SCHEMA XML

XML (eXtensible Markup Language) es un lenguaje de marcas utilizado fundamentalmente para almacenamiento de información. XML es un estándar no una implementación concreta. XML es un metalenguaje de marcas, lo que significa que no dispone de un conjunto finito de etiquetas (como ocurre con HTML) que todo el mundo debe conocer. XML permite definir los elementos que se necesiten y con la estructura que se precise.

También veremos que un documento XML una vez creado un lenguaje de marcas para uso específico, se puede validar indicando que elementos pueden aparecer (vocabulario), el orden de aparición, que estructura y atributos tiene un elementos, que elementos son optativos y cuales obligatorios, etc. Existen dos técnicas relevantes para validar un documento: los DTD y los esquemas XML La primera está en desuso en la actualidad aunque fué dominante. También existen otras técnicas de validación de documentos XML como: Relax NG y Schematron. De esta forma se han creado lenguajes basados en XML para usos específicos, como SVG, WML, RSS, FO, ...

XML tiene herramientas para el tratamiento y la recuperación de datos dentro de estas podemos destacar:

Bases de datos XML nativas, que son una alternativa a las BBDD relacionales. Si estas son adecuadas para almacenar datos, las BBDD nativas XML son más adecuadas para almacenar documentos, como BaseX.

Para acceder a los documentos XML existen lenguajes de consulta. Podemos destacar: XPath y XQuery. XPath es un lenguaje de expresiones que permite acceder a partes de un documento XML. Xquery hace uso de XPath y permite la manipulación de documentos XML.

Se pueden conectar documentos XML con Xlink y XPointer.

Existen cada vez más sistemas gestores de bases de datos relacionales que soportan el almacenamiento XML y que, con sus herramientas, permiten la consulta y modificación de los mismos. Dentro de éstas podemos destacar: Oracle y Microsoft SQL Server.

Casi todos los lenguajes de alto nivel modernos (Java, C##, ...) permiten acceder a documentos XML.

Para transformar documentos XML se utiliza XSL que es una familia de lenguajes, basados en XML Dentro de estos podemos destacar XSLT y XSL-FO. XSLT nos permite transformar documentos XML en otros documentos XML, HTML o texto. XSL-FO es un lenguaje de maquetado que, combinado con las transformaciones, permite generar documentos de salida en formatos pdf, PostScript, RTF, etc. a partir de datos contenidos en documentos XML.

PARA QUE SIRVE XML:

Es muy utilizado para intercambio de información. Ya que xml es texto plano es adecuado para almacenar información y transmitirla.

- Es muy utilizado para estructurar documentos. Podríamos decir que, lo mismo que las bases de datos relacionales son adecuadas para almacenar y tratar datos, XML es adecuado para almacenar y tratar documentos.
- XML no hace nada por si mismo Estructura la información y permite que otros programas la utilicen.

Limitaciones de los DTD:

Algunas limitaciones:

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

Validación de documentos XML con esquemas

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

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

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

Los documentos XML que se validan contra un esquema se llaman instancia del esquema, se puede ver como que el esquema es el molde y los documentos XML los objetos que tratan de ajustarse a ese molde, comparando con Java, los esquemas son las clases y los documentos XML son los objetos.

Herramientas para validar esquemas:

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

Herramienta válida de validación online:

http://www.corefiling.com/opensource/schemaValidate.html

Estructura de un documento XML.- Componentes

Reglas:

- Su elemento raíz se llama <schema>
- su espacio de nombres debe ser http://www.w3.org/2001/XMLSchema, se podrá no definir un prefijo o utilizar uno de los más comunes xs o xsd.
  Esquema sencillo:

```
<xml version =”1.0”>
<xs:schema xmlns:xs=”http://www3.org/2001/XMLSchema”>
…
<xs:schema>
```

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

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

Un posible esquema es:

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

- Puede haber esquemas con múltiples elementos raíz, por ejemplo con los elementos raíz <simple> y <complejo>, cualquier documento xml cuyo elemento raíz sea bien simple o bien complejo, se considerará conforme con este esquema:

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

- Un esquema es un conjunto de declaraciones de elementos y atributos y definición de tipos que sirve para fijar la estructura que deben tener sus documentos instancia XML. El orden en que se declara los elementos, llamados componentes en un esquema no es significativo ni afecta al funcionamiento del mismo.
- La vinculación de un esquema al documento XML se realiza en el documento XML insertando **xmlns:xsi y xsi:noNamespacesSchemaLocation como atributos del elemento raíz.** Si el esquema estuviese asociado a un espacio de nombres se usaría el atributo **xsi:schemaLocation**

