Práctica 3

Sesión 4

Objetivos

- Comprender los fundamentos básicos de XML y su utilidad.
- Entender los conceptos de documento bien formado y documento válido.Conocer cómo crear nuevos lenguajes de marcas mediante un DTD (Document Type Definition) y XML Schema.
- Comprender las limitaciones del DTD y cómo las suple XML Schema.

eXtensible Markup Language

eXtensible Markup Language (XML) es un lenguaje de marcas desarrollado por el W3C que se emplea para estructurar, representar y transportar información, como serían documentos, datos, configuraciones, libros, transacciones, facturas, y muchos más. Es un formato ampliamente utilizado para compartir información estructurada entre programas, personas, ordenadores y personas, tanto de forma local como a través de redes (W3C, http://www.w3.org/standards/xml/core). XML puede ser procesado automáticamente e intercambiado entre diversos hardware, sistemas operativos y aplicaciones.

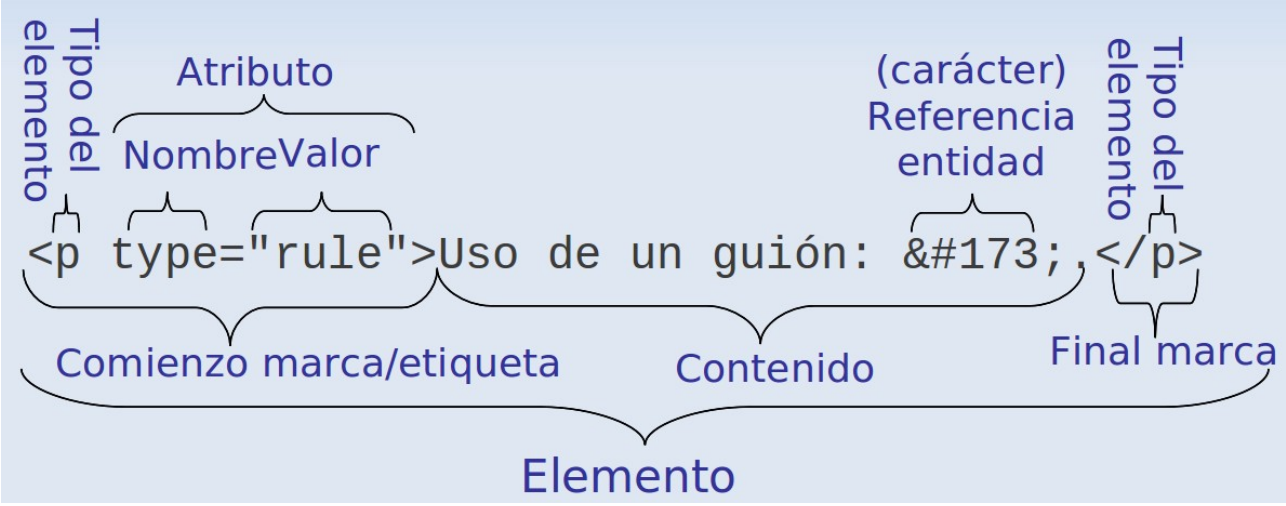
Es muy importante diferenciar el hecho de que XML es una herramienta para representar y transportar datos, mientras que HTML se orienta a formatear contenidos y mostrarlos. Por tanto, el primero se centra en los datos en sí, mientras que el segundo en cómo se muestran éstos. No son lenguajes competidores, sino complementarios.

XML en sí no hace nada por sí sólo, salvo dotar de una estructura a los datos y nos permite describir el sentido o la semántica de estos. Con XML podemos inventarnos nuestras propias etiquetas (eXtensible). Más aún, podemos crear nuestros propios lenguajes de marcado, razón por lo que se denomina a XML un metalenguaje (un lenguaje para crear lenguajes de marcado). De hecho, XML ha dado lugar a multitud de ellos bien conocidos (XHTML, WAP, RSS, RDF, OWL, etc.), y a su vez surge como una simplificación de otro metalenguaje mucho más complejo y extenso, denominado SGML (Standard Generalized Markup Language).

