Tema 6. XML Schema.

Tema 6. XML Schema.

1. Introducción.

Además de los DTD estudiados en el tema anterior, existen otras formas de validar documentos XML tales como los **XML Schema (XSchema)**.

Al igual que un DTD, un XSchema define los elementos que puede contener un documento XML, su organización, sus tipos y sus atributos. Sin embargo, presenta las siguientes ventajas:

- Usa **sintaxis XML**, por lo que no hay que aprender una sintaxis nueva para utilizarlo y además es analizable como cualquier documento XML.
- Permite especificar **tipos de datos**, por ejemplo, podemos indicar si un elemento o un atributo es un número entero, una fecha, una cadena de caracteres, un elemento de una lista, etc.
- Permite utilizar **diferentes espacios de nombres** (**namespace**), lo que implica que algunas de las reglas para validar ciertos elementos estarán definidas en cierto documento, mientras que otras reglas podrán estar en otro.
- Es **extensible**, es decir, el conjunto de etiquetas que lo constituye se puede ampliar según las necesidades que surjan.

2. XML Schema.

Los XML Schema están basados en el lenguaje **XML Schema Definition (XSD)** desarrollado por el World Wide Web Consortium (W3C) alcanzando el nivel de recomendación en mayo de 2001. Posteriormente, se creó una segunda edición revisada, disponible desde octubre de 2004.

Esta última recomendación (http://www.w3.org/XML/Schema), se expone en tres partes:

- **XML Schema Parte 0 *Primer*:** es una introducción no normativa al lenguaje, que proporciona una gran cantidad de ejemplos y explicaciones detalladas para una primera aproximación a XML Schema.
- **XML Schema Parte 1 *Structures*:** es una extensa descripción de los componentes del lenguaje.
- **XML Schema Parte 2 *Datatypes*:** complementa la parte 1 con la definición de los tipos de datos incorporados en XML Schema y sus restricciones.

Los documentos XML que se validan mediante un esquema se denominan **instancias del esquema.**

2.1. Referencia a un XML Schema.

Para indicar en un documento XML cuál es el archivo **XML Schema** con el que se pretende validar, se añaden ciertos atributos en elemento raíz del documento XML. Existen dos posibilidades:

Si en el documento XML a validar no se declara ningún espacio de nombres, se usa el atributo **noNameSpaceSchemaLocation** asignándole el nombre del archivo XML Schema (normalmente con extensión xsd)**.** Esta situación no es aconsejable ya que siempre es conveniente definir en los documentos XML algún espacio de nombres, aunque sea un espacio de nombres por defecto. No obstante, se usa bastante para hacer pruebas; de hecho lo usaremos en los ejemplos de los apuntes por simplicidad.

Como el atributo noNameSpaceSchemaLocation pertenece al vocabulario definido en el espacio de nombres XML Schema Instance, (**http://www.w3.org/2001/XMLSchema-instance**), hay que definir dicho espacio de nombres para poder usarlo. Por ejemplo:

```
<elemento_raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNameSpaceSchemaLocation="esquema.xsd">
```

Si en el documento XML a validarse se declaran espacio de nombres, se usa en su lugar el atributo **schemaLocation.** Este atributo esta formato por una par de cadenas: la primera indica el espacio de nombres que será validado por el esquema (el espacio de nombres correspondiente al propio documento XML); y la segunda la ruta al documento XMLSchema. Por ejemplo:

```
<elemento_raiz xmlns="http://www.ejemplo.org"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.ejemplo.org esquema.xsd">
```

Si el documento XML utiliza varios espacios de nombres, habría que indicar varios esquemas de validación, cada uno de ellos con las reglas a cumplir por las etiquetas de cada espacio de nombres. Para tal fin el atributo **schemaLocation** permite contener una lista de espacios de nombres y localizaciones de esquemas, separados éstos por espacios en blanco.

A continuación se muestra un ejemplo en el que se permite validar elementos y atributos situados en dos espacios de nombres distintos mediante dos esquemas diferentes:

```
<raiz xmlns:doc="http://www.ejemplo.org/doc"
      xmlns:img="http://www.ejemplo.org/img"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ejemplo.org/doc esquemaDoc.xsd
                          http://www.ejemplo.org/img esquemaImg.xsd">
```

3. Estructura de un XML Schema.

Un esquema es un documento XML que se guarda en un archivo con extensión **xsd** y, como tal, presenta la estructura habitual de cualquier documento de tipo XML con la particularidad de que el elemento raíz se llama **schema**. Por tanto, la estructura general de un XML Schema será:

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

Donde **xs** es el prefijo del espacio de nombres que identifica al espacio de nombres de XML Schema: http://www.w3.org/2001/XMLSchema

Los componentes básicos de un esquema son:

- **xs:schema.** El elemento raíz de todo esquema.
- Definición de **elementos** mediante etiquetas **xs:element**. Indican los elementos permitidos en los documentos que sigan el esquema.
- Definición de **atributos**, mediante etiquetas **xs:attribute**. Indican los atributos de los elementos.

El ámbito de aplicación de los elementos y atributos dentro de un esquema, queda determinado por el lugar en el que se encuentran sus definiciones. De aquí que se distingan dos posibilidades a la hora de declarar componentes:

- De **ámbito global**. Se trata de elementos o tipos del esquema situados dentro de la etiqueta raíz **schema** pero fuera de cualquier otra. Estos componentes se pueden utilizar o reusarse en cualquier parte del esquema.
- De **ámbito local**. Se trata de elementos o tipos definidos dentro de otros. En ese caso, se pueden utilizar sólo dentro del componente en el que están incluidos pero no en otras partes del documento. Por ejemplo, si dentro de la definición de un elemento colocamos la definición de un tipo de datos concreto, este tipo de datos sólo se puede utilizar dentro del componente **xs:element** en el que se encuentra dicha definición.

El orden de los componentes que se encuentran dentro de un mismo ámbito de un esquema no es significativo, es decir, las declaraciones se pueden hacer en cualquier orden.

