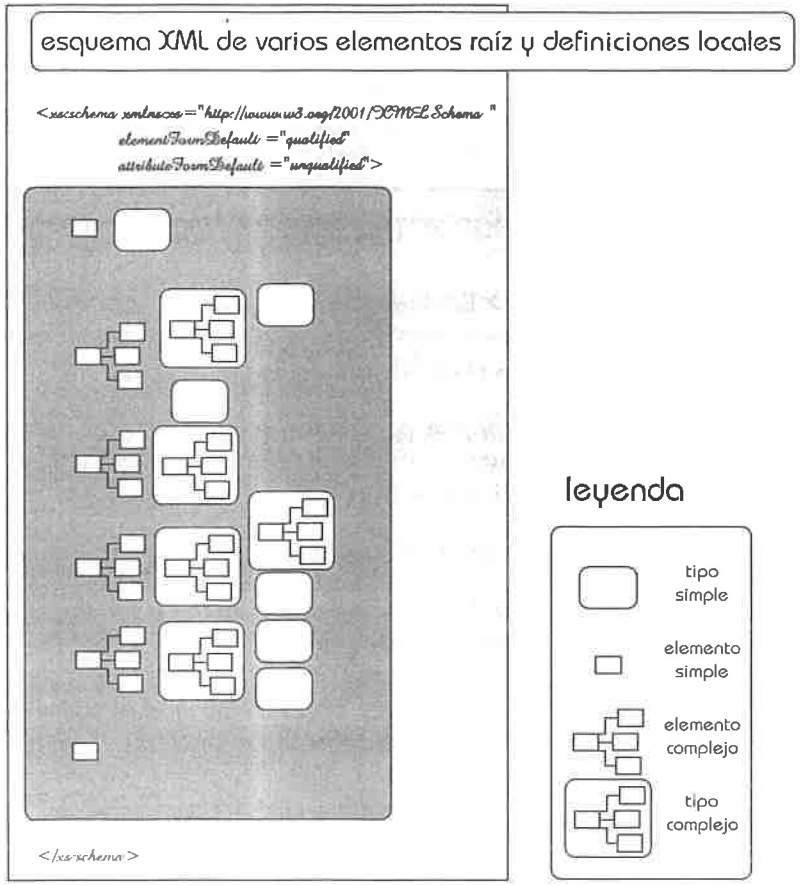


Figura 6.6: Esquema de varias raíces y definiciones locales

6.8 DECLARACIÓN DE ATRIBUTOS

Un atributo es como un elemento simple, es decir, no puede contener otros elementos a atributos. A su vez, un atributo sólo puede declararse una vez en cada elemento, por lo que no puede acompañarse en su definición de los atributos "minOccurs" y "maxOccurs".

La sintaxis para definir un atributo es:

```
<xs:attribute name="nombre" type="xs:tipo" use="tipo_atributo"
              default="valor" fixed="valor" >
```

Donde:

- "nombre" → Es el nombre del atributo.
- "tipo" → Es el tipo de contenido del atributo.
- "tipo_atributo" → Es uno de los siguientes valores:
  - optional -> Si es opcional en el elemento.
  - required -> Si es obligatorio en el elemento.
  - prohibited -> Si está prohibido su uso en el elemento, no se tomaría como válido.
- "fixed" → Se establece el contenido si el atributo aparece en el documento XMl. Si se omite, no se establece el contenido.
- "default" → Se determina cual sería el valor del atributo si se omite del documento XML.

Los atributos "fixed" y "default", al igual que en el caso de los elementos, no se pueden incluir ambos en la misma declaración.

Los atributos se pueden definir local y globalmente. Además, disponemos de la posibilidad de modificar el tipo de datos sobre el que se define el atributo mediante una definición de tipo anónima:

```
<xs:attribute name="nombre2">
   <xs:simpleType>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
</xs:attribute>
```


Los atributos definidos sirven como base para la definición de tipos complejos. Un atributo sólo puede incluirse en un elemento mediante la creación de un tipo complejo. El atributo se definirá localmente al tipo complejo o globalmente y luego se reutilizará en el tipo complejo.

6.9 DEFINICIÓN DE TIPOS DE DATOS COMPLEJOS

Se entiende por tipo de datos complejo a todo aquel que puede contener elementos o atributos. Para declarar un elemento del tipo complejo se utiliza el elemento "complexType".