Documentos XML

XML se basa en el concepto de documento, el cual está compuesto por elementos. Éstos a su vez pueden contener otros elementos o contenido textual propiamente dicho. Los elementos pueden disponer de atributos, con valores asociados, y representan alguna propiedad de estos. Como vemos, conceptualmente es la misma estructura que tienen los elementos de un documento HTML.

En esta figura, vemos la composición de un elemento (http://www.ukoln.ac.uk/metadata/dcmi/dc-elem-prop/#sec2):



Un documento XML, por tanto, se puede representar en forma de árbol, compuesto de nodos. Existen varios tipos:

- Raíz: nodo abstracto del que cuelga el documento completo. De él cuelgan nodos hijos que pueden ser otros elementos, comentarios, instrucciones y el nodo del elemento raíz del documento.
- Nodo elemento: contiene un elemento con un nombre, un conjunto de atributos y una lista de hijos.
- Nodo texto: texto que se encuentra entre las marcas de apertura y cierre asociadas a un elemento.
- Nodo comentario: texto de la forma *<!-- Esto es un comentario -->*.
- Nodo de instrucciones de procesado: contiene instrucciones para, por ejemplo, diseñar la presentación de un documento.

Los elementos son los componentes lógicos de un documento y pueden contener texto, otros elementos o estar vacíos. Pueden tener elementos padres e hijos.

```
<elemento> … </elemento> <!-- Por ejemplo: -->
<negrita> Esto es un ejemplo de <italica> elemento </italica> dentro de otro </negrita>
```

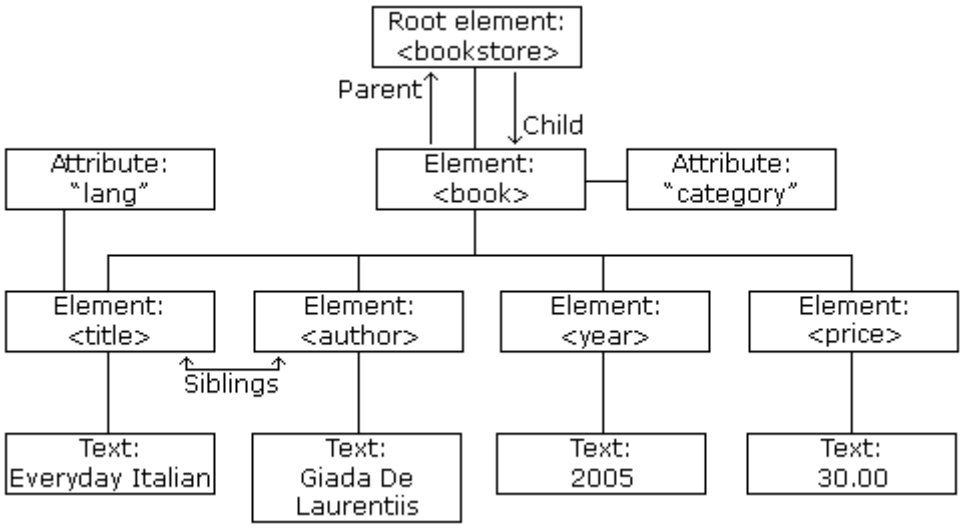
Los atributos se emplean para asociar pares (nombre, valor) a los elementos y suelen ofrecer información que no son parte de los datos en sí (o sí :-)). ¿Cuándo introducir un atributo a un elemento o un elemento nuevo? Una reglilla nos dice que las propiedades que pueden considerarse metadatos (datos sobre los propios elementos) figurarán como atributos, mientras que los que puedan considerarse datos en sí mismos, figurarán como elementos:

```
<elemento atributo="valor"> … </elemento> <!-- Por ejemplo: -->
<persona sexo="hombre">
        <nombre>Juan</nombre>
        <apellido>Fernández</apellido>
</persona>
```

El siguiente ejemplo muestra un fragmento de un documento XML, que contiene datos básicos de los libros que hay en una librería, y su árbol asociado (http://www.w3schools.com/xml/xml_tree.asp):