– **Ejemplo:**
A continuación en el documento XML se declara un espacio de nombres de prefijo xsi, propio de las instancias de esquemas XML y se indica la ubicación del esquema, en este caso simple.xsd

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

Aclaración: Vinculación Schema con documento XML

En los documentos instancia XML, la referencia a un esquema XML se hace mediante atributos en el elemento raíz del documento instancia XML. Estos atributos, especificados en el estándar, se encuentran definidos en el espacio de nombres "http://www.w3.org/2001/XMLSchema-instance". Por lo tanto para poder usarlos hay que hacer referencia a dicho espacio de nombres en el documento instancia XML, mediante el atributo "xmlns". Normalmente a este espacio de nombres se le asigna el prefijo "xsi" (aunque se puede usar cualquier prefijo), con lo que la referencia al espacio de nombres quedaría así:
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Con esto podemos utilizar los atributos **"noNameSpaceLoction" y "schemaLocation"**, que nos permiten asociar un documento instancia XML con un esquema:
noNamespaceSchemaLocation: identifica un documento de schema que no tiene un espacio de nombres de destino (no incluye el atributo "targetNamespace") y lo asocia al documento XML instancia.
schemaLocation: Asocia un documento de esquema que tiene un espacio de nombres de destino (targetNamespace) con un documento de instancia. Este atributo tendrá dos valores, separados por un espacio en blanco. El primer valor coincide con el del "targetNamespace" especificado en el schema. El segundo es la ubicación donde se encuentra definido el XML schema.
Por ejemplo, supongamos que tenemos el archivo "apuntes.xsd" que contiene un XML schema. Si en dicho schema se definió
targetNamespace:"http://www.prueba.es/esquema1"
En un documento instancia que queremos asociar a este esquema tendremos:

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation:"http://www.prueba.es/esquema1 apuntes.xsd"

Si por el contrario en el documento "apuntes.xsd" no se especifica Un "targetNamespace", en el documento instancia XML tendremos:

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="apuntes.xsd"

Componentes básicos de un esquema.-

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

Todos los elementos que se vayan a usar en el ejemplar XML tienen que declararse en el esquema. Las declaraciones de elementos en XML Schema tienen esta sintaxis:

```
<xs:element name="nombreElemento" type="tipoSimple/tipoComplejo"
            minOccurs="valor"
            maxOccurs="valor"
            fixed=”valor"
            default=”valor”/>
```

name: es el nombre del elemento
type: el tipo de elemento.

Modelos de contenido para elementos.-

El contenido de un elemento se define mediante un modelo de contenido. En XML Schema existen cuatro modelos de contenido para elementos:

1.- Texto.- El elemento solo puede contener datos carácter. Usamos el string (cadena de caracteres)

```
<xs:element name=”autor” type=”xs:string”/>
```

2.- Vacío:el elemento no tiene contenido pero si puede contener atributos. En este caso hay que declararlos como tipo complejos, si no contienen atributos pueden declararse como tipo simples.

Ejemplo: declarar el elemento antigüedad, de contenido vacío pero es de tipo complejo porque contiene atributo anosdeservicio” que es de tipo “positiveInteger”

```
<xs:element name=”antiguedad”
<xs:ComplexType>
<xs:attribute name=”anosdeservicio” type=”xs:positiveInteger”/>
</xs:complexType>
</xs:element>
```

3.- Elementos: el elemento puede contener dentro sub-elementos. Existen tres formas de especificar los sub-elementos dentro del elemento, mediante tres tipos de elementos predefinidos en XML Schema:”sequence”,”choice” y “all”

El elemento <xs:sequence> Se utiliza este elemento para indicar una secuencia de elementos que tienen que aparecer en el documento XML. Deben aparecer todos y en el mismo orden en que se especifican. Ejemplo:

```
<xs:element name="camiseta">
<xs:complexType>
<xs:sequence>
        <xs:element name="color" type="xsd:string"/>
        <xs:element name="talla" type="xsd:string"/>
<xs:sequence>
</xs:complexType>
</xs:element>
```

El elemento <xs:choice> especifica una lista de elementos de los cuales sólo puede aparecer uno en el documento XML.

Ejemplo:

```
<xs:element name="vehiculoMotor">
<xsd:complexType>
<xs:choice>
        <xs:element name="coche" type="xs:string"/>
        <xs:element name="moto" type="xs:string"/>
        <xs:element name="furgoneta" type="xs:string"/>
        <xs:element name="camion" type="xs:string"/>
<xs:choice>
</xs:complexType>
</xs:element>
```

El elemento “choice” puede incluir opcionalmente los atributos minOccurs y maxOccurs, para especificar el mínimo y máximo número de elementos hijos que pueden incluirse en el documento.

