CURSO 2022-2023
CICLO SUPERIOR DE DESARROLLO DE APLICACIONES WEB
IES AGUADULCE

Lenguajes de Marcas y Sistemas de Gestión de la Información

Francisco Javier Sueza Rodríguez

2 de enero de 2023

Índice general

Índice de figuras

Tema 1

Lenguajes de Marcas y Sistemas de Gestión de la Información

En esta unidad vamos a estudiar los aspectos básicos de los lenguajes de marcas y los sistemas de gestión de la información. Por un lado, veremos la evolución de los **lenguajes de marcas**, desde GML hasta HTML, así como sus elementos y atributos, haciendo especial énfasis en XML. A continuación, veremos en que consisten los **sistemas de gestión de la información**, en concreto los **ERP**, sus características, configuración básica, personalización,..etc.

1.1 Definición y Clasificación de los Lenguajes de Marcas

Los «lenguajes de marcas» sirven para **codificar un documentos**. Estos incorporan **etiquetas** o marcas con **información adicional** sobre como se estructura el texto o como se presenta. El lenguaje de marcas será el que defina que etiquetas se permiten, donde deben colocarse y que significado tienen.

Todo lenguaje de marcas esta definido en un documento denominado **DTD**, donde se establecen las marcas, los elementos utilizados por dicho lenguaje y sus correspondientes etiquetas y atributos, así como su sintaxis.

Los lenguajes de marcas se pueden clasificar, principalmente, en tres grupos:

- **Orientados a la presentación**: son los utilizados generalmente por los procesadores de texto y definen como debe presentarse el documento, es decir, el formato que tiene.
- **De procedimientos**: orientados también a la presentación, pero en este caso, dentro de un **marco procedural** que permite la definición de macros, es decir, el programa que representa el documento debe interpretar el código en el mismo orden que aparece. Algunos ejemplos son **TeX**, **LaTeX** y **Postscript**
- **Descriptivos o semánticos**: estos lenguajes no describen la presentación del documento, sino que **describen la información**, que es lo que se esta representando sin especificar como debe presentarse.

Algunos ejemplos de lenguajes de marcado agrupados por su ámbito de uso son los siguientes:

- **Documentación Electrónica**:
    - **RTF** (Rich Text Format): fue desarrollado por Microsoft en 1987 y permite el intercambio de documentos entre los diferentes procesador de texto.

- **TeX**: creado por Donald Knuth, este lenguaje esta especialmente enfocado en la creación de textos científicos. Es considerado la mejor forma de componer formulas matemáticas complejas. [1]
- **Wikitexto**: permite la creación de páginas wiki en servidores preparados para soportar este lenguaje.
- **DocBook**: permite generar documentos separando la estructura lógica del documento de su formato, permitiendo que estos documentos puedan publicarse en diferentes formatos sin tener que modificar el documento original.

- **Tecnologías de Internet**:
  - **HTML**,**XHTML** (Hypertext Markup Language, eXtensible Hypertext Markup Language): estos lenguajes están orientados a la creación de páginas web.
  - **RSS** (Really Simple Sindication): permite la difusión de contenido web mediante la sindicación de contenidos.
- Otros lenguajes especializados:
  - **MathML** (Mathematica Markup Language): especializado en expresar los formalismos matemáticos de forma que puedan ser entendidos por diferentes aplicaciones.
  - **VoiceXML** (Voice eXtended Markup Language): permite el intercambio de información entre usuarios y una aplicación con capacidad de reconocer el habla.
  - **MusicXML**: permite el intercambio de partituras entre diferentes editores de partituras.

1.2 Evolución de los Lenguajes de Marcas

A finales de los **años 60** surgen unos lenguajes informáticos, diferentes de los lenguajes de programación, orientados a la gestión de la información. Con el desarrollo de los editores y procesadores de texto surgen los primeros lenguajes informáticos orientados a la descripción y estructuración de la información: **los lenguajes de marcas**. Paralelamente también surgen otros lenguajes orientados a la representación, almacenamiento y consultar de grandes cantidades de datos: lenguajes y sistemas de bases de datos.

Los lenguajes de marcas surgieron inicialmente como lenguajes formados por un conjunto de códigos que los procesadores de textos insertaban en los documentos para dirigir el proceso de presentación (impresión) mediante una impresora. Al igual que los lenguajes de programación, estos estaban **ligados** a las características de los **procesadores de texto**y las **impresoras** en los que se usaban y no permitían a los programadores abstraerse de dichas características.

Posteriormente se añadió como medio de presentación a la pantalla y se automatizó el proceso, teniendo ya solo que pulsar una combinación teclas para lograr los resultados deseados en vez de hacerlo a mano. Este marcado estaba orientado exclusivamente a la presentación de la información, aunque posteriormente se le dieron nuevos uso surgiendo con ello el **formato generalizado**.

1.2.1 El origen: GML y SGML

Uno de los problemas que ha tenido la informática ha sido la **falta de estandarización** en los formatos de información usados por los diferentes programas.

En los años 60, **IBM** encargo a **Charles F. Goldfarb** la construcción de un sistema de edición, almacenamiento y búsqueda de documentos legales. Después de analizar el funcionamiento de la empresa se

llego a la conclusión de que necesitaban un formato estándar a todos los departamentos para gestionar la documentación.

Así fue como se creó **GML**, un formato que permitía describir los documentos de tal forma que el resultado fuese independiente de la plataforma o la aplicación utilizada. Este formato evolucionó hasta que en 1986 se creó el estándar **ISO 8879** donde se especificaba el formato **SGML**, un lenguaje muy complejo y que requería de unas herramientas de software caras, por lo que su uso quedó relegado a grandes aplicaciones industriales.

```
<email>
    <remitente>
        <nombre>Peter</nombre>
        <apellido>Pan</apellido>
    </remitente>
    <destinatario>
        <direccion>campanilla@paisdenuncajamas.com</direccion>
    </destinatario>
    <asunto>Paseo</asunto>
    <mensaje>¿Te apetece dar una vuelta?</mensaje>
</email>
```

Figura 1.2.1: Documento SGML simple

1.2.2 La Popularización: HTML

En 1990, **Tim Berners-Lee** creó el World Wide Web y conociendo SGML, se encontró con la necesidad de compatibilizar, enlazar y organizar gran cantidad de documentos procedentes de diversos sistemas. Como solución, a partir de la sintaxis de SGML, creó un lenguaje de descripción de documentos llamado **HTML**, combinando dos estándares ya existentes:

- **ASCII**: código basado en el alfabeto latino, tal como se usa en inglés moderno [2]. Cualquier procesador de textos simple puede reconocer y almacenar este formato, permitiendo la transferencia de datos entre dos ordenadores.
- **SGML**: lenguaje que permite dar estructura al texto aplicando diferentes formatos.

HTML es una **versión simplificada de SGML**, ya que solo utiliza las instrucciones absolutamente necesarias. Gracias a su simplicidad, tuvo un éxito rotundo en la World Wide Web, convirtiéndose rápidamente en el **estándar general** para la **creación de páginas web**. Actualmente, HTML es el tipo de documento más utilizado en el mundo.

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Ejemplo1</title>
    </head>
    <body>
        <p>Párrafo de ejemplo</p>
    </body>
</html>
```

Figura 1.2.2: Documento HTML simple

1.2.3 La Madurez: XML

Uno de los problemas que surgió con