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

```
<bookstore>
	<book category="COOKING">
		<title lang="en">Everyday Italian</title>
		<author>Giada De Laurentiis</author>
		<year>2005</year>
		<price>30.00</price>
	</book>
	<book category="CHILDREN">
		<title lang="en">Harry Potter</title>
		<author>J K. Rowling</author>
		<year>2005</year>
		<price>29.99</price>
	</book>
	<book category="WEB">
		<title lang="en">Learning XML</title>
		<author>Erik T. Ray</author>
		<year>2003</year>
		<price>39.95</price>
	</book>
</bookstore>
```



El elemento raíz es *bookstore*, el cual tiene tres nodos hijos, todos del elemento *book*. Dentro de cada uno de ellos, nos encontramos nodos asociados a los elementos *title*, *author*, *year*, *price*, los cuales tienen un nodo hijo, que es de tipo texto.

Con la primera línea de este ejemplo estamos indicando que este fichero corresponde a un documento XML.

Reglas sintácticas de XML

Las reglas que guían la creación de documentos XML son simples y fáciles de utilizar. Son las

siguientes:

- Todos los elementos tienen que llevar una etiqueta de cierre.
- XML es sensible a las mayúsculas, lo cual hay que tener en cuenta al escribir las etiquetas y atributos (<receta> es diferente de <Receta>).
- Los elementos deben estar anidados correctamente. Este ejemplo es incorrecto:

```
<capitulo> <seccion> título </capitulo> </seccion>
```

- Todos los documentos deben llevar un nodo raíz.
- Los valores de los atributos deben ir entre comillas dobles o simples.
- Los caracteres especiales en XML se incluyen por medio de entidades (< corresponde al símbolo '<').
- Los espacios en blanco se preservan.
- Los comentarios tienen la misma sintaxis que en HTML.
- Los nombres de elementos pueden contener letras, números y otros caracteres, pero no pueden empezar por números o por caracteres de puntuación, ni contener espacios en blanco. Tampoco pueden empezar por "xml". En general, inténtese no emplear guiones (-), puntos (.) y dos puntos (:).

  Se dice que un documento está **bien formado**, cuando cumple las reglas sintácticas anteriores.

Ejercicio:

Escribe un documento XML que contenga entradas de una agenda de contactos, la cual se divide en dos partes: la del trabajo, donde aparecen los contactos profesionales, y la personal, donde estaría el resto. Cada contacto podrá tener varios teléfonos (casa, móvil, trabajo, etc.) y correos electrónicos.

Document Type Definition

Se dice que un documento XML **es válido** cuando está bien formado y ha sido correctamente validado frente a un DTD, el cual es una especificación que almacena la descripción de la estructura de un documento XML, indicando los elementos posibles, y sus atributos, y cómo se relacionan entre ellos. En otras palabras, es válido cuando está bien formado y sigue las reglas estructurales de formación del documento y el nombre de los elementos y atributos, información que usualmente especificadas en un *Document Type Definition (DTD)*. Los DTDs suelen disponerse en ficheros con extensión .dtd (externos), aunque también pueden ir incluidos directamente en el fichero XML (internos).

De forma concreta, un DTD lista los nombres de los elementos que pueden aparecer en los documentos XML que sigan el DTD, qué elementos pueden aparecer en combinación con otros, cómo se pueden anidar y qué atributos están disponibles y en qué elementos.

Para indicar qué DTD externo es el que sigue un documento XML se incluye una línea en la que referencia el nombre del fichero que almacena el DTD:

```
<?xml version="1.0"?>
<!DOCTYPE agenda SYSTEM "agenda.dtd">
<agenda>
      …
</agenda>
```

En este ejemplo, se está indicando mediante *DOCTYPE* que el fichero *agenda.dtd* contendrá las reglas que definen el documento cuyo elemento raíz es *agenda*.