La declaración de un tipo complejo posee atributos miliares a los de una declaración de tipo simple y se realiza mediante el elemento "*complexType*":

```
<xs:complexType name="productos">
   <xs:sequence>
      <xs:element name="producto-a" minOccurs="0" maxOccurs="unbounded">
      </xs:element>

      <xs:element name="producto-b" minOccurs="0" maxOccurs="unbounded">
      </xs:element>

      <xs:element name="producto-c" minOccurs="0" maxOccurs="unbounded">
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

La declaración consta de los siguientes atributos posibles:

- name → Corresponde con el tipo de dato.
- abstract → Este debe ser del tipo "bolean". Si su valor es "true" el tipo definido no podrá aparecer como tipo de dato en una declaración de elemento ni de atributo. El valor por defecto es "false".
- block → se verá más adelante en este capítulo, en el apartado sobre grupos de sustitución.
- final → Indica si el elemento se va a poder usar como tipo base para otro tipo de elemento. Los valores permitidos son:
  - #all. No se permiten otras derivaciones.

- extension. Se puede derivar otro tipo pero no por el método "extension".
- restriction. Se puede derivar otro tipo pero no por el método "restriction".

- mixed → Indica si el elemento va a contener elementos y texto entremezclados. Los valores posibles son "true" y "false".

Dentro de la declaración del tipo tendremos varias posibilidades para establecer los elementos que va a incluir. En el ejemplo anterior hemos usado "sequence". Disponemos de los siguientes:

- group:
- sequence:
- choice:
- all:

La idea es que creamos grupos de elementos para incluirlos dentro del tipo complejo. Estos grupos pueden tener nombre o ser anónimos. Si van a tener nombre usamos "*group*" para crear el grupo de forma global y luego lo usamos en el tipo, si no van a tener nombre, lo usamos directamente en el tipo.

Antes de analizar cada uno por separado, veamos como se usa un elemento global definido previamente.

6.9.1 Uso de elementos globales ya definidos

Los componentes que se han definido globalmente pueden ser invocados posteriormente. Se utiliza el atributo "*ref*", para hacer referencia a un elemento declarado de forma global.

Cuando queremos usar un componente global del tipo que sea, creamos un componente local del mismo tipo pero en lugar de especificar su nombre, utilizamos el atributo ref para indicar que es un uso de un componente global ya definido. Por ejemplo:

```
<xs:attribute name="nombre" type="xs:string" use="required" default="valor"/>
```



```
<xs:attribute ref="nombre"/>
```

Esto es válido para elementos, atributos y grupos de elementos.

Una vez que se ha incluido la referencia al elemento global, se pueden añadir algunos parámetros que completen la definición del componente.

- En el caso de los elementos, se pueden incluir los atributos "minOccurs" y "maxOccurs" que en la definición global no se pudieron incluir como ya se vio.
- En el caso de los atributos no se puede utilizar "use" en la definición global pero sí en la referencia (indica si el atributo es opcional, obligatorio).

6.9.2 Grupos de elementos

Los grupos se crean con nombre o anónimos, lo que equivale a crearlos globales y locales.

Si se crean con nombre se usará la siguiente sintaxis:

```
<xs:group name="hola">
   <xs:sequence>
      <xs:element name="Nombre" type="xs:string"/>
      <xs:element name="Edad" type="xs:decimal"/>
      <xs:element name="Sexo" type="xs:string"/>
      <xs:element name="Estado_Civil" type="xs:string"/>
   </xs:sequence>
</xs:group>
```

Y luego se hará referencia al grupo dentro del tipo complejo donde se quiera utilizar.

El contenido de un grupo es un conjunto de elementos que tiene una cierta organización. Los elementos de agrupan en tres organizaciones distintas, sequence, choice y all, con los siguiente significados:

sequence

Con este elemento se define un grupo que contiene elementos en un orden predeterminado y que deben aparecen en dicho orden.

Ejemplo:

```
<xs:sequence>
       <xs:element name="Nombre" type="xs:string">
       </xs:element>
       <xs:element name="Edad" type="xs:decimal">
       </xs:element>
       <xs:element name="Sexo" type="xs:string">
       </xs:element>
       <xs:element name="Estado_Civil" type="xs:string">
       </xs:element>
</xs:sequence>
```

Su equivalente en una DTD sería:

```
<!ELEMENT Ficha (Nombre, Edad, Sexo, Estado_Civil)>
<!ELEMENT Edad (#PCDATA)>
<!ELEMENT Sexo (#PCDATA)>
<!ELEMENT Estado_Civil (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

choice

Con este elemento se define un grupo de elementos de entre los que se deberá elegir uno.

```
   <xs:choice>
      <xs:element name="Menor10" type="xs:string"/>
      <xs:element name="entre10y20" type="xs:string"/>
      <xs:element name="entre20y30" type="xs:string"/>
   </xs:choice>
```

Su equivalente en una DTD sería:

```
<!ELEMENT Ficha (Menor10 | entre10y201 | entre20y30)>
<!ELEMENT Menor10 (#PCDATA)>
<!ELEMENT entre10y20 (#PCDATA)>
<!ELEMENT entre20y30 (#PCDATA)>
```


all

Con este elemento se define un grupo de elementos de los cuales podrán aparecer todos en el documento, en cualquier orden y siendo opcionales, pero sólo apareciendo una vez. Es decir, cada elemento aparece una o ninguna vez y en el orden que quieran.

```
  <xs:all>
     <xs:element name="Menor10" type="xs:string"/>
     <xs:element name="entre10y20" type="xs:string"/>
     <xs:element name="entre20y30" type="xs:string"/>
  </xs:all>
```

Su equivalente en DTD es muy engorroso y difícil de conseguir, por eso existe este elemento, para simplificar la creación de este contenido.

Ejemplos de grupos

Con estos contenidos, creamos los grupos necesarios y los utilizamos.

Veamos un ejemplo complejo que utiliza los tipos complejos, simples, elementos globales y referencias, para ilustrar las formas de trabajar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:group name="gr-a">
     <xs:sequence>
        <xs:element name="Nombre" type="xs:string"/>
        <xs:element name="Edad" type="xs:decimal"/>
        <xs:element name="Sexo" type="xs:string"/>
        <xs:element name="Estado_Civil" type="xs:string"/>
     </xs:sequence>
  </xs:group>
  <xs:group name="gr-b">
     <xs:choice>
        <xs:element name="Nombre"  type="xs:string"/>
        <xs:element name="Edad" type="xs:decimal"/>
        <xs:element name="Sexo" type="xs:string"/>
        <xs:element name="Estado_Civil" type="xs:string"/>
     </xs:choice>
  </xs:group>
  <xs:group name="gr-c">
     <xs:all>
        <xs:element name="Nombre" type="xs:string"/>
        <xs:element name="Edad" type="xs:decimal"/>
        <xs:element name="Sexo" type="xs:string"/>
        <xs:element name="Estado_Civil" type="xs:string"/>
     </xs:all>
```

```
    </xs:group>
    <xs:complexType name="nuevo">
       <xs:sequence>
          <xs:element name="datosUsuario" type="direccionUsuario"/>
          <xs:element name="datosOtroUsuario" type="direccionUsuario"/>
       </xs:sequence>
       <xs:attribute name="fecha" type="xs:date"/>
    </xs:complexType>
    <xs:complexType name="direccionUsuario">
       <xs:sequence>
          <xs:element name="calle" type="xs:string"/>
          <xs:element name="numero" type="xs:string"/>
          <xs:element name="ciudad" type="xs:string"/>
          <xs:element name="provincia" type="xs:string"/>
          <xs:element name="codigoPostal" type="xs:decimal"/>
       </xs:sequence>
       <xs:attribute name="pais" type="xs:NMTOKEN" fixed="España"/>
    </xs:complexType>
    <xs:complexType name="productos">
       <xs:sequence>
          <xs:element name="producto-a" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="producto-b" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element name="producto-c" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
    </xs:complexType>
    <xs:element name="nombre">
       <xs:simpleType name="espacios">
          <xs:restriction>
             <xs:whiteSpace value="preserve"/>
          </xs:restriction>
       </xs:simpleType>
    </xs:element>
    <xs:element name="ej1">
       <xs:complexType>
          <xs:group ref="gr-a"/>
          <xs:attribute name="atrib1" type="xs:string" use="required"/>
          <xs:attribute name="atrib2" type="xs:string" use="optional"
                        default='"hola"'/>
       </xs:complexType>
    </xs:element>
    <xs:element name="ej2">
       <xs:complexType>
          <xs:group ref="gr-b"/>
       </xs:complexType>
    </xs:element>
    <xs:element name="ej3">
       <xs:complexType>
          <xs:sequence>
             <xs:group ref="gr-b"/>
             <xs:group ref="gr-c"/>
             <xs:choice>
                <xs:element ref="nombre"/>
                <xs:element name="elementoHijo" type="xs:integer"/>
             </xs:choice>
          </xs:sequence>
       </xs:complexType>
    </xs:element>
</xs:schema>
```


6.10 ELEMENTOS DE TIPO COMPLEJO

Un elemento de tipo complejo es aquel que en el documento XML puede contener elementos o atributos. Para declarar elementos de tipos complejos, se debe haber definido, o definir sobre la marcha tipos complejos como ya hemos visto.

La clasificación de todos los elementos es:

- Elementos vacíos.
- Elementos vacíos con atributos.
- Elementos con texto.
- Elementos con texto y atributos.
- Elementos con otros elementos.
- Elementos con otros elementos y atributos.
- Elementos con otros elementos, atributos y texto.

De estos, algunos son simples como los elementos vacíos y los elementos con texto, el resto son complejos y se verán a continuación.

6.10.1 Elementos vacíos

Son elementos simples que se declaran con un atributo que indica que no van a tener contenido.

Este es un ejemplo:

```
<xs:element name="elementoVacio" type="xs:string" nillable="true"/>
```

6.10.2 Elementos vacíos con atributos

Este caso se produce cuando se quiere añadir un atributo a un elemento de tipo simple y contenido vacío.

Este es un ejemplo:

```
<xs:element name="elementoVacio2" nillable="true">
   <xs:complexType>
      <xs:attribute name="atr1" type="xs:string"/>
   </xs:complexType>
</xs:element>
```

6.10.3 Elementos con texto

Estos son los elementos simples que hemos estado definiendo hasta ahora.

Un ejemplo de estos elementos es:

```
<xs:element name="elementoTexto" type="xs:string"/>
```

6.10.4 Elementos con texto y atributos

En este caso, tenemos un elemento simple pero con atributos, lo que lo convierte en un elemento de tipo complejo.

Para crear estos tipos, tenemos que indicar que el tipo es complejo pero el contenido no va a llevar otros elementos, es decir el contenido es simple. Tememos para ello los elementos "*complexType*" y "*simpleContent*". Estos elementos combinados establecen la especificación de lo que queremos, luego en el interior del contenido simple, especificamos la extensión a realizar que nos permite especificar el tipo base del contenido y nos da pie a incluir el atributo.

Veamos un ejemplo:

```
<xs:element name="elementoTexto2">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="atr1" type="xs:string"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```


6.10.5 Elementos con elementos

Estos son los elementos basados en tipos complejos como los definidos hasta ahora.

Un ejemplo es:

```
<xs:element name="ej6">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Nombre" type="xs:string"/>
         <xs:element name="Edad" type="xs:decimal"/>
         <xs:element name="Sexo" type="xs:string"/>
         <xs:element name="Estado_Civil" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:complexType name="tipoEj7">
   <xs:sequence>
      <xs:element name="Nombre" type="xs:string"/>
      <xs:element name="Edad" type="xs:decimal"/>
      <xs:element name="Sexo" type="xs:string"/>
      <xs:element name="Estado_Civil" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:element name="ej7" type="tipoEj7"/>
```

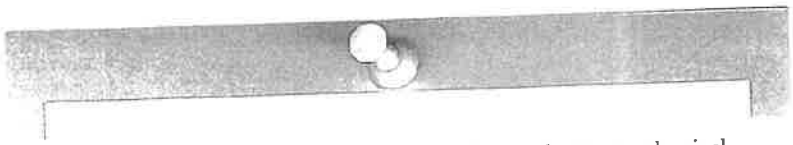
6.10.6 Elementos con elementos y atributos

Los elementos de este tipo se definen con contenido de tipo complejo como el anterior, y se incluyen luego los atributos que sean necesarios.

Por ejemplo:

```
<xs:element name="ej8">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Nombre" type="xs:string"/>
```

```
            <xs:element name="Edad" type="xs:decimal"/>
            <xs:element name="Sexo" type="xs:string"/>
            <xs:element name="Estado_Civil" type="xs:string"/>
         </xs:sequence>
         <xs:attribute name="ari" type="xs:decimal"/>
         <xs:attribute name="afe" type="xs:decimal"/>
         <xs:attribute ref="ahu"/>
      </xs:complexType>
   </xs:element>
```



Los atributos siempre deben ir en último lugar y al nivel inmediatamente inferior al elemento complexType.

6.10.7 Elementos con elementos, texto y atributos

Estos elementos son los más complejos. Equivalen a los elementos mixed definidos en las DTD.