3.1. El elemento <xs:schema>.

Este componente es el que realiza la declaración del esquema, siendo el elemento raíz de todo el esquema. Puede tener algunos atributos. De hecho, generalmente aparece de una forma similar a ésta:

```
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.empresa.com"
            xmlns="http://www.empresa.com"
            elementFormDefault="qualified">
            ... ...
</xs:schema>
```

- xmlns:xs="http://www.w3.org/2001/XMLSchema" declara el espacio de nombres que utilizan los esquemas. En este espacio de nombres se define el vocabulario, (etiquetas, atributos, etc.) que se puede usar en un XML Schema). Esta es la manera de diferenciar las etiquetas XML del esquema, respecto a otras que no lo son. Se suele usar el prefijo **xs**, aunque a veces también **xsd.**

- targetNamespace="http://www.empresa.com". Es una referencia obligatoria al espacio de nombres declarado en el documento XML para indicar que los elementos que se definen en el esquema (es decir, los elementos, atributos, etc. del documento XML) pertenecen a dicho espacio de nombres. Si en el documento XML se usan varios espacios de nombres, se tendrán que referenciar todos ellos separándolos mediante espacios en blanco.
- xmlns="http://www.empresa.com" Un XML Schema es un documento XML, y por lo tanto, como en cualquier documento XML podemos definir espacios de nombres para informar que el esquema se aplica a un espacio de nombres privado correspondiente a la entidad a la que se quiere aplicar el esquema. Como ya sabemos, este espacio de nombres se puede declarar como espacio de nombres por defecto (es lo habitual) o usar un prefijo; incluso se pueden indicar varios espacios de nombres.
- elementFormDefault= "qualified | **unqualified**". Indica si los elementos usados en el documento XML instancia, deben estar cualificados (qualified) o no (unqualified). Es decir, si deben pertenecer a un espacio de nombres (aunque sea un espacio de nombres por defecto), o no. El valor por defecto es unqualifed, es decir, que los elementos del documento XML no pertenecen a ningún espacio de nombres.

Ojo. **xmlns** y **targetNamespace** no sirven para lo mismo; el primero declara espacios de nombre y el segundo sirve para indicar a qué espacio de nombres pertenecen los elementos definidos en el esquema. Aunque en la práctica ciertamente el contenido de ambos atributos es el mismo.

3.2. El elemento <xs:element>.

Permite la declaración de elementos y representa la aparición de un elemento en el documento XML. Los atributos principales (hay más…) son:

- **id=**"identificador único para el elemento".
- **name=**"nombre del elemento". Obligatorio si el elemento padre es <xs:schema>.
- **type=**"tipo de datos del elemento".
- **ref=**"referencia a otro elemento el cual realiza la descripción". No se usa si el elemento padre es <xs:schema>
- **form=**"qualified|unqualified" Fuerza a que el elemento sea o no clasificado dentro del espacio de nombres mediante un prefijo. El valor por defecto es el definido para el atributo elementFormDefault de <xs:schema>.
- **minOccurs**="número mínimo de ocurrencias que puede haber del elemento". Valores: 0, 1, 2 , … unbounded
- **maxOccurs**="número máximo de ocurrencias que puede haber del elemento". Valores: 0, 1, 2 , … unbounded
- **default=**"valor por defecto para el elemento cuando en el documento XML no haya recibido ningún valor". Sólo para elementos con contenido simple o texto.
- **fixed=**"indica el único valor que puede tomar el elemento". Sólo para elementos con contenido simple o texto.

Supongamos el siguiente documento XML, junto al documento XML Schema articulo.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<articulo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:noNamespaceSchemaLocation="articulo.xsd">
    <nombre>tornillo</nombre>
    <unidades>50</unidades>
    <fecha>2012-02-14</fecha>
</articulo>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="articulo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="descripcion" type="xs:string" />
                <xs:element name="unidades" type="xs:nonNegativeInteger" />
                <xs:element name="fecha" type="xs:date" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

El elemento articulo queda definido mediante su modelo de contenido (descripcion, unidades, precio); y éstos últimos mediante el atributo type.

Otra forma posible para definir un elemento es usando el atributo ref="elemento", y luego definir dicho elemento referenciado. Por ejemplo:

```
<xs:element ref="descripcion" />
```

y luego definir el elemento referenciado

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

Los elementos simples pueden tener un valor por defecto o un valor prefijado. Un valor por defecto se asigna automáticamente al elemento cuando no especificamos ninguno. En el ejemplo se asigna *tuerca*

```
<xs:element name=" descripcion " type="xs:string" default="tuerca"/>
```

Un elemento con valor prefijado siempre tiene el mismo valor y no podemos asignarle otro distinto:

```
<xs:element name="fecha" type="xs:date" fixed="2000-01-01"/>
```

3.3. El elemento <xs:attribute>.

Permite la declaración de atributos y representa la aparición de un atributo de un elemento en el documento XML. Los atributos principales son:

- **id=**"identificador único para el atributo".
- **name=**"nombre del atributo". Este atributo no puede aparecer simultáneamente con **ref**.
- **type=**"tipo de dato del atributo".
- **ref=**"permite hacer referencia a otro lugar en donde se realiza la descripción".
- **form=**"qualified|unqualified" Fuerza a que el atributo sea o no clasificado dentro del espacio de nombres mediante un prefijo. El valor por defecto es el definido para el atributo elementFormDefault de <xs:schema>.
- **use=**"indica si el atributo es opcional, obligatorio o prohibido (**optional**, required, prohibited)".
- **default=**"valor por defecto para el atributo cuando en el documento XML no haya recibido ningún valor".
- **fixed=**"indica el único valor que puede tomar el atributo". No puede aparecer simultáneamente con **default**.

Ejemplo: para la línea XML siguiente:

```
<descripcion codigo="A10">clavo</descripcion >
```

La declaración del atributo codigo podría ser:

```
<xs:attribute name="codigo" type="xs:string" use="required"/>
```

Ejemplo: atributo de nombre divisa, con tipo de dato texto y con valor por defecto Euros.

```
<xs:attribute name="divisa" type="xs:string" default="Euros"/>
```

4. Tipos de datos.

En los ejemplos anteriores se han declarado elementos y atributos de ciertos tipos de datos. Los datos se pueden clasificar de diferentes formas. Una de ellas es la siguiente:

- Datos **predefinidos.** Son los que vienen integrados en la propia especificación del esquema y pertenecen al espacio de nombres http://www.w3.org/2001/XMLSchema. Como por ejemplo xs:integer, xs:string, xs:double, etc.
- Datos **construidos.** Son definidos por el usuario basándose en algún tipo predefinido o en otro construido previamente. Mediante estos tipos, el usuario puede construir sus **propios tipos** a partir de los predefinidos en XSD a través de uniones de tipos de datos, restricciones a los tipos de datos y listas de datos. Para ello se usará como base el elemento **xs:simpleType**.

Por otra parte, los tipos de datos se organizan de forma jerárquica, al estilo de la jerarquía de clases en la POO, de manera que cada tipo será igual que su tipo padre más alguna característica propia.

Existen dos tipos especiales que no se suelen usar por que están en la parte superior de la jerarquía y por lo tanto son demasiado genéricos:

- xs:anyType. Es el tipo raíz del que derivan todos los demás tipos existentes. Para los elementos es el tipo por defecto si no se especificara ninguno. Como en realidad es un tipo complejo, veremos posteriormente que no podrá usarse para los atributos.
- xs:anySimpleType. Representa en general a cualquier tipo simple (no complejo). Es el tipo por defecto para los atributos.