Para declarar un elemento se emplea ELEMENT. Existen varias posibilidades:

–Elementos vacíos: `<!ELEMENT nombre_del_elemento EMPTY>`

```
<!ELEMENT linea_horizontal EMPTY>
```

Y su uso sería:

```
<linea_horizontal/>
```

–Elementos que sólo contienen texto, el cual será analizado por los analizadores (*parsers*):

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

–Elementos que sólo contienen texto, pero que no será analizado por los parsers:

```
<!ELEMENT nombre (#CDATA)>
```

–Elementos que contienen secuencias de elementos:

```
<!ELEMENT nombre (elemento_hijo1, elemento_hijo2, …)>
```

Por ejemplo:

```
<!ELEMENT mensaje (de, para, asunto, cuerpo)>
```

–Elementos que contienen sólo una ocurrencia de un elemento:

```
<!ELEMENT nombre (elemento_hijo1)>
```

Por ejemplo:

```
<!ELEMENT localidad (provincia)>
```

–Elementos que contienen una o más ocurrencias de un elemento:

```
<!ELEMENT nombre (elemento_hijo1+)>
```

Por ejemplo:

```
<!ELEMENT capitulo (seccion+)>
```

–Elementos que contienen cero o más ocurrencias de un elemento:

```
<!ELEMENT nombre (elemento_hijo1*)>
```

Por ejemplo:

```
<!ELEMENT texto (linea*)>
```

–Elementos que contienen cero o una ocurrencia de un elemento:

```
<!ELEMENT nombre (elemento_hijo1?)>
```

Por ejemplo:

```
<!ELEMENT titulo (subtitulo?)>
```

–Elementos que contienen elementos alternativos:

```
<!ELEMENT nombre (elemento_hijo1 | elemento_hijo2)>
```

Por ejemplo:

```
<!ELEMENT nota ( mensaje_sms | mensaje_whatsup)>
```

–Elementos que contienen texto u otros elementos:

```
<!ELEMENT nota (#PCDATA | de | para | cabecera | mensaje)*>
```

Así, y según las necesidades, se pueden realizar todas las combinaciones que nos interesen de estas especificaciones de elementos.

Ejercicio:

Escribe un documento XML, llamado ejemplo_libro.xml, que esté bien formado con respecto al DTD siguiente almacenado en el fichero book.dtd:

```
<!ELEMENT p (#PCDATA)>
<!ELEMENT BOOK  (OPENER,SUBTITLE?,INTRODUCTION?,(SECTION | PART)+)>
<!ELEMENT OPENER   (TITLE_TEXT)*>
<!ELEMENT TITLE_TEXT  (#PCDATA)>
<!ELEMENT SUBTITLE    (#PCDATA)>
<!ELEMENT INTRODUCTION (HEADER, p+)+>
<!ELEMENT PART       (HEADER, CHAPTER+)>
<!ELEMENT SECTION     (HEADER, p+)>
<!ELEMENT HEADER      (#PCDATA)>
<!ELEMENT CHAPTER     (CHAPTER_NUMBER, CHAPTER_TEXT)>
<!ELEMENT CHAPTER_NUMBER (#PCDATA)>
<!ELEMENT CHAPTER_TEXT (p)+>
```

Este formato anterior es el que se emplea cuando el DTD se especifica de forma externa. En el caso de no ser externo al fichero XML, aparecería de la siguiente forma dentro del fichero ejemplo_libro.xml:

```
<?xml version="1.0"?>
<!DOCTYPE BOOK [
  <!ELEMENT p (#PCDATA)>
  <!ELEMENT BOOK  (OPENER,SUBTITLE?,INTRODUCTION?,(SECTION | PART)+)>
  <!ELEMENT OPENER   (TITLE_TEXT)*>
  <!ELEMENT TITLE_TEXT  (#PCDATA)>
  <!ELEMENT SUBTITLE    (#PCDATA)>
  <!ELEMENT INTRODUCTION (HEADER, p+)+>
  <!ELEMENT PART       (HEADER, CHAPTER+)>
  <!ELEMENT SECTION     (HEADER, p+)>
  <!ELEMENT HEADER      (#PCDATA)>
  <!ELEMENT CHAPTER     (CHAPTER_NUMBER, CHAPTER_TEXT)>
  <!ELEMENT CHAPTER_NUMBER (#PCDATA)>
  <!ELEMENT CHAPTER_TEXT (p)+>
]>
<BOOK>
        <OPENER>
                ...
```

```
    </OPENER>
    ...
</BOOK>
```

Para declarar un atributo se emplea ATTLIST según el formato:

```
<!ATTLIST nombre_de_elemento nombre_de_atributo tipo_de_atributo
        #FIXED valor_por_defecto #REQUIRED #IMPLIED>
```