El elemento <xs:all> Se comporta igual que el elemento <xs:sequence> pero no es obligado que en el documento XML aparezcan todos los elementos especificados, ni en el mismo orden.

```
<xsd:element name="camiseta">
```

```
<xs:complexType>
<xs:all>
        <xs:element name="color" type="xs:string"/>
        <xs:element name="talla" type="xs:string"/>
</xs:all>
</xs:complexType>
</xs:element>
```

4.- Mixto (Mixed). El elemento puede contener tanto datos carácter como elementos hijos. Los elementos hijos se definen igual que en el modelo anterior, mediante los elementos “sequence” “choice” o “all”, para indicar que además el elemento pueda incluir datos carácter se usa el atributo “mixed” con valor igual a “true” en el elemento “complexType”.
Ejemplo:

```
<xs:element name="confirmacionPedido">
<xs:complexType mixed=”true”>
        <xs:sequence>
                <xs:element name=”intro” type=”xs:string”/>
                <xs:element name=”nombre” type=”xs:string”/>
                <xs:element name=”fecha” type=”xs:string”/>
                <xs:element name=”titulo” type=”xs:string”/>
        <xs:sequence>
</xs:complexType>
</xs:element>
```

El elemento definido arriba podría usarse dentro de un documento XML de la siguiente manera:

```
<confimacionPedido>
        <intro>Para:</intro>
        <nombre>Antonio Lara</nombre>
        Confirmamos que con fecha <fecha>24-01-2005</fecha> hemos recibido su pedido
        de <titulo>Raices</titulo>. Su título será enviado en 2 días hábiles desde la
        recepción del pedido. Gracias,Ediciones Aranda.
</confimacionPedido>
```

Referencia a otros elementos.-

En un schema es posible declarar elementos de forma global y luego hacer referencias a ellos desde otros elementos. La principal ventaja de esto es que permite reutilizar una misma definición de un elemento en varios sitios del schema. Otra ventaja de este mecanismo es que puede ayudar a que los schema estén mejor estructurados y sean más fácilmente legibles. Para referenciar a un elemento se utiliza el atributo "ref" cuyo valor es el nombre del elemento referenciado, en lugar del atributo "name" seguido de la definición del elemento.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Libro">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Título"/>
                        <xsd:element ref="Autores"/>
                        <xsd:element ref="Editorial"/>
                </xsd:sequence>
                <xsd:attribute name="precio" type="xsd:double"/>
        </xsd:complexType>
</xsd:element>
<xsd:element name="Título" type="xsd:string"/>
<xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
<xsd:element name="Editorial" type="xsd:string"/>
</xsd:schema>
```

Como se declaran los atributos

La sintaxis genérica de declaración de atributos es la siguiente:

```
<xs:attribute name="nombreAtributo"
type="tipoSimple"
use="valor"
default="valor"
fixed="valor"/>
```

name: es el nombre del atributo.
type: el tipo del atributo. Los atributos sólo pueden contener tipos simples.
use (Opcional): puede tomar uno de los siguientes valores:
required: el atributo debe aparecer en el documento XML.
optional: el atributo puede aparecer o no aparecer en el documento XML. Este es el valor por defecto.
prohibited: el atributo no debe aparecer en el documento XML.
default (Opcional): si el atributo no aparece en el documento XML, se le asigna el valor especificado en el atributo "default". Los valores por defecto sólo tienen sentido si el atributo es opcional, de lo contrario tendremos un error.
fixed (Opcional): define un valor fijo para el atributo.

Si el valor del atributo está presente en la instancia del documento XML, el valor debe ser el mismo que el que indica el atributo "fixed"

Si el atributo no está presente en el documento XML, se le asigna el valor contenido en el atributo "fixed"

Los valores de los atributos "default" y "fixed" son mutuamente exclusivos, por lo tanto habrá un error si una declaración contiene ambos.

Recordemos que sólo los elementos de tipo compuesto pueden contener atributos. Las declaraciones de atributos para un elemento deben aparecer siempre al final del bloque

delimitado por la etiqueta de inicio <complexType> y la de fin </complexType>, después de las especificaciones de todos los demás componentes.
Veamos un ejemplo de un elemento que contiene un atributo:

```
<xs:element name="cliente">
<xs:complexType>
        <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellido" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="numCliente" type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Tipos de datos simples predefinidos en XML Schema xs

Tipos simples: son elementos que sólo pueden contener datos carácter; no pueden incluir otros elementos ni tampoco atributos. Ejemplo:

```
<xs:element name=”fecha” type=”xs:date”/>
```

Declaramos un elemento llamado “fecha”, de tipo “date” (el prefijo xs: indica que este tipo de datos “date” es parte del vocabulario de XML Schema).