En estos elementos se pueden incluir atributos o no, y texto entrelazado con los subelementos.

La forma de definir estos elementos es:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:element name="cuerpoDocumento">
      <xsd:complexType mixed="true">
         <xsd:sequence>
            <xsd:element name="encabezado">
               <xsd:complexType mixed="true">
                  <xsd:sequence>
                     <xsd:element name="nombre" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
            <!-- ... -->
         </xsd:sequence>
         <xs:attribute name="ari" type="xs:decimal"/>
         <xs:attribute name="afe" type="xs:decimal"/>
```



```
      </xsd:complexType>
   </xsd:element>
```

Como vemos, el atributo "mixed=true" permite aparecer caracteres entre los hijos del elemento "*cuerpoDocumento*".

6.11 OTRAS POSIBILIDADES DE LOS ESQUEMAS

Los esquemas tienen una gran riqueza expresiva, y además de las posibilidades que ya hemos analizado, tenemos otros muchas de entre las cuales hemos seleccionado un conjunto de ellas, por su importancia.

Para un conocimiento más profundo de las posibilidades más avanzadas la mejor fuente es la especificación que deberán poder entender sin problemas una vez que hayan leído este texto.

Para acceder a la especificación de los esquemas consulte el capítulo de recursos XML al final del libro.

6.11.1 Creación de un grupo de atributos

Se trata de aumentar la legibilidad de los Esquemas, y facilitar la actualización ya que un grupo de atributos se puede cambiar más sencillamente que modificar cada atributo individualmente. Se utiliza para ello el elemento "*attributeGroup*".

En el siguiente ejemplo se va a declarar el elemento "libro" con los atributos "tema", "color", "tamaño", "peso":

```
<xs:element name="elementoTexto3">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="atr1" type="xs:string"/>
            <xs:attributeGroup ref="conjunto_atributos"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
<xs:attributeGroup name="conjunto_atributos">
```

```
    <xs:attribute name="tema" type="xs:string" use="required"/>
    <xs:attribute name="color" type="xs:string" use="required"/>
    <xs:attribute name="tamanio" type="xs:string" use="required"/>
    <xs:attribute name="peso" type="xs:decimal" use="required"/>
  </xs:attributeGroup>
```

6.11.2 Conflicto en la asignación de nombres

Si se declaran dos componentes del mismo tipo y son globales, se produciría un conflicto. Los posibles conflictos son:

- Declaración de dos tipos globales de igual nombre.
- Declaración de dos elementos globales de igual nombre.
- Declaración de dos atributos globales de igual nombre.
- Declaración de dos grupos globales de igual nombre.
- Declaración de dos grupos de atributos globales de igual nombre.

En cambio no son conflictivas situaciones en las que los tipos sean distintos o no sean definiciones globales. Por ejemplo:

- Si se declara un tipo y un elemento o atributo con igual nombre no se producirá conflicto.
- Si se declara dos elementos con el mismo nombre, pero alguno de ellos es local, no se produciría conflicto.

En general el conflicto surgirá cuando aparezca una situación en la que el analizador no sepa cual de los dos escoger.

6.11.3 Tipo anyType, anySimpleType y anyURI

Un elemento declarado anyType no restringe su contenido a ningún tipo, es decir, podría contener un valor de cualquiera de los tipos existentes. De igual forma, anySimpleType aceptará valores de cualquier tipo simple y anyURI cualquier URI válida.

La sintaxis a utilizar es la siguiente:

```
<xsd:element name="cualquiercosa" type="xsd:anyType"/>
```

6.11.4 Elementos sustitutos

Una interesante facilidad de la que se disponen los esquemas es la de crear relaciones de sustitución entre elementos. Esto se hace mediante el atributo *"substitutionGroup"* del que disponen los elementos. Este atributo, permite establecer que un elemento está en el grupo de sustitución de otro.

Estableciendo esto, podemos escribir documentos en los que alternemos el uso de los elementos. Esto quiere decir que si en la definición se estableció que debía ir el elemento A, y B está en su grupo de sustitución, podemos poner B y será correcto. Además, la sustitución es transitiva, si un elemento C se declara sustituto de B, se convertirá en sustituto de A.

Con la sustitución se crea una jerarquía, para un elemento X, tenemos una serie de sustitutos. Si se crea un elementos que exige la aparición de X, se podrá usar uno de sus sustitutos. La única restricción a los grupos de sustitución es que los elementos sustitutos deben ser de un tipo compatible, más concretamente, del mismo tipo o de un tipo derivado del tipo del elemento principal.

El siguiente ejemplo ilustra estos conceptos. En primer lugar tenemos un esquema con dos grupos de sustitución, A, B y C; X, Y y Z. El esquema tiene un elemento doc que incluye a A y X un mínimo de dos veces:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sus="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sus:A" minOccurs="2" maxOccurs="20"/>
        <xs:element ref="sus:X" minOccurs="2" maxOccurs="20"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="A" type="xs:string"/>
  <xs:element name="B" type="xs:string" substitutionGroup="sus:A"/>
```

```
    <xs:element name="C" type="xs:string" substitutionGroup="sus:B"/>
    <xs:element name="X" type="xs:string" />
    <xs:element name="Y" type="xs:string" substitutionGroup="sus:X"/>
    <xs:element name="Z" type="xs:string" substitutionGroup="sus:X"/>
</xs:schema>
```

El siguiente documento es válido para el esquema anterior:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute
H:\1-Uah\3-Libros\Libro-XML-Fer\ejemplos\substitutionGroupEjem.xsd">
    <A/>
    <B/>
    <A/>
    <C/>
    <X/>
    <X/>
    <Y/>
    <Z/>
</doc>
```

Los grupos de sustitución se pueden usar para crear esquemas con elementos válidos que pertenezcan a varios idiomas. De esta forma, distintos editores de documentos XML crearán documentos con un marcado en su idioma que será equivalente al marcado en los demás idiomas.

6.11.5 Combinación de esquemas

Existe la posibilidad de combinar varios esquemas en un mismo documento. Para realiza estas acciones disponemos de dos elementos con semántica distinta, "*include*" e "*import*". Con estos elementos, conseguiremos que un esquema existente pase a formar parte del esquema que estamos creando.

La sintaxis a utilizar es:

```
<xs:include schemaLocation="www.tal.es/esquema.xsd">
</xs:include>
<xs:import namespace="www.tal2.es"  schemaLocation="www.tal2.es/esquema.xsd">
</xs:import>
```

Estos elementos deben aparecer antes que cualquier definición del esquema. La semántica en ambos casos implica la utilización de un esquema externo, pero en el efecto es como si se hubiera escrito en el propio esquema y se integrará en el espacio de nombres de nuestro esquema. En el caso de import, las definiciones pertenecen a otro espacio de nombres.

Para profundizar en la combinación de esquemas consulte la especificación del W3C que está disponible en el capítulo de recursos XML.

6.12 UN EJEMPLO DE ESQUEMA

El siguiente ejemplo corresponde con una posible aplicación sobre la gestión de una biblioteca. Este es el ejemplo final del capítulo y se intenta que con él se tenca una idea de un esquema con algo más de contenido de los usados como ejemplo a lo largo del tema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XML Spy v4.1 (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
   <xs:complexType name="AlumnoType">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="idalumno"/>
            <xs:element ref="nombre"/>
            <xs:element ref="apellido1"/>
            <xs:element ref="apellido2"/>
            <xs:element ref="dni"/>
            <xs:element ref="edad"/>
            <xs:element ref="nacionalidad"/>
            <xs:element name="domicilio_año_escolar"
                        type="domicilio_año_escolarType"/>
            <xs:element name="domicilio_familiar"
                        type="domicilio_familiarType"/>
            <xs:element ref="telefono_movil"/>
            <xs:element ref="mail"/>
            <xs:element name="foto" type="fotoType"/>
      </xs:sequence>
      <xs:attribute name="tipo_usuario" type="xs:string" use="required"/>
      <xs:attribute name="clave_alumno" type="xs:ID" use="required"/>
   </xs:complexType>
   <xs:element name="Base_de_datos">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Libro" type="LibroType"
                              minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Alumno" type="AlumnoType"
                              minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Ficha" type="FichaType"
                              minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id_Base_de_datos" type="xs:ID" use="required"/>
            <xs:attribute name="fecha_creacion" type="xs:string"
                          use="required"/>
            <xs:attribute name="tipo" use="required">
                  <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                              <xs:enumeration value="local"/>
                        </xs:restriction>
```

```
                    </xs:simpleType>
                </xs:attribute>
                </xs:complexType>
    </xs:element>
    <xs:complexType name="FichaType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="idficha"/>
                <xs:element ref="usuario"/>
                <xs:element ref="fecha_ida"/>
                <xs:element ref="fecha_vuelta"/>
                <xs:element ref="estado"/>
                <xs:element ref="uso"/>
                <xs:element ref="tipousuario"/>
                <xs:element ref="numero_libros_prestados"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="LibroType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="idlibro"/>
                <xs:element ref="titulo" maxOccurs="unbounded"/>
                <xs:element ref="autor" maxOccurs="unbounded"/>
                <xs:element ref="editorial"/>
                <xs:element ref="tema" maxOccurs="unbounded"/>
                <xs:element ref="fecha_entrega"/>
                <xs:element ref="fecha_devuelta"/>
                <xs:element ref="estado"/>
        </xs:sequence>
        <xs:attribute name="idlibro" type="xs:ID" use="required"/>
        <xs:attribute name="idioma" type="xs:string" use="required"/>
        <xs:attribute name="solicitante" type="xs:string" use="required"/>
        <xs:attribute name="motivo" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:element name="apellido1" type="xs:string"/>
    <xs:element name="apellido2" type="xs:string"/>
    <xs:element name="autor" type="xs:string"/>
    <xs:element name="cod_postal" type="xs:string"/>
    <xs:element name="cod_postalf" type="xs:string"/>
    <xs:element name="departamento" type="xs:string"/>
    <xs:element name="dni" type="xs:string"/>
    <xs:element name="domicilio" type="xs:string"/>
    <xs:complexType name="domicilio_año_escolarType">
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="domicilio"/>
                        <xs:element ref="municipio"/>
                        <xs:element ref="cod_postal"/>
                        <xs:element ref="provincia"/>
                        <xs:element ref="telefono"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="domicilio_familiarType">
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="domiciliof"/>
                        <xs:element ref="municipiof"/>
                        <xs:element ref="cod_postalf"/>
                        <xs:element ref="provinciaf"/>
                        <xs:element ref="telefonof"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="domiciliof" type="xs:string"/>
        <xs:element name="edad" type="xs:string"/>
        <xs:element name="editorial" type="xs:string"/>
        <xs:element name="estado" type="xs:string"/>
```



```
    <xs:element name="fecha_devuelta" type="xs:date"/>
    <xs:element name="fecha_entrega" type="xs:date"/>
    <xs:element name="fecha_ida" type="xs:date"/>
    <xs:element name="fecha_vuelta" type="xs:date"/>
    <xs:simpleType name="numero_dias_con_libro">
            <xs:restriction base="xs:duration">
                    <xs:minInclusive value="P0Y0M0D"/>
                    <xs:maxExclusive value="P0Y0M15D"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="fotoType">
            <xs:attribute name="filename" type="xs:string" use="required"/>
            <xs:attribute name="x" type="xs:string" use="required"/>
            <xs:attribute name="y" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:element name="idalumno" type="xs:ID"/>
    <xs:element name="idficha" type="xs:ID"/>
    <xs:element name="idlibro" type="xs:ID"/>
    <xs:element name="mail" type="xs:string"/>
    <xs:simpleType name="multas_pendientes">
            <xs:restriction base="xs:string">
                    <xs:pattern value="Si | No"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:element name="municipio" type="xs:string"/>
    <xs:element name="municipiof" type="xs:string"/>
    <xs:element name="nacionalidad" type="xs:string"/>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="numero_libros_prestados" type="xs:string"/>
    <xs:element name="provincia" type="xs:string"/>
    <xs:element name="provinciaf" type="xs:string"/>
    <xs:element name="telefono" type="xs:string"/>
    <xs:element name="telefono_movil" type="xs:string"/>
    <xs:element name="telefonof" type="xs:string"/>
    <xs:element name="tema" type="xs:string"/>
    <xs:element name="tipousuario" type="xs:string"/>
    <xs:simpleType name="titulacion">
            <xs:restriction base="xs:string">
                    <xs:enumeration value="I.T.T. Especialidad Telematica"/>
                    <xs:enumeration
                            value="I.T.T. Especialidad Sistemas Electronicos"/>
                    <xs:enumeration
                            value="I.T.T. Especialidad Sistemas de Teleco"/>
                    <xs:enumeration value="IngenieroTelecomunicacion"/>
                    <xs:enumeration value="Ingeniero Electronica"/>
                    <xs:enumeration value="I.T.Industrial"/>
                    <xs:enumeration value="I. en Geodesia y cartografia"/>
                    <xs:enumeration value="I.T. en Informatica de Sistemas"/>
                    <xs:enumeration value="I.T. en Informatica de Gestion"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:element name="titulo" type="xs:string"/>
    <xs:element name="uso" type="xs:string"/>
    <xs:element name="usuario" type="xs:string"/>
    <xs:simpleType name="dias_con_libro">
            <xs:restriction base="xs:integer">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="15"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="departamento">
            <xs:restriction base="xs:string">
```

```
                <xs:enumeration value="Ciencias_de_la_Computacion"/>
                <xs:enumeration value="Electronica"/>
                <xs:enumeration value="Automatica"/>
                <xs:enumeration value="Teoria_de_la_Señal"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="dni">
        <xs:restriction base="xs:string">
                <xs:pattern value="(\d){9}-(\S)"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="cod_postal">
        <xs:restriction base="xs:integer">
                <xs:pattern value="(\d){5}"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="cod_postalf">
        <xs:restriction base="xs:integer">
                <xs:pattern value="(\d){5}"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="telefono">
        <xs:restriction base="xs:integer">
                <xs:pattern value="(\d){2}-(\d){3}-(\d){2}-(\d){2}"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="telefonof">
        <xs:restriction base="xs:integer">
                <xs:pattern value="(\d){2}-(\d){3}-(\d){2}-(\d){2}"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="telefono_movil">
        <xs:restriction base="xs:integer">
                <xs:pattern value="(\d){3}-(\d){3}-(\d){3}"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="tipousuario">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Ajeno"/>
                <xs:enumeration value="Estudiante"/>
                <xs:enumeration value="Profesor"/>
                <xs:enumeration value="otros"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="estado">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Bueno"/>
                <xs:enumeration value="Malo"/>
                <xs:enumeration value="Perdido"/>
                <xs:enumeration value="Cedido"/>
                <xs:enumeration value="Roto"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="numero_libros_prestados">
        <xs:restriction base="xs:integer">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="3"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="numero_multas">
        <xs:restriction base="xs:integer">
                <xs:minInclusive value="0"/>
```



```
                <xs:maxInclusive value="2"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="tema">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Idiomas"/>
                <xs:enumeration value="Inormatica"/>
                <xs:enumeration value="Matematicas"/>
                <xs:enumeration value="Fisica"/>
                <xs:enumeration value="Electronica"/>
                <xs:enumeration value="Otros"/>
        </xs:restriction>
</xs:simpleType>
<xs:element name="Alquiler" type="xs:string"/>
<xs:element name="Compra" type="xs:string"/>
<xs:element name="Donativo" type="xs:string"/>
<xs:element name="Estado" type="xs:string"/>
<xs:element name="Otra_Facultad" type="xs:string"/>
<xs:element name="Otros" type="xs:string"/>
<xs:element name="origen">
        <xs:complexType>
                <xs:choice>
                        <xs:element ref="Otra_Facultad"/>
                        <xs:element ref="Estado"/>
                        <xs:element ref="Donativo"/>
                        <xs:element ref="Compra"/>
                        <xs:element ref="Alquiler"/>
                        <xs:element ref="Otros"/>
                </xs:choice>
        </xs:complexType>
</xs:element>
</xs:schema>
```

6.12.1 Un ejemplo curioso

El siguiente ejemplo, es una muestra de la flexibilidad de que disponemos en XML, de la coherencia de sus estructuras y de lo perfectamente acopladas que están. La interpretación del mismo queda para el lector que seguro no tendrá ningún problema en ello.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE doc [
<!ENTITY cadena "esta es la cadena">
]>
<doc xmlns="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cc.uah.es/jmgm/xml/libro/ejemplos/substitute
H:\1-Uah\3-Libros\Libro-XML-Fer\ejemplos\substitutionGroupEjem.xsd">
   <A/>
   <B/>
   <A/>
   <C> &cadena;</C>
```

```
    <X/>
    <X/>
    <Y/>
    <Z/>
</doc>
```

6.13 RESUMEN

En este capítulo se han tratado fundamentalmente los siguientes puntos:

- Ventajas e inconvenientes de la utilización de los Schemas frente a las DTDs para validar documentos.
- Forma de utilizar un Schema, relación con un documento XML.
- Elementos de los que consta un Schema.

Además de los puntos reseñados, se han puesto ejemplos de todos los conceptos par se pueda entender de una forma más sencilla, rápida y eficaz.

Por último se ha incluido un esquema de mayor tamaño como ejemplo.