Por ejemplo, podemos crear un atributo denominado *tipo* asociado a un elemento *pago*, con el valor *tarjeta* por defecto:

```
<!ATTLIST pago tipo #CDATA "tarjeta">
```

El valor por defecto significa que si no se asigna explícitamente, como en

```
<pago tipo="cheque"/>
```

y se escribe

```
<pago/>
```

automáticamente se le asigna "tarjeta" al atributo. Si se aplica la palabra #REQUIRED en la definición del atributo, se obliga a asignarle un valor, aun cuando haya valor por defecto. En el caso de #IMPLIED, no se fuerza a incluir un atributo, aun cuando no haya valor por defecto. Por el contrario, con #FIXED se obliga a que el atributo tome un valor fijo.

CDATA significa que el atributo contiene caracteres, sin ninguna restricción. Otras alternativas para indicar tipos de atributos puedes verlas en http://www.w3schools.com/dtd/dtd_attributes.asp.

Por último, también se pueden incluir entidades dentro de un DTD, las cuales son atajos a textos o a caracteres especiales. Se definen como sigue:

```
<!ENTITY nombre_entidad "valor">
```

Por ejemplo:

```
<!ENTITY autor "Juan Manuel"> <!-- ó -->
<!ENTITY copyright "Copyright DECSAI">
```

Y su uso sería:

```
<autoria>&autor; - &copyright;</autoria>
```

También se pueden referenciar entidades que están definidas en otros DTDs mediante su dirección web (URL):

```
<!ENTITY nombre_entidad  SYSTEM "URI/URL">
```

Por ejemplo:

```
<!ENTITY autor SYSTEM "http://decsai.ugr.es/entidades.dtd"> <!-- ó -->
<!ENTITY copyright "http://decsai.ugr.es/entidades.dtd">
```

Ejercicio:

Diseña un DTD, denominado receta.dtd, que contenga las reglas para definir una colección de recetas de cocina, incorporando datos de nutrición para cada una de ellas en forma de atributos (proteínas, hidratos de carbono, grasas y calorías).

Dado un documento XML, para ver si es válido, emplearemos los programas conocidos como validadores. En este caso, cualquier error que se encuentre hará que finalice el proceso. Existen una gran cantidad de validadores en línea. Algunos ejemplos son:

–http://validator.w3.org/

–http://www.w3schools.com/xml/xml_validator.asp

–http://www.xmlvalidation.com/

–http://www.stg.brown.edu/service/xmlvalid/

XML Schema

El uso de los DTDs tiene un par de limitaciones importantes: la primera es que el DTD no está en XML, lo cual dificulta su procesado automático; por otro lado, es muy pobre en cuestión de tipos de datos. Estas y otras razones originaron que se diseñara un mecanismo nuevo para definir la estructura de documentos XML mucho más rico y potente, considerando una amplia gama de tipos de datos, y escrito en XML también: *XML Schema Definition (XSD)*.

XML Schema se fundamenta en el concepto de espacios de nombres (namespaces), que es una colección de elementos XML y atributos, identificados grupalmente de forma única. También se denomina vocabulario XML. El principal objetivo para definirlos es evitar conflictos cuando se emplean múltiples vocabularios.

Las definiciones en XML Schema, al ser también documentos XML, parte de elementos y atributos que están definidos en un espacio de nombres. Concretamente, viene de http://www.w3.org/2001/XMLSchema, el cual es un espacio de nombres reservado que contiene los elementos y atributos definidos por W3C para la estructura de XML Schema. Utilizando estos componentes, podemos definir nuevos elementos y atributos, y a su vez nuevos espacios de nombres (conocidos como target namespace).