4.1. Tipos de datos predefinidos.

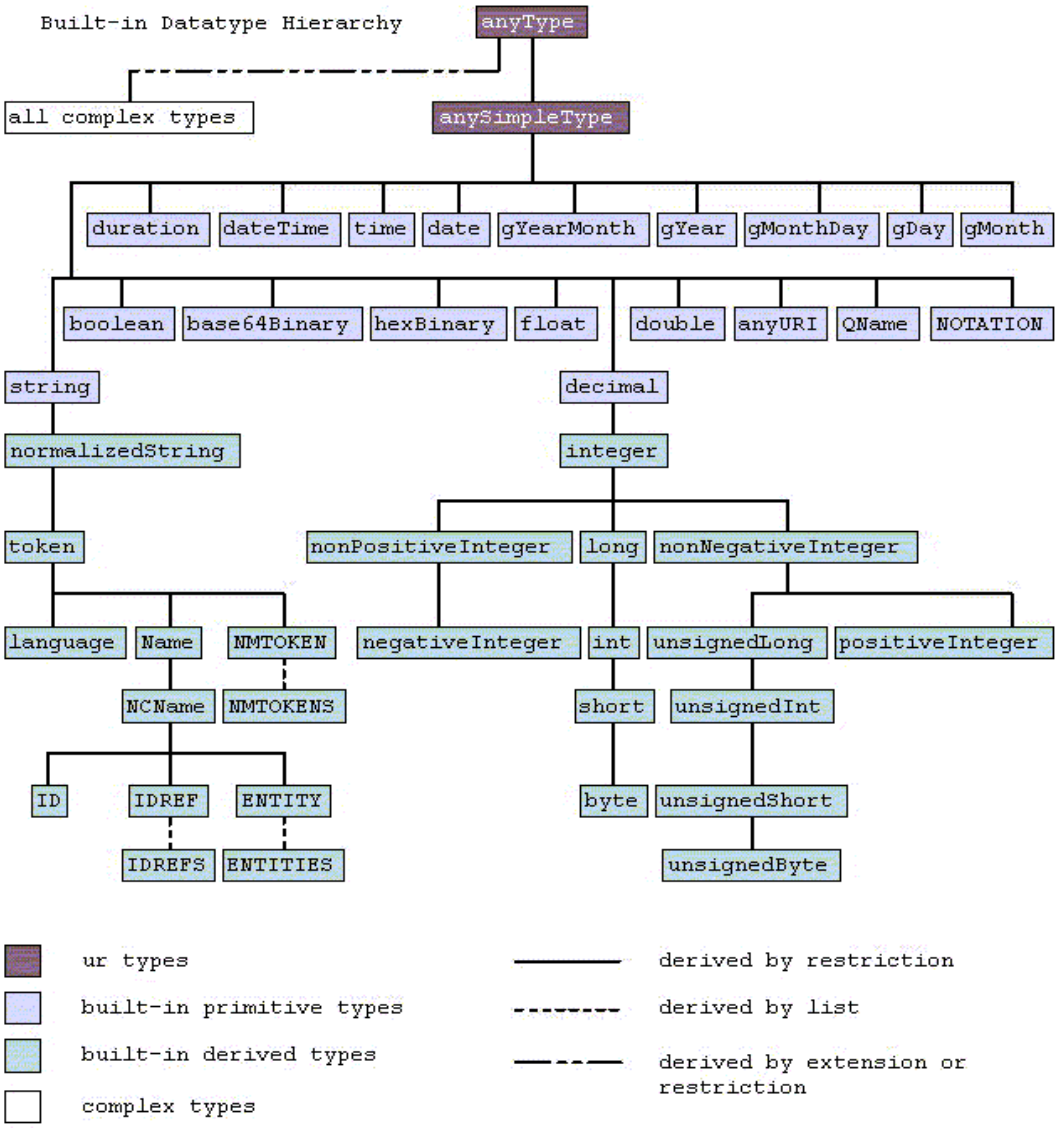
Los tipos predefinidos son tipos simples (derivan de xs:anySimpleType), y se clasifican en:

- Primitivos. Son los que derivan directamente de xs:anySimpleType.
- Derivados (o no primitivos). Son los que derivan de algunos de los primitivos anteriores.

Se agrupan según su tipo de contenido en *numéricos*, *fecha*, *texto*, *binarios* y *booleanos*. La figura siguiente (http://www.w3.org/TR/xmlschema-2/) muestra la jerarquía de datos predefinidos (*primitivos*, *derivados* y *especiales*).

Los tipos de datos predefinidos pueden ser asignados a elementos y a atributos, y pertenecen al espacio de nombres http://www.w3.org/2001/XMLSchema.

Algunos de los tipos predefinidos que se usan más habitualmente son (ver figura):



- **xs:string**, para textos con cualquier contenido, excepto <, >, & y comillas (simples y dobles), para los que se ha de utilizar la entidad correspondiente.
- **xs:boolean**, sólo puede contener los valores verdadero y falso, escritos como *true* y *false* o *1* (verdadero) y *0* (falso).
- **xs:decimal**, para números reales en coma fija.
- **xs:integer**, para números enteros.
- **xs:float**, para números reales en coma flotante y simple precisión. Se usa el punto decimal como separador decimal.
- **xs:double**, números reales en coma flotante y doble precisión. Se usa el punto decimal.

- **xs:date**, para fechas en formato *aaaa-mm-dd*.
- **xs:time**, para horas en formato *hh:mm:ss*.

Algunos tipos de **datos derivados de los primitivos** definidos por XSD son:

- **xs:normalizedString**, texto sin retornos de carro, saltos de línea o tabuladores.
- **xs:token**, texto sin retornos de carro, saltos de línea, tabuladores, espacios al principio o al final, o dos o más espacios seguidos a lo largo de él.
- **xs:language**, texto que contiene el nombre de un idioma (en, en-EN, en-US, es, es-ES, etc.)
- **xs:negativeInteger**, enteros estrictamente menores que cero. (z < 0)
- **xs:positiveInteger**, enteros estrictamente mayores que cero. (z > 0)
- **xs:nonNegativeInteger**, enteros mayores o iguales que cero. (z >= 0)
- **xs:nonPositiveInteger**, enteros menores o iguales que cero. (z <= 0)
- Los tipos siguientes se definen para mantener la compatibilidad con los tipos de atributos que aparecen en los DTD.
  - **xs:ID, xs:IDREF, xs:IDREFS**.
  - **xs:ENTITY, xs:ENTITIES**.
  - **xs:NMTOKEN, xs:NMTOKENS**.
  - **xs:NOTATION**.

En http://www.w3.org/TR/xmlschema-0 se encuentra todos los tipos de datos definidos en XSD.

5. Tipos de datos simples y complejos.

Los datos también se pueden clasificar en simples y complejos.

Todos los tipos de datos predefinidos vistos anteriormente son simples y en general representan valores atómicos. Se pueden asignar tanto a elementos como a atributos. También se pueden construir datos simples partiendo de un tipo predefinido a través de restricciones, uniones y listas.

Los tipos de datos complejos sólo se pueden asignar a elementos que tengan descendientes y/o atributos. Como luego veremos los tipos complejos pueden ser de cuatro clases: estar vacíos (con o sin atributos), contener sólo texto pero con atributos, contener sólo a otros elementos (con o sin atributos), y contener mezclado tanto texto como otros elementos (con o sin atributos).

Cuando se definen tipos por construcción, éstos pueden ser *anónimos* o con *nombre*. Con los primeros se gana en rapidez, pero los tipos con nombre permiten la reusabilidad. Ejemplo:

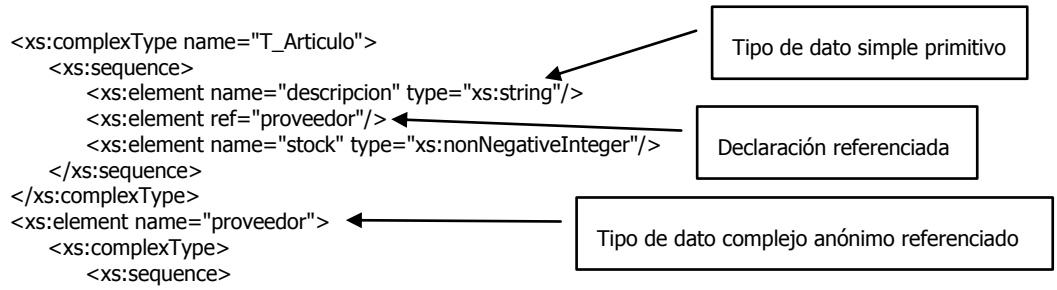
```
<?xml version="1.0" encoding="UTF-8"?>
<almacen  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
            xsi:noNamespaceSchemaLocation="almacen.xsd">
    <articulo>
        <descripcion>ali-kates</descripcion>
        <proveedor>
            <codigo-proveedor>PR-342</codigo-proveedor>
            <nombre-proveedor>Alfonso</nombre-proveedor>
        </proveedor>
        <stock>345</stock>
    </articulo>
    <articulo>
        <descripcion>martillo pilon</descripcion>
        <proveedor>
            <codigo-proveedor>PR-999</codigo-proveedor>
            <nombre-proveedor>Mauricio</nombre-proveedor>
        </proveedor>
        <stock>12</stock>
    </articulo>
</almacen>
```

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

    <xs:element name="almacen">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="articulo" type="T_Articulo" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Tipo de dato complejo *anónimo*

Tipo de dato complejo con *nombre*




```
    <xs:complexType name="T_Articulo">
        <xs:sequence>
            <xs:element name="descripcion" type="xs:string"/>
            <xs:element ref="proveedor"/>
            <xs:element name="stock" type="xs:nonNegativeInteger"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="proveedor">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="codigo-proveedor" type="xs:string"/>
                <xs:element name="nombre-proveedor" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

5.1. Definición de tipos simples por restricción.

La definición de tipos simples se realiza con el componente <xs:simpleType>, y una de las formas de definirlos es mediante una restricción sobre un tipo base. Con el elemento <xs:restriction> podemos indicar dicha restricción limitando el rango de valores sobre un tipo de datos base.

La sintaxis es la siguiente:

```
<xs:simpleType name="nombreDelTipo">
    <xs:restriction base="tipoBase">
        Definición de la(s) restricción(es)
    </xs:restriction>
</xs:simpleType>
```

El atributo base indica el tipo de datos sobre el que se establece las posibles restricciones.

Ejemplo:

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

A partir de este momento el tipo de dato T_Longitud se podrá usar en todo el esquema, y como en realidad no se ha definido con ninguna restricción, será equivalente al tipo xs:float

```
<xs:element name="distancia" type="T_Longitud"/>
```

Al elemento distancia se le ha asignado el tipo T_Longitud

Ejemplo:

```
<xs:simpleType name="T_Clave">
    <xs:restriction base="xs:string">
        <xs:minLength value="6"/>
        <xs:maxLength value="10"/>
    </xs:restriction>
</xs:simpleType>

<xs:element name="password" type="T_Clave"/>
```

Se ha definido el tipo T_Clave restringiendo el rango de valores permitidos al tipo base xs:string.

Cuando definimos un tipo de datos con nombre, éste podrá ser referenciado repetidas veces usando dicho nombre. Otra solución, pero menos ventajosa, es definirlo de forma anónima, como por ejemplo:

```
<xs:element name="password" >
    <xs:simpleType>
      <xs:restriction base="xs :string">
        <xs:minLength value="6"/>
        <xs:maxLength value="10"/>
      </xs:restriction>
    </xs:simpleType>
</xs:element >
```

En este caso el elemento password queda perfectamente descrito, pero no hemos definido un tipo con nombre que posteriormente pudiéramos usarlo para definir otros elementos o atributos.

A continuación, vemos los diferentes tipos de restricciones (facetas) que se pueden imponer. En función del tipo base se podrán aplicar una u otras facetas.

FACETA	**USO**	**DATOS DONDE SE USA**
xs:minInclusive	Indica el límite inferior del rango de valores, incluido el propio valor	Numéricos y fecha/hora
xs:maxInclusive	Indica el límite superior del rango de valores, incluido el propio valor	Numéricos y fecha/hora
xs:minExclusive	Indica el límite inferior del rango de valores, excluido el propio valor	Numéricos y fecha/hora
xs:maxExclusive	Indica el límite superior del rango de valores, excluido el propio valor	Numéricos y fecha/hora
xs:enumeration	Indica una lista de valores permitidos	Todos
xs:pattern	Indica una expresión regular que deben cumplir los valores	Texto
xs:whiteSpace	Indica cómo tratar los espacios en blanco. Los posibles valores son *preserve, replace* y *collapse*	Texto
xs:length	Indica el número de caracteres o elementos de una lista permitidos	Texto
xs:minLength	Indica el número mínimo de caracteres o elementos de una lista permitidos	Texto
xs:maxLength	Indica el número máximo de caracteres o elementos de una lista permitidos	Texto
xs:fractionDigits	Indica el número máximo de cifras de la parte decimal permitidas	Numéricos con parte decimal
xs:totalDigits	Indica el número máximo de dígitos totales permitidos	Numéricos

5.1.1. Restricciones de longitud de un texto.

- **xs:length**, indica un número determinado de caracteres para el texto.
- **xs:maxLength**, indica el número máximo de caracteres para el texto.
- **xs:minLength**, indica el número mínimo de caracteres para el texto.

En el siguiente ejemplo se define el tipo T_nombre como una cadena de caracteres con un número de caracteres comprendido entre 2 y 20.

```
<xs:simpleType name="T_nombre">
    <xs:restriction base="xs:string">
        <xs:minLength value="2"/>
        <xs:maxLength value="20"/>
    </xs:restriction>
</xs:simpleType>
```

5.1.2. Cifras de un número.

- **xs:totalDigits**, indica el número máximo de cifras totales de un número (incluidas las decimales).
- **xs:fractionDigits**, indica el número máximo de cifras decimales de un número.

En el ejemplo siguiente se define el tipo T_nota como un número de, como máximo, cuatro cifras de las que, como mucho, dos son decimales.

```
<xs:simpleType name="T_nota">
    <xs:restriction base="xs:decimal">
        <xs:totalDigits value="4"/>
        <xs:fractionDigits value="2"/>
    </xs:restriction>
</xs:simpleType>
```

5.1.3. Valores máximo y mínimo de un rango.

- **xs:minExclusive**, abarca el rango de valores estrictamente mayores que el indicado.
- **xs:maxExclusive**, abarca el rango de valores estrictamente menores que el indicado.
- **xs:minInclusive**, abarca el rango de valores mayores o iguales al indicado.
- **xs:maxInclusive**, abarca el rango de valores menores o iguales al indicado.

Para que el elemento edad sea un entero comprendido entre 18 y 65, escribiremos:

```
<xs:element name="edad">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="18"/>
            <xs:maxInclusive value="65"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

5.1.4. Restricciones sobre los espacios en blanco.

Se lleva a cabo mediante **xs:whiteSpace** con tres posibles valores para el atributo value:

- **preserve**. El documento XML se procesará para que se conserven espacios en blanco, tabuladores, saltos de línea y retornos de carro.
- **replace**. El documento XML se procesará para que cada doble espacio, tabulador, salto de línea o retorno de carro se sustituya por un único espacio en blanco.
- **collapse**. Es similar al anterior pero, además, elimina espacios a izquierda y derecha.

5.1.5. Enumeraciones.

Permiten indicar, uno a uno, todos los posibles valores que forman parte de un tipo.

En este ejemplo se define el tipo T_EstadoCivil con todos sus posibles valores.

```
<xs:simpleType name="T_EstadoCivil">
    <xs:restriction base="xs:string">
        <xs:enumeration value="soltero"/>
        <xs:enumeration value="casado"/>
        <xs:enumeration value="separado"/>
        <xs:enumeration value="divorciado"/>
        <xs:enumeration value="viudo"/>
    </xs:restriction>
</xs:simpleType>
```

5.1.6. Plantillas (patrones).

Permiten definir la secuencia exacta de caracteres que se admiten en un tipo de dato. Para ello, se utiliza la etiqueta **<xs:pattern value="expresion"/>**, donde **expresion** es una expresión regular con estas normas:

Patrón	Significado	Ejemplo
.	Cualquier carácter (excepto retorno de carro o nueva línea)	;
\w	Cualquier letra o dígito	K
\W	No cualquier letra o dígito	%
\d	Un dígito decimal	5
\D	No dígito decimal	@
\s	Cualquier carácter de espaciado (tabulador, espacio, etc.)	\t
\S	No carácter de espaciado	H
(a|b)	Alternativa entre 2 expresiones	b
texto	Sólo se admite dicho "*texto*".	texto
[xyz]	Permite elegir uno entre los caracteres *x, y, z*.	y

[^xyz]	Prohíbe usar los caracteres *x, y, z*.	a
[a-z]	Permite cualquier carácter comprendido entre *a* y *z*.	h
[A-E-[C]]	Substracción de un carácter de un rango	B
+	Sucesión de 1 o más ocurrencias.	
?	Sucesión de 0 ó 1 ocurrencias.	
*	Sucesión de 0 o más ocurrencias.	
{n}	Acepta que el carácter precedente aparezca exactamente *n* veces.	
{n,}	Acepta que el carácter precedente aparezca al menos *n* veces.	
{n,m}	Acepta que el carácter precedente aparezca entre *n* y *m* veces.	
^	Inicio de línea.	
$	Final de línea.	
\carácter	Permite escribir caracteres reservados, como ^, $, +, *, -, (,), ?,	

A continuación, se define el tipo de dato T_DNI:

```
<xs:simpleType name="T_DNI">
      <xs:restriction base="xs:string">
          <xs:pattern value="[0-9]{8}[A-Z]"/>
      </xs:restriction>
</xs:simpleType>
```

También podemos definir el tipo T_sexo con dos posibles valores, *hombre* o *mujer*, de la forma:

```
<xs:simpleType name="T_sexo">
      <xs:restriction base="xs:string">
          <xs:pattern value="hombre | mujer"/>
      </xs:restriction>
</xs:simpleType>
```

Una definición de tipo para números de teléfonos en formato de 9 dígitos separados por un punto cada tres dígitos, como por ejemplo 958.225.431 sería:

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

5.2. Definición de tipos simples por unión de contenidos

Se puede definir un nuevo tipo de dato a partir de una colección de datos simples previamente definidos. Para ellos se usa el componente **xs:union** que indica que un valor es válido si al menos es válido para al menos uno de los tipos de datos que forman la unión.

Por ejemplo, podemos crear el tipo de dato T_enteroNoCero de la siguiente forma:

```
<xs:simpleType name=" T_enteroNoCero">
    <xs:union memberTypes="xs:NegativeInteger xs:PositiveInteger"/>
</xs:simpleType>
```

5.3. Definición de tipos simples mediante listas.

El componente **xs:list** permite definir tipos cuyos valores no son atómicos o indivisibles, sino múltiples formados por una secuencia finita o lista de valores de un tipo de datos base. Cada valor de la lista debe aparecer separado por un espacio en blanco.

El atributo itemType permite indicar el tipo de los datos de la lista. Este tipo puede ser predefinido o bien un tipo simple definido. El ejemplo siguiente define un elemento llamado <serie> cuyo tipo es T_listaEnteros el cual define una lista de enteros.

```
<xs:element name="serie" type=" T_listaEnteros ">
```

```
<xs:simpleType name="T_listaEnteros">
    <xs:list itemType="xs:integer"/>
</xs:simpleType>

<serie>-5 -4 -3 -2 -1 0 1 2 3 4 5</serie>
```

En el siguiente ejemplo, notas se define como un tipo anónimo para una lista de reales pero con una restricción respecto a los valores mínimo y máximo de los valores de la lista.

```
<xs:element name="notas">
    <xs:simpleType>
        <xs:list>
            <xs:simpleType>
                <xs:restriction base="xs:float">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="10"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:list>
    </xs:simpleType>
</xs:element>

<notas>6.5 8.3 6.8 7.2 4.5</notas>
```

También se podría haber definido un tipo, por ejemplo T_Reales_0_10 para definir números reales entre 0 y 10; y posteriormente usarlo para definir la lista de la forma <xs:list itemType="T_Reales_0_10">.

6. Definición de tipos de datos complejos.

Estos tipos de datos usan el componente **complexType** y sólo pueden ser asignados a elementos. Los elementos complejos son aquellos que contienen a otros **elementos descendientes y/o atributos**.

En general, se definen con esta sintaxis:

```
<xs:complexType name="nombre" mixed="true | false">
    Definición del elemento …
</xs:complexType>
```

El atributo name indica el nombre del tipo que será obligatorio si el componente es hijo de xs:schema, de lo contrario no está permitido. El atributo mixed indica si se intercala o no texto con los elementos descendientes. El valor por defecto es false.

El contenido de un elemento XML, lo que hay entre las etiquetas de apertura y cierre puede ser:

- **Contenido simple** (xs:simpleContent). Solo contiene texto, sin elementos descendientes.
- **Contenido complejo** (xs:complexContent). Contiene elementos descendientes y puede o no contener texto.

Cuando hablamos de elementos descendientes, aparece el concepto de **modelo de contenido.** El modelo de contenido indica la relación o distribución que tienen los elementos descendientes. Hay tres posibilidades las cuales se declaran con un componente distinto.

❖ **Secuencia**

Los elementos aparecen unos detrás de otros en un orden fijo. Se declaran con el componente **xs:sequence**. Dicha secuencia puede aparecer un número configurable de veces estableciéndose con los atributos minOccurs y maxOccurs cuyos valores por defecto son uno.

Ejemplo. El nombre de una persona <nombre> con los elementos descendientes <npila>, <ape1> y <ape2>.

```
<xs:complexType name="T_Nombre">
    <xs:sequence>
        <xs:element name="npila" type="xs:string"/>
        <xs:element name="ape1" type="xs:string"/>
        <xs:element name="ape2" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

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

❖ **Alternativa**

Se establece con el componente **xs:choice** y permite que se pueda escoger uno sólo entre varios elementos. La alternativa puede aparecer un número configurable de veces estableciéndose con los atributos minOccurs y maxOccurs cuyos valores por defecto son uno.

En el siguiente ejemplo una empresa puede actuar como cliente o como proveedor.

```
<xs:element name="empresa">
    <xs:complexType>
        <xs:choice>
            <xs:element name="cliente" type="xs:string"/>
            <xs:element name="proveedor" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

❖ **Todos**

Indica que los elementos hijos aparecen en cualquier orden, pero cada uno de ellos puede aparecer como máximo una sola vez. Se usa el componente **xs:all**.

En el ejemplo el nombre y código deben aparecer los dos y en el orden que se quiera.

```
<xs:element name="identificador">
    <xs:complexType>
        <xs:all>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="codigo" type="xs:positiveInteger"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

Para indicar que estos elementos puedan no aparecer, es necesario utilizar el atributo minOccurs="0" en la etiqueta <xs:all> de la siguiente forma:

```
<xs:all minOccurs="0">
```

En general, para los elementos descendientes de otros, cuando se desea establecer el número de veces que aparecerán, se hace uso de los indicadores minOccurs y maxOccurs. Por defecto, el valor de ambos es 1 pero podemos cambiarlo a cualquier número entero mayor o igual que cero. Además, para expresar un valor ilimitado se usa la palabra unbounded.

```
<xs:element name="persona">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="nombrehijo" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

6.1. Distintas posibilidades de elementos complejos.

Las diferentes posibilidades de contenido para los elementos complejos son:

- Estar vacíos, pudiendo tener o no atributos. (*Empty*).
- Contener solo texto y atributos (no tiene elementos descendientes). (*Simple Content*).
- Contener otros elementos descendientes, pudiendo tener o no atributos (*Complex Content*).
- Contener tanto texto como otros elementos descendientes, pudiendo tener o no atributos (*Mixed Content*).

6.2. Elementos vacíos.

Se declara como un tipo de datos complejo (**xs:complexType**) donde no aparece ningún modelo de contenido (no contiene elementos descendientes). Sólo pueden aparecer declaraciones de atributos.

La siguiente línea de código XML puede validarse con el código XSD que aparece a continuación:

```
<articulo codigo="14" fabricante="Acme"/>
```

```
<xs:element name="articulo">
    <xs:complexType>
        <xs:attribute name="codigo" type="xs:positiveInteger"/>
        <xs:attribute name="fabricante" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

O bien, de esta otra forma:

```
<xs:element name="articulo"  type="T_Articulo"/>

<xs:complexType name="T_Articulo">
    <xs:attribute name="codigo" type="xs:positiveInteger"/>
    <xs:attribute name="fabricante" type="xs:string"/>
</xs:complexType>
```

Si no se desearan atributos para el elemento (caso poco frecuente pues sería un elemento sin contenido ni atributos), entonces simplemente no existiría el componente <xs:attribute>, el resto se mantendría:

```
<xs:complexType name="T_Articulo" />
```

6.3. Elementos que sólo contienen texto y atributos.

Se declara como un tipo de datos complejo (**xs:complexType**) donde para indicar que sólo contiene texto y no a otros elementos descendientes, se usa el componente **xs:simpleContent**.

Para indicar que el elemento va a contener atributos se usa el componente **xs:extension**. Su misión es extender el tipo de datos de un elemento simple (indicado mediante su atributo base), con declaraciones de atributos.

Por ejemplo:

```
<xs:complexType name="T_Articulo">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="codigo" type="xs:positiveInteger"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<articulo codigo="14">Tornillo</articulo>
```

6.4. Elementos sólo con elementos descendientes y atributos.

Estos contenidos se refieren a los elementos que contienen otros elementos (pero nunca texto libre) y que puedan contener atributos. Se declara con un tipo de datos complejo (**xs:complexType**) con contenido complejo que contenga algún elemento descendiente.

Recordemos que los elementos descendientes podían aparecer según uno de los tres modelos de contenido: secuencias, alternativa y todos. Si además de elementos descendientes se definen atributos, éstos se indican siempre al final, justo antes de la etiqueta de cierre </xs:complexType>.

- Ejemplo:

```
<xs:complexType name="T_Factura">
    <xs:sequence>
        <xs:element name="numero" type="xs:integer" />
        <xs:element name="cliente" type="xs:string" />
        <xs:element name="importe" type="xs:float" />
    </xs:sequence>
    <xs:attribute name="fecha" type="xs:date" />
</xs:complexType>

<xs:element name="facturacion" type="T_Factura" />
```

El siguiente fragmento XML cumpliría con la declaración del esquema anterior

```
<facturacion fecha="2012-03-1">
    <numero>4523</numero>
    <cliente>Obdulio Flores</cliente>
    <importe>67.89</importe>
</facturacion>
```

- Ejemplo:

```
<xs:element name="empresa">
    <xs:complexType>
        <xs:choice>
            <xs:element name="cliente" type="xs:string"/>
            <xs:element name="proveedor" type="xs:string"/>
        </xs:choice>
        <xs:attribute name="codigo" type="xs:integer" use="required" />
    </xs:complexType>
</xs:element>
```

El siguiente fragmento XML cumpliría con la declaración del esquema anterior

```
<empresa codigo="45">
    <proveedor>ACME</proveedor>
</empresa>
```

6.5. Elementos con contenido textual y elementos descendientes.

Se declara con un tipo de datos complejo (**xs:complexType**) con contenido complejo que contenga algún elemento descendiente y contenido textual. Puede tener o no declaraciones de atributos.

Para indicar que un elemento de tipo complejo es mixto, es decir, que puede contener texto y otros elementos descendientes se usa el atributo mixed en el componente xs:complexType de la forma:

```
<xs:complexType mixed="true">
```

Este código XML podría validarse con el siguiente código XSD:

```
<carta>
    Estimado señor o señora <nombre>Luis de la Casa</nombre>:
    Su pedido <numpedido>1503</numpedido>
    le será enviado el <fechaenvio>2012-05-10</fechaenvio>
</carta>
```

```
<xs:element name="carta">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element name="nombre" type="xs:string" />
            <xs:element name="numpedido" type="xs:positiveInteger" />
            <xs:element name="fechaenvio" type="xs:date" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

En el caso de que el elemento también tuviera atributos, se indicarían al final, justo antes de la etiqueta de cierre </xs:complexType>.

```
<carta referencia="145" autor="MariPili">
    Estimado señor o señora <nombre>Luis de la Casa</nombre>:
    Su pedido <numpedido>1503</numpedido>
    le será enviado el <fechaenvio>2012-05-10</fechaenvio>
</carta>
```

```
<xs:element name="carta">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element name="nombre" type="xs:string" />
            <xs:element name="numpedido" type="xs:positiveInteger" />
            <xs:element name="fechaenvio" type="xs:date" />
        </xs:sequence>
        <xs:attribute name="referencia" type="xs:integer" use="required" />
        <xs:attribute name="autor" type="xs:string" />
    </xs:complexType>
</xs:element>
```

6.6. Extensiones de tipo complejo.

Se pueden añadir nuevos elementos y atributos a tipos complejos ya existentes. Estos elementos añadidos aparecerán al final de los elementos del tipo base.

- Ejemplo. Supongamos que disponemos de un tipo complejo T_Nombre definido de la forma:

```
<xs:complexType name="T_Nombre">
    <xs:sequence>
        <xs:element name="npila" type="xs:string"/>
        <xs:element name="ape1" type="xs:string"/>
        <xs:element name="ape2" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Ahora queremos declarar un tipo complejo denominado T_Alumno que almacene además del nombre y sus dos apellidos, tres elementos para almacenar domicilio, localidad y provincia; y un atributo numérico entero para el número de matrícula del alumno.

Por lo tanto se trata de definir un nuevo tipo complejo, cuyo contenido es de tipo complejo (xs:complexContent) al que se le añade una extensión (xs:extension) a cuyo atributo base se le asigna el tipo de datos complejo que sirve de base (T_Nombre)

```
<xs:complexType name="T_Alumno">
    <xs:complexContent>
        <xs:extension base="T_Nombre">
        <xs:sequence>
            <xs:element name="domicilio" type="xs:string"/>
            <xs:element name="localidad" type="xs:string"/>
            <xs:element name="provincia" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="matricula" type="xs:integer" use="required" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

La declaración de un elemento <alumno> de tipo T_Alumno sería:

```
<xs:element name="alumno" type="T_Alumno" />
```

Así por ejemplo la siguiente descripción de elementos sería válida según el tipo anterior

```
<alumno matricula="7012">
    <npila>Juan</npila>
    <ape1>Ruiz</ape1>
    <ape2>Morales</ape2>
    <domicilio>Pablo Picasso, 5</domicilio>
    <localidad>La Palma del Condado</localidad>
    <provincia>Huelva</provincia>
</alumno>
```

7. Grupos.

A menudo ocurre que un mismo grupo de elementos aparece repetidas veces a lo largo del código de un esquema. Con el fin de simplificar su escritura, podemos declararlo una sola vez, y referenciarlo posteriormente desde otros elementos complejos tantas veces como sea necesario.

Los grupos no son tipos de datos, sino contenedores que incluyen un conjunto de elementos o atributos que pueden ser usados en la definición de tipos complejos. Para ello se utilizan los componentes xs:group y xs:attributeGroup.

Cuando un elemento desea incorporar un grupo, utiliza el componente xs:group, y con el atributo ref se indicaría el nombre del grupo definido anteriormente. He aquí un ejemplo de uso:

```
<xs:group name="infoPedido">
    <xs:sequence>
        <xs:element name="cliente" type="xs:string" />
        <xs:element name="detallepedido" type="xs:string" />
        <xs:element name="facturado_a" type="xs:string" />
        <xs:element name="enviado_a" type="xs:string" />
    </xs:sequence>
</xs:group>

<xs:element name="pedido">
    <xs:complexType>
        <xs:group ref="infoPedido" />
        <xs:attribute name="estado" type="xs:string" />
    </xs:complexType>
</xs:element>
```

Al igual que los elementos, los atributos también se pueden agrupar para ser referenciados en bloque desde uno o varios elementos complejos.

En el siguiente ejemplo se define el grupo de atributos codigoIncidencia, que es usado a continuación en un elemento complejo:

```
<xs:attributeGroup name="codigoIncidencia">
    <xs:attribute name="anno" type="xs:positiveInteger" />
    <xs:attribute name="numero" type="xs:integer" />
</xs:attributeGroup>

<xs:complexType name="T_incidencia">
    <xs:sequence>
        <xs:element name="informa" type="xs:string" />
        <xs:element name="centro" type="xs:string" />
        <xs:element name="fecha" type="xs:date" />
        <xs:element name="estado" type="xs:string" />
    </xs:sequence>
    <xs:attributeGroup ref="codigoIncidencia"/>
</xs:complexType>
```

8. Documentación.

A un esquema se le pueden añadir componentes de tipo comentario que permitan posteriormente con alguna aplicación a medida generar documentación que es obtenida del propio esquema.

Estos comentarios se indican con el componente **xs:annotation** y sus subelementos **xs:appinfo** y/o **xs:documentation**.

El componente xs:annotation puede incluirse dentro del elemento xs:schema o dentro del resto de los componentes de un esquema, pero siempre aparecerá en primer lugar.

El componente xs:appinfo se usa para indicar información que vaya a ser usada por la aplicación que genere la documentación.

El componente xs:documentation se usa para asociar comentarios de texto usados por las personas.

El componente xs:documentation puede usar el atributo xml:lang para especificar el idioma en que se expresa la documentación. Ambos componentes pueden usar el atributo source para indicar una URI donde localizar información adicional sobre la aplicación.

Las anotaciones no tienen efecto sobre la validación del esquema. A continuación se muestra un ejemplo de documentación de un elemento:

```
<xs:element name="producto" type="T_Producto">
    <xs:annotation>
        <xs:documentation xml:lang= "es-ES">
            Este producto es de fabricación propia.
        </xs:documentation>
        <xs:appinfo>
            <app:dbmapping>
                <app:tb>PRODUCT_MASTER</app:tb>
            </app:dbmapping>
        </xs:appinfo>
    </xs:annotation>
    Resto de declaración del elemento. . .
</xs:element>
```

9. Modelos de diseño de esquemas.

A la hora de diseñar esquemas podemos elegir varios métodos de descripción. Todos son válidos, aunque dependiendo de la complejidad de los esquemas y de su tamaño, algunos de ellos podrán ser más eficientes o cómodos para el usuario. Para construir diferentes esquemas usando diferentes modelos, partiremos del siguiente documento

```
<?xml version="1.0" encoding="UTF-8"?>
<almacen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <articulo cod="23">
        <descripcion>ali-kates</descripcion>
        <proveedor>
            <codigo-proveedor>PR-342</codigo-proveedor>
            <nombre-proveedor>Alfonso</nombre-proveedor>
        </proveedor>
        <stock fecha="2012-12-21">345</stock>
    </articulo>
    <articulo cod="78"> . . . </articulo>
        . . .
    <articulo cod="32"> . . . </articulo>
</almacen>
```

9.1. Modelo de diseño plano.

Se declaran los elementos y los atributos, y posteriormente se indican referencias a sus definiciones que se realizan en otro lugar del documento. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="almacen">
      <xs:complexType>
          <xs:sequence>
              <xs:element ref="articulo" maxOccurs="unbounded" />
          </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="articulo">
      <xs:complexType>
          <xs:sequence>
              <xs:element ref="descripcion"/>
              <xs:element ref="proveedor"/>
              <xs:element ref="stock"/>
          </xs:sequence>
      <xs:attribute ref="cod" use="required" />
      </xs:complexType>
</xs:element>
<xs:element name="descripcion" type="xs:string" />
<xs:element name="proveedor">
      <xs:complexType>
          <xs:sequence>
              <xs:element ref="codigo-proveedor"/>
              <xs:element ref="nombre-proveedor"/>
          </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="codigo-proveedor">
      <xs:simpleType>
          <xs:restriction base="xs:string">
              <xs:pattern value="PR-\d{3}"/>
          </xs:restriction>
      </xs:simpleType>
</xs:element>
<xs:element name="nombre-proveedor" type="xs:string"/>
```

```
<xs:element name="stock">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:integer">
                <xs:attribute name="fecha" type="xs:date" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:attribute name="cod" type="xs:integer" />
</xs:schema>
```

9.2. Modelo con tipos de nombres reutilizables.

Se definen los tipos de datos simples o complejos dándoles un nombre. Cuando se declaren los elementos y los atributos se indicarán que son de algunos de los tipos declarados. Ejemplo:

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

    <xs:element name="almacen" type="T_almacen"/>

    <xs:complexType name="T_almacen">
        <xs:sequence>
            <xs:element name="articulo" type="T_articulo"  maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="T_articulo">
        <xs:sequence>
            <xs:element name="descripcion" type="T_descripcion"/>
            <xs:element name="proveedor" type="T_proveedor"/>
            <xs:element name="stock" type="T_stock"/>
        </xs:sequence>
        <xs:attribute name="cod" type="T_cod" use="required" />
    </xs:complexType>

    <xs:simpleType name="T_descripcion">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>

    <xs:complexType name="T_proveedor">
        <xs:sequence>
            <xs:element name="codigo-proveedor" type="T_codigo_proveedor"/>
            <xs:element name="nombre-proveedor" type="T_nombre_proveedor"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="T_codigo_proveedor">
        <xs:restriction base="xs:string">
            <xs:pattern value="PR-\d{3}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="T_nombre_proveedor">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>

    <xs:complexType name="T_stock">
        <xs:simpleContent>
            <xs:extension base="xs:integer">
                <xs:attribute name="fecha" type="xs:date" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:simpleType name="T_cod">
        <xs:restriction base="xs:integer"/>
    </xs:simpleType>

</xs:schema>
```

9.3. Modelo de diseño anidado.

Cada elemento y atributo se describe en el mismo lugar donde se declara. Esto puede producir que elementos del mismo tipo se describan varias veces, y que elementos con el mismo nombre tengan distintas descripciones.

Se consiguen esquemas más compactos pero más difíciles de entender y seguir. Ejemplo:

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

  <xs:element name="almacen">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="articulo" maxOccurs="unbounded">
                  <xs:complexType>
                      <xs:sequence>

                          <xs:element name="descripcion" type="xs:string"/>
                          <xs:element name="proveedor">
                              <xs:complexType>
                                  <xs:sequence>
                                      <xs:element name="codigo-proveedor">
                                          <xs:simpleType>
                                              <xs:restriction base="xs:string">
                                                  <xs:enumeration value="PR-342"/>
                                              </xs:restriction>
                                          </xs:simpleType>
                                      </xs:element>
                                      <xs:element name="nombre-proveedor" type="xs:string"/>
                                  </xs:sequence>
                              </xs:complexType>
                          </xs:element>

                          <xs:element name="stock">
                              <xs:complexType>
                                  <xs:simpleContent>
                                      <xs:extension base="xs:integer">
                                          <xs:attribute name="fecha" use="required" type="xs:date"/>
                                      </xs:extension>
                                  </xs:simpleContent>
                              </xs:complexType>
                          </xs:element>

                      </xs:sequence>
                      <xs:attribute name="cod" use="required" type="xs:integer"/>
                  </xs:complexType>
              </xs:element>
          </xs:sequence>
      </xs:complexType>
  </xs:element>

</xs:schema>
```