Tipos complejos: estos elementos pueden incluir otros elementos y/o atributos. El contenido de estos elementos se define entre la etiqueta de inicio

<xs:complexType> y la correspondiente de cierre </xs:complexType>.

Ejemplo:

```
<xs:element name=”libro”>
<xs:complexType>
        <xs:attribute name=”formato” type=”xs:string” use=”required”/>
</xs:complexType>
</xs:element>
```

Existen 19 tipos de datos simples predefinidos primitivos, que se pueden agrupar en 4 categorías:

Tipos cadena

- **string**: secuencia de longitud finita de caracteres*
- **anyURI**: una uri estándar de Internet
- **NOTATION**: declara enlaces a contenido externo no-XML
- **Qname**: una cadena legal Qname (nombre con cualificador)

Tipos binario codificado

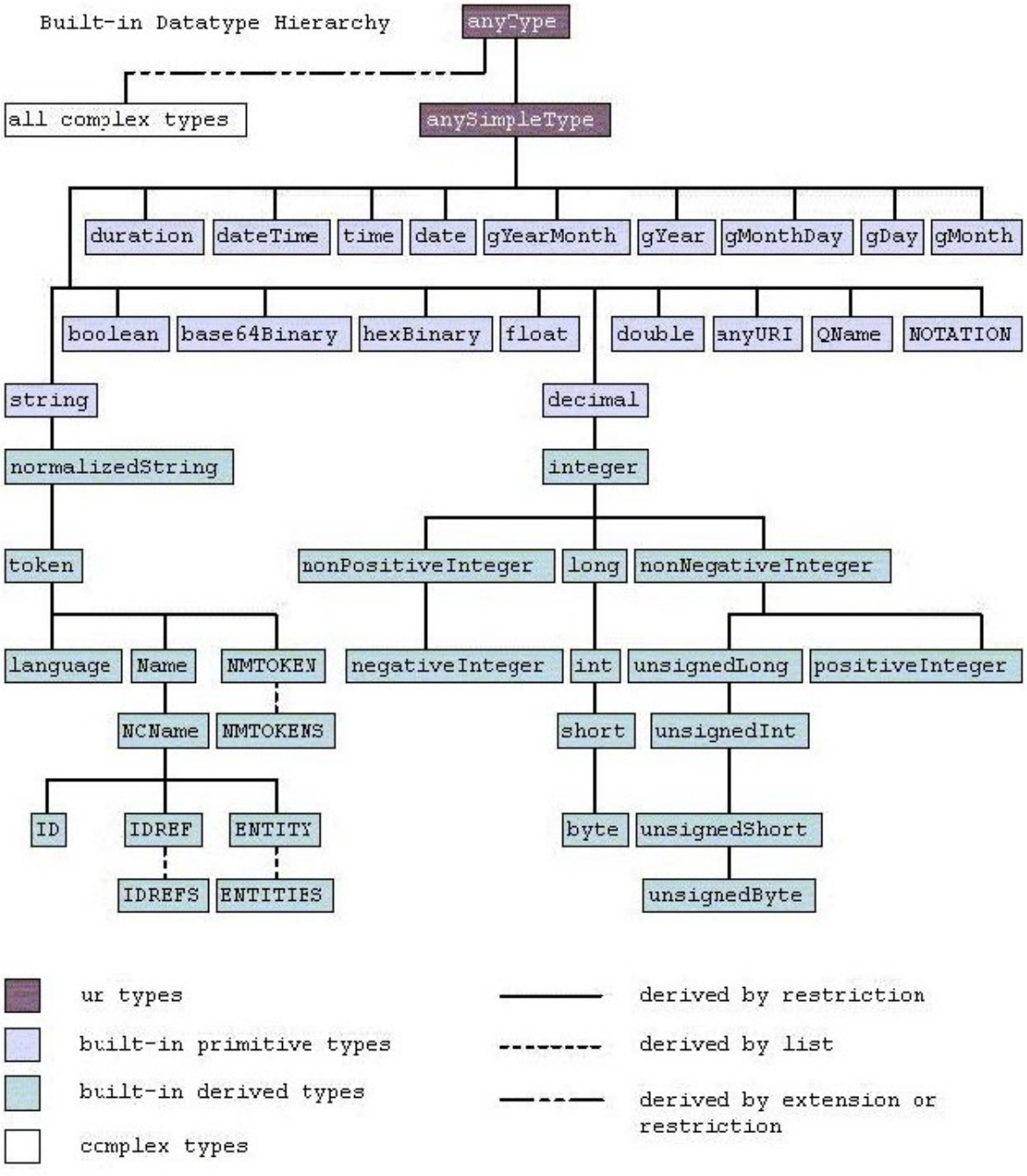
- **boolean**: toma los valores “true” o “false” *
- **hexBinary**: dato binario codificado como una serie de pares de dígitos hexadecimales
- **base64Binary**: datos binarios codificados en base 64