Veamos seguidamente cómo se plasman estos conceptos al construir un XSD. El elemento raíz de cualquier documento XSD, es *schema*, el cual, puede contener varios atributos:

```
<?xml version="1.0"?>
<xs:schema      xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.w3schools.com/mensaje.xsd"
                xmlns="http://www.w3schools.com/mensaje.xsd"
                elementFormDefault="qualified"
                attributeFormDefault="unqualified">
                ...
                ...
</xs:schema>
```

El primero, *xmlns:xs* indica que los elementos y tipos empleados en el XSD vendrán del espacio de nombres estándar, *http://www.w3.org/2001/XMLSchema*, y que estarán precedidos por el prefijo "*xs:*". Por otro lado, el valor del atributo *targetNamespace*, establecerá el espacio de nombres de los elementos que se definan en este esquema (si no aparece este atributo, se considerará el espacio "no namespace"). El atributo *xmlns* albergará el espacio de nombres por defecto, y por último, *elementFormDefault* mediante el valor *qualified* establece que los documentos que se creen a partir de este XSD deben aparecer cualificados por el espacio de nombres que se defina en este XSD. Análogamente para los atributos, con *unqualified* en *attributeFormDefault*, estamos indicando que éstos

no deberán ir cualificados.

Por otro lado, un documento creado bajo el amparo de este XSD (si suponemos que éste define la estructura de mensajes de correo electrónico), tendrá a su vez la siguiente forma:

```
<?xml version="1.0"?>
<mensaje xmlns="http://www.w3schools.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3schools.com mensaje.xsd">
      <!-- ó xsi:noNamespaceSchemaLocation="mensaje.xsd"  -->
    <para>Juan</para>
    <de>Jose</de>
    <asunto>Recordatorio reunión de mañana</asunto>
    <cuerpo>No olvides la reunión de mañana por la tarde. Saludos.</cuerpo>
</mensaje>
```

El par (*xmlns*, "*http://www.w3schools.com*") le indica al validador de esquemas, que todos los elementos que se emplean en este documento XML están declarados en el espacio de nombres indicado. Una vez que se indica el espacio de nombres para instancias XML (documentos XML que se derivan de XSD) mediante el atributo *xmlns:xsi*, se puede emplear el atributo que nos permite especificar el esquema empleado, así como su ubicación. Para ello se usa el atributo *xsi:schemaLocation* y se le asigna los dos valores anteriormente indicados. En caso de que en el XSD no aparezca *targetNamespace* como atributo de *schema*, se emplea el atributo *noNamespaceSchemaLocation* y el valor que toma es el nombre del fichero XSD correspondiente.

El atributo mínimo para el elemento schema del XSD es *xmlns:xs*, mientras que para una instancia XML (documento), *xmlns:xsi* y *xsi:noNamespaceSchemaLocation*.

El diagrama de la Figura 1 (http://www.oracle.com/technetwork/articles/srivastava-namespaces-092580.html) ilustra gráficamente la definición de un XSD y la creación de una instancia XML. En esta misma página, se puede encontrar una descripción más detallada de los espacios de nombres y su uso.

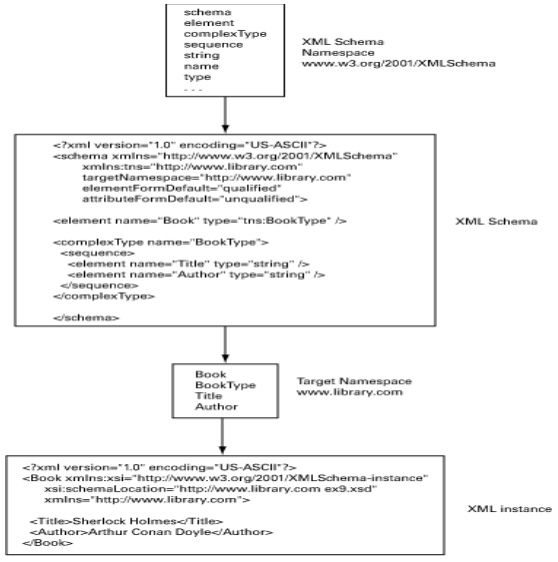


Ilustración 1: Del XML Schema a la instancia XML Schema

Definición de elementos

Veamos seguidamente cómo definir elementos simples, que son aquellos que sólo pueden contener texto (ni otros elementos ni atributos):

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

De esta forma indicamos el nombre del elemento y su tipo, que puede ser *xs:date*, *xs:string*, *xs:integer*, *xs:decimal*, *xs:boolean*, *xs:date* y *xs:time*. Por ejemplo:

```
<xs:element name="fecha_factura", type="xs:date"/>
<xs:element name="nombre", type="xs:string"/>
```

Si añadimos el atributo *default* al elemento, podremos indicar un valor por defecto si aparece un elemento sin haber especificado ningún valor explícitamente. El atributo *fixed* establece un valor fijo para el elemento.

```
<xs:element name="color", type="xs:string" default="rojo"/>
<xs:element name="cumpleanios", type="xs:date" fixed="2013-01-01"/>
```

Se pueden establecer una serie de restricciones sobre los valores que puede tomar un elemento. Así, en este ejemplo define el atributo *edad* y se establece una restricción en los valores posibles:

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

Se indica que el elemento edad puede tomar como mínino el valor 0 y como máximo el valor 120. Esta es otra forma de definir el elemento edad a partir de un tipo simple:

```
<xs:element name="edad" type="tipoEdad/>
    <xs:simpleType name="tipoEdad">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Véase http://www.w3schools.com/schema/schema_facets.asp para conocer las restricciones existentes.

Alternativamente, los tipos complejos en XSD nos permiten crear contenedores de definiciones de elementos, especificando qué nodos hijos podría contener, así como sus atributos. Por ejemplo, podríamos decir que un cliente está formado por un nombre y una dirección. Esto se haría como sigue:

```
<xs:element name="cliente">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="nombre" type="xs:string" />
     <xs:element name="direccion" type="xs:string" />
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Con el elemento *xs:complexType* se indica que se está creando un tipo de datos complejo, que está formado por otros elementos. Básicamente es un contenedor de definiciones de otros elementos. *xs:sequence* es un elemento que indica que se va especificar la declaración de otros elementos, pero éstos deben aparecer en el documento XML en el orden en el que se han definido. Si apareciera *xs:choice*, indicamos que sólo uno de los elementos definidos podría aparecer, mientras que *xs:all*, se indica que los elementos descritos pueden ocurrir en cualquier orden. Estos elementos se denominan indicadores de orden (orders indicators) y establece el orden de los elementos.

La cardinalidad, es decir, el número de ocurrencias de un elemento, se fija con los atributos *minOccurs* y *maxOccurs*. Si no aparecen, se asignan el valor 1 por defecto, indicando que sólo aparecerá una ocurrencia del elemento. Si *minOccurs* tiene el valor 0, significa que puede que no aparezca, y por tanto, que el elemento sea opcional. Si *maxOccurs* toma el valor "*unbounded*", el elemento puede aparecer todas las veces que se considere. Por ejemplo:

```
<xs:element name="Asignatura", type="xs:string", minOccurs="1" maxOccurs="10"/>
```

Estos atributos se conocen como indicadores de ocurrencia (*occurrence indicators*).

Por último, existen los denominados indicadores de grupo, que no es más que un método para agrupar elementos y atributos, indicando su orden correspondiente, para luego luego integrarlos en una definición de elemento:

```
<xs:group name="direccion">
        <xs:sequence>
                <xs:element name="calle" type="xs:string"/>
                <xs:element name="numero" type="xs:decimal"/>
                <xs:element name="codigoPostal" type="xs:integer"/>
        </xs:sequence>
</xs:group>
<xs:element name="empresa" type="informacionEmpresarial"/>
<xs:complexType name="informacionEmpresarial">
        <xs:sequence>
                <xs:group ref="direccion"/>
                <xs:element name="nombre" type="xs:string"/>
        </xs:sequence>
</xs:complexType>
```

En este ejemplo agrupamos elementos, con su indicador de orden correspondiente, y luego se emplea en un tipo complejo. Esta también es una definición que nos permite crear tipos complejos que

luego utilizaríamos en otros lugares.