Tipos numéricos

- **decimal:** número decimal de precisión (dígitos significativos) arbitraria *
- **float:** número de punto flotante de 32 bits de precisión simple *
- **double**: número de punto flotante de 64 bits de doble precisión *

Tipos de fecha/hora

- **duration:** duración de tiempo
- **dateTime:** instante de tiempo específico, usando calendario gregoriano, en formato "YYYYMM-DDThh:mm:ss"
- **date**: fecha específica del calendario gregoriano, en formato "YYYY-MM-DD” *
- **time**: una instancia de tiempo que ocurre cada día, en formato "hh:mm:ss"
- **gYearMonth**: un año y mes del calendario gregoriano
- **gYear:** año del calendario gregoriano
- **gMonthDay**: día y mes del calendario gregoriano
- **gMonth**: un mes del calendario gregoriano
- **gDay**: una fecha del calendario gregoriano (día)


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
ccmplex types
derived by restriction
derived by list
derived by extension or restriction

Crear un tipo complejo con un nombre:

La forma más sencilla de crear un nuevo tipo es crear un elemento complexType al que se le asigna un **nombre mediante el atributo "name".**

Ejemplo:

```
<xsd:complexType name="precioInfo">
        <xsd:sequence>
                <xsd:element ref="precioTipo"/>
                <xsd:element ref="precioN"/>
        </xsd:sequence>
</xsd:complexType>
<xsd:element name="precioTipo" type="xsd:string"/>
<xsd:element name="precioN" type="xsd:decimal"/>
```

Con esto creamos un tipo nuevo llamado "precioInfo" que está formado por una secuencia de dos elementos, "precioTipo" y "precioN", a los que referencia. Podremos utilizar este tipo que hemos creado para definir elementos, por ejemplo:

```
<xsd:element name="precio" type="precioInfo"/>
```

Observamos que en la definición del elemento precio, el nombre del tipo "precioInfo" no lleva el prefijo "xsd", porque no pertenece al espacio de nombres del estándar XML Schema. La principal ventaja de definir tipos de datos propios es, por un lado, que estos tipos se pueden utilizar donde se quiera, y además, que estos tipos se pueden utilizar como tipos base para definir otros tipos. A continuación vamos a ver los mecanismos que existen para definir tipos derivados de otros.

Crear un tipo por restricción de un tipo simple predefinido (xs:restriction)

La forma más sencilla de crear un nuevo tipo a partir de uno ya existente es añadir condiciones a alguno de los tipos predefinidos en el XML Schema. Esto se hace con el elemento **<xsd:restriction>**. Por ejemplo:

```
<xs:simpleType name="monedaUSA">
        <xs:restriction base="xsd:decimal">
        <xs:fractionDigits value="2"/>
</xs:restriction>
</xs:simpleType>
```

En este ejemplo creamos un nuevo tipo simple y le asignamos el nombre "monedaUSA". Mediante el uso del elemento "xs:restriction" definimos el tipo "monedaUSA" como un subtipo del tipo base "xs:decimal", en el que el número de cifras decimales es 2 (xs:fractionDigits value="2"). **Las restricciones (también llamadas facetas)** que pueden aplicarse a los tipos simples son:

- minInclusive: mínimo valor que puede tomar el número; por ejemplo, si minInclusive vale 5, el número tiene que ser mayor o igual que 5.
- minExclusive: el número debe ser mayor que este valor; por ejemplo, si minExclusive vale 5, el número debe ser mayor que 5.
- maxInclusive: máximo valor que puede tomar el número; por ejemplo, si maxinclusive vale 5, el número tiene que ser menor o igual que 5.
- maxExclusive: el número debe ser menor que este valor; por ejemplo, si maxExclusive vale 5,el número debe ser menor que 5.
- totalDigits: total de cifras en el número, incluyendo las enteras y las decimales.
- fractionDigits: número de cifras decimales.
- length: número de unidades del valor literal; para la mayoría de los tipos (por ejemplo los tipos "string" y sus derivados, la faceta "length" se refiere a caracteres, pero para listas (que veremos más adelante) se refiere al número de elementos en la lista, y para valores binarios se refiere al número de octetos. No se puede aplicar a los tipos "integer", "float" o "double" (para estos se puede utilizar "totalDigits").

- minLength y maxLength: valor mínimo y máximo respectivamente para la faceta “length”.
- pattern: formato que debe tener el valor, especificado mediante una expresión regular tradicional.
- enumeration: conjunto de posibles valores que puede tomar el dato
- whiteSpace: controla la forma que tendrá el contenido de este dato una vez haya sido procesado; puede tomar los siguientes valores:
  - “preserve”: los datos no se modifican, quedan tal y como aparecen escritos.
  - “replace”: los tabuladores, saltos de línea y retornos de carro son sustituidos por espacios.
  - “collapse”: hace lo mismo que “replace”, pero además sustituye espacios múltiples por un solo espacio.

Estas facetas se pueden combinar. Veamos algunos ejemplos, en los que las facetas se aplican a definiciones de elementos.

Ejemplo 1: facetas “minInclusive” y “maxInclusive”,mínimo valor 2 y máximo valor 20.

```
<xs:element name="edad">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                <xs:minInclusive value="2"/>
                <xs:maxInclusive value="20"/>
        </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Define el elemento “edad” de tipo simple, como un entero en el rango [0-120].

Ejemplo 2: facetas “minLength” y “maxLength”.

```
<xs:element name="edad">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                <xs:minLength value="0"/>
                <xs:maxLength value="120"/>
        </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Ejemplo 3: Define el elemento “contraseña” como una cadena de entre 5 y 8 caracteres.

```
<xs:element name="contraseña">
        <xs:simpleType>
                <xs:restriction base="xs:String">
                <xs:minLength value="5"/>
                <xs:maxLength value="8"/>
        </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Crear un tipo enumerado (xs:enumeration)

Una forma muy útil de utilizar facetas es crear tipos enumerados para limitar los valores que puede tomar un elemento o atributo. Ejemplo:

```
<xsd:attribute name="demanda">
        <xsd:simpleType>
                <xs:restriction base="xsd:string">
```

```
            <xs:enumeration value="bajo"/>
            <xs:enumeration value="medio"/>
            <xs:enumeration value="alto"/>
            </xs:restriction>
        </xs:simpleType>
</xsd:attribute>
```

Se define el atributo demanda como una restricción del tipo base “xsd:string” donde sólo existen tres valores posibles “alto” “medio” y “bajo”. El atributo “demanda” debe tomar uno de esos tres valores.

Listas (xsd:list)

Las listas son similares a la faceta “enumeration”, pero permiten incluir valores múltiples, separados mediante espacios. Las listas permiten elegir el tipo de datos de los valores en la lista, mediante el atributo "itemType”, incluyendo tipos definidos por el usuario. También se pueden aplicar otras facetas, como “length”, etc. Ejemplo:

```
<xsd:simpleType name="posiblesTiendas">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="oBoy"/>
            <xsd:enumeration value="Yoohoo!"/>
            <xsd:enumeration value="ConJunction"/>
            <xsd:enumeration value="Anazone"/>
        </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="listaTiendas">
        <xsd:list itemType="posiblesTiendas"/>
</xsd:simpleType>
<xsd:simpleType name="tiendas">
        <xsd:restriction base="listaTiendas">
            <xsd:maxLength value="3"/>
        </xsd:restriction>
</xsd:simpleType>
```

Se define el tipo “tiendas” como una restricción del tipo “listaTiendas” donde “length no puede ser mayor de tres. Es decir, en “tiendas” puede haber hasta tres valores de la list. El tipo “listaTiendas” se define como una lista donde cada elemento de la lista es del tipo “posiblesTiendas”. Este último se define como un “enumeration”. Se puede por ejemplo definir un elemento llamado “vendedores” de la siguiente manera:

```
<xsd:element name=”vendedores” type=”tiendas”/>
```

Este elemento puede tomar hasta tres valores de los que se ha especificado en el tipo “posiblesTiendas”. Por ejemplo, en un documento instancia XML podríamos encontrarnos:

```
<tiendas> oBoy Yoohoo!</tiendas>
```

Añadir atributos a tipos simples por extensión (xsd:extension)

Sabemos que los elementos de tipos simples son los que sólo pueden contener datos carácter, pero no atributos ni elementos hijo. Por otro lado, los elementos de tipo complejo pueden tener tanto atributos como elementos hijo. ¿Qué pasa si queremos que un elemento pueda tener atributos, pero no elementos hijo? Existe una forma de hacer esto, utilizando los elementos "xsd:simpleContent" y "xsd:extension". Veámoslo con un ejemplo:

```
<xsd:element name="nombreOriginal">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                        <xsd:attribute name="confirmado" default="no"/>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:element>
```

En este ejemplo definimos el elemento "nombreOriginal" de tipo complejo, pero al usar el elemento <xsd:simpleContent> estamos diciendo que el contenido de "nombreOriginal" tiene que ser sólo datos carácter. Sin embargo, este elemento sí tiene un atributo. Esto se especifica definiendo el elemento "xsd:simpleContent" como una extensión del tipo base "xsd:string" a la que se le añade el atributo "confirmado", cuyo valor por defecto es "no"

Métodos de diseño XSD XML Schema

Hay varias maneras de abordar el diseño de un XML schema. Usaremos una u otra, o una combinación de varias, dependiendo de factores tales como la complejidad, extensión y el tipo de documentos que estamos definiendo (por ejemplo, sin son documentos donde predomina una colección de datos estructurados, o son documentos con mucho texto libre). A continuación se describen tres formas de diseñar XML schemas.

Diseño Anidado o de muñecas rusas..

Se llama así porque se anidan declaraciones de elementos unas dentro de otras. Se describe cada elemento y atributo en el mismo lugar donde se declaran. Consiste en, partiendo de un documento XML, seguir la estructura del mismo e ir definiendo los elementos que aparecen en el mismo de forma secuencial, incluyendo la definición completa de cada elemento en el mismo orden en el que aparecen en el documento instancia. Este método de diseño es muy sencillo, pero puede dar lugar a esquemas XML difíciles de leer y mantener cuando los documentos son complejos. Además produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones. Es el método de diseño menos recomendable.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http;//www.w3.org/2001/XMLSchema">
   <xs:element name="libro">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="titulo" type="xs:string"/>
            <xs:element name="autor" type="xs:string"/>
            <xs:element name= "personaje" minOccurs= "O"
                  maxOccurs =" unbounded" >
               <xs:complexType>
                  <xs:sequence>
```

```
                    <xs:element name="nombre" type="xs:string"/>
                    <xs:element name="amigoDe" type= "xs : string"
minOccurs="O" maxOccurs="unbounded"/>
                    <xs:element name="desde" type="xs:date"/>
                    <xs:element name="calificación" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Diseño Plano o de uso de referencias a elementos y atributos

En este método primero se definen los diferentes elementos y atributos, para después referenciarlos utilizando el atributo “ref”. La declaración y su definición están en diferentes sitios. Esta técnica es la que más se parece al diseño con DTDs. En el ejemplo que mostramos a continuación se comienza a definir los elementos más simples para terminar con los más complejos, pero puede hacerse al revés (como en los DTDs) comenzando por definir los elementos de mayor nivel y, descendiendo, ir definiendo los elementos de menor nivel hasta llegar a los elementos simples.

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

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

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

    <!-- definición de elementos de tipo complejo -->
    <xs:element name="personaje">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nombre"/>
                <xs:element ref="amigoDe" minOccurs="O" maxOccurs="unbounded"/>
                <xs:element ref="desde"/>
                <xs:element ref="calificación"/>
                <!-- los elementos de tipo simple se referencian ref -->
                <!-- la definición de cardinalidad en elemento referenciado-->
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
```

```
            <xs:element ref="titulo"/>
            <xs:element ref="autor"/>
            <xs:element ref="personaje" minOccurs="O" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute ref="isbn"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Diseño con tipos con nombre.

Con este método se definen clases o tipos utilizando los elementos de XML Schema “simpleType” y “complexType” con un nombre. Estos tipos definidos se pueden utilizar mediante el tributo “type” de los elementos. Este mecanismo permite reutilizar de definiciones de tipos en diferentes puntos del documento. Es el método más aconsejado ya que permite la reutilización. En el ejemplo, al igual que en el caso anterior, se ha seguido un diseño ascendente (definiendo primero el de menor nivel) pero se puede hacer un diseño descendente.

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

   <!-- definición de tipos simples-->
   <xs:simpleType name="TipoNombre" >
      <xs:restriction base="xs:string">
      <xs:raaxLength value="32"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="TipoDesde">
      <xs:restriction base="xs:date"/>
   </xs:simpleType>
   <xs:simpleType name= "TipoDescripcion" >
      <xs:restriction base="xs:string"/>
   </xs:simpleType >
   <xs:simpleType name="TipoISBN">
      <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{13}"/>
      </xs:restriction>
   </xs:simpleType>

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

   </xs:complexType>
   <xs:complexType name="TipoLibro">
      <xs:sequence>
         <xs:element name="titulo" type="TipoNombre"/>
         <xs:element name="autor" type="TipoNombre"/>
```

```
            <xs:element name="personaje" type="TipoPersonaje" minOccurs="O"/>
        </xs:sequence >
        <xs:attribute name="isbn" type="TipoISBN" use="required"/>
    </xs:complexType>

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

Ejercicio.- Se dispone del siguiente documento XML, persona.xml

```
<?xml version='1.0' encoding="UTF-8"?>
<persona dni='12345678-L' xsi:noNamespaceSchemaLocation='persona.xsd'
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<nombre>Juan Antonio</nombre>
<apellido>Abascal</apellido>
<estadoCivil>Soltero</estadoCivil>
<edad>60</edad>
<enActivo/>
</persona>
```

El esquema XML asociado se quiere que cumpla ciertas restricciones semánticas:
El atributo dni es obligatorio y su formato es de 8 dígitos, un guión, una letra mayúscula
El estado civil puede ser:Soltero, Casado, Divorciado o viudo. Por defecto es Soltero
La edad debe ser de 0 a 150
El elemento <enActivo> es optativos
Crear el esquema que cumpla con esto en persona.xsd.

Construcción avanzada de esquemas.-

Grupos de elementos y grupos de atributos.-

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

Ejemplo.-

```
<!--definición de un grupo de elementos-->
<xs:group name='ElementosPrincipalesLibro'>
<xs:sequence>
<xs:element name='titulo' type='TipoNombre'/>
<xs:element name='autor' type='TipoNombre'/>
</xs:sequence>
</xs:group>
<!-- definición de un grupo de atributos-->
<xs:attributeGroup name='AtributosLibros'>
<xs:attribute name='isbn' type='ISBN' use='required'/>
<xs:attribute name='disponible' type='xs:string'/>
</xs:attributeGroup>
<!-- Lo grupos se usan en la definición de un tipo complejo-->
<xs:complexType name='TipoLibro'>
<xs:sequence>
<xs:group ref='ElementosPrincipalesLibro'/>
<xs:element name='personaje' tyype='TipoPersonaje' minOccurs='0' maxOccurs='unbounded'/>
</xs:sequence>
<xs:attributeGroup ref='AtributosLibros'/>
</xs:complexType>
```

Redefinición de Tipos.-

Este elemento nos va a permitir redefinir en un esquema un tipo de datos, simple o complejo, un grupo de elementos o un grupo de atributos, que hubieran sido ya definidos en un documento de esquema al que se referencia. Es semejante a una extensión o herencia de una clase en programación orientada a objeto.
Xs:redefine.-
elemento padre:xs:schema
atributos obligatorios: schemaLocation:URI del documento de esquema al que se referencia
Atributos optativos principales:
id:especifica un identificador único para el elemento

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

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

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

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

```
<?xml version='1.0'?>
 <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 <xs:redefine schemaLocation="esquemaInicial.xsd">
 <xs:complexType name="TipoTrayecto">
    <xs:complexContent>
    <xs:extension base='TipoTrayecto'>
       <xs:sequence>
          <xs:element name='duracion'/>
       </xs:sequence>
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>
</xs:redefine>
<xs:element name='ruta' type='TipoTrayecto'/>
</xs:schema>
```

Grupos de sustitución.-
Permite definir elementos que son sinónimos. Un uso muy habitual es para definir los mismos elementos en varios idiomas.

Ejemplo.-

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

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

<xs:complexType name='TipoInfoCliente'>
   <xs:sequence>
      <xs:element ref='nombre'/>
   </xs:sequence>
</xs:complexType>
<xs:element name='cliente' type='TipoInfoCliente'/>
<xs:element name='customer' substitutionGroup='cliente'/>
```

Dos fragmentos xml igualmente válidos.-

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

Y también

<customer><name>José Rodríguez</name></customer>

xs:any

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

Xs:anyAttribute

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

Xs:include
Es un componente de inclusión de esquemas externos. Añade las declaraciones y definiciones de un esquema externo al esquema actual. El documento de esquema externo debe tener el mismo espacio de nombres que el esquema actual. Se utiliza para reutilizar tipos ya definidos, en ocasiones en librerías de tipos.
Elemento padre:xs:schema
Atributos obligatorios:
-schemaLocation:URI del esquema a incluir en el espacio de nombres del esquema contenedor.
Atributos optativos principales:
id: especifica un identificador único para el documento.

Ejemplo.-

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

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

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

xs:import
Es un componente de importación de esquemas externos.Ofrece la misma funcionalidad que xs:include a excepción que el esquema importado tiene diferente espacio de nombres que el actual.
Elemento padre: xs:schema
atributos optativos principales: id especifica un identificador único para el elementos
namespace: indica el URI del esquema del espacio de nombres importado.
SchemaLocation: especifica el URI del esquema del espacio de nombres importado
Ejemplo.-

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

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

<xs:import namespace="http://www.w3.org/1999/xhtml"/>
…..
<xs:ComplexType name="UntipoComplejo">
<xs:sequence>
<xs:element ref="xhtml:p" minOccurs="0"/>
..
</xs:sequence>
</xs:CompleType>
…
<xs:schema>
```