También se puede crear un tipo complejo a partir de otro tipo complejo, extendiéndolo:

```
 <xs:element name="empleado" type="informacionPersonalCompleta"/>
<xs:complexType name="informacionPersonal">
      <xs:sequence>
           <xs:element name="nombre" type="xs:string"/>
           <xs:element name="apellidos" type="xs:string"/>
      </xs:sequence>
</xs:complexType>

<xs:complexType name="informacionPersonalCompleta">
      <xs:complexContent>
             <xs:extension base="informacionPersonal">
                 <xs:sequence>
                       <xs:element name="direccion" type="xs:string"/>
                            <xs:element name="cidudad"
                                        type="xs:string"/>
                            <xs:element name="pais" type="xs:string"/>
                       </xs:sequence>
            </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

El elementos *xs:any* permite indicar si se puede extender o no el número de elementos en una futura extensión, aunque esos elementos no estén definidos en el esquema.

Definición de atributos

Los atributos se definen en el XSD mediante el siguiente elemento:

```
<xs:attribute name="nombre_atributo" type="tipo_del_atributo" use="valor"/>
```

Para indicar que un atributo puede ser opcional (por defecto), el valor de use será *optional*, mientras que para establecer que sea obligatorio, se asignará *required* al atributo *use*.

```
<xs:element name="Factura">
      <xs:complexType>
        <xs:attribute name="NumFactura"  type="xs:int" use="required"/>
      </xs:complexType>
</xs:element>
```

El otro indicador de grupo al que nos referíamos anteriormente, es el de atributos, que nos permite agrupar un número de ellos para luego integrarlo en un elemento:

```
<xs:attributeGroup name="grupoAtributosPersonales">
```

```
        <xs:attribute name="nombre" type="xs:string" />
        <xs:attribute name="apellidos" type="xs:string" />
        <xs:attribute name="cumpleanios" type="xs:date" />
    </xs:attributeGroup>

    <xs:element name="persona">
        <xs:complexType>
            <xs:attributeGroup ref="grupoAtributosPersonales" />
        </xs:complexType>
    </xs:element>
```

Análogamente, mediante *xs:anyAttribute* se indica que se puede extender el número de atributos, aun cuando no estén definidos en el esquema.

En el ejemplo siguiente, podemos ver cómo un elemento puede tener datos, otros elementos y atributos:

```
    <xs:element name="Parrafo">
        <xs:complexType mixed="true">
            <xs:sequence>
                <xs:element name="Negrita" type="xs:string" />
                <xs:element name="Italica" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Para tal fin, se emplea el valor *true* del atributo *mixed* del elemento *xs:complexType*. De esta manera, podríamos tener documentos con el siguiente aspecto:

```
    <Parrafo> Esto es un <Negrita>Ejemplo</Negrita> de contenido <Italica>mixto</Italica>,
     en donde encontramos elementos mezclados con elementos de datos.</Parrafo>
```

En cuanto a la estructura básica de un documento XSD, lo habitual es tener varias secciones: la primera sería para la definición de tipos simples; la segunda, para atributos, y finalmente, la tercera para tipos complejos. En http://www.w3schools.com/schema/schema_example.asp puede verse un ejemplo completo.

Ejercicio:

Diseña un XSD, denominado receta.xsd, que contenga la definición de una colección de recetas de cocina, incorporando datos de nutrición para cada una de ellas en forma de atributos (proteínas, hidratos de carbono, grasas y calorías). Intenta emplear el mayor número de elementos vistos en este guión.

Por último, comentar que los esquemas también se validan. En http://www.w3.org/2001/03/webdata/xsv o en http://schneegans.de/sv/, por ejemplo, podemos encontrar herramientas para este fin.

En http://msdn.microsoft.com/en-us/library/ms256129.aspx se puede encontrar un ejemplo de XSD y una instancia XML correspondiente.

eXtensible StyleSheet Language (XSL)

XSL es una recomendación que define transformaciones y presentaciones de documentos XML. Es un lenguaje de estilo para documentos XML. Está formada por XSL Transformation (XSLT), que permite especificar cómo transformar los documentos; Xpath, para acceder a partes de documentos XML, y XSL Formatting Objects (XSL-FO), para especificar el formato. Más información en http://www.w3.org/Style/XSL/.

Bibliografía

–http://www.w3.org/standards/xml/core

–http://www.w3schools.com/xml/

–http://www.w3schools.com/schema/default.asp

–http://es.wikipedia.org/wiki/Extensible_Markup_Language

–http://flanagan.ugr.es/xml/

–http://geneura.ugr.es/~jmerelo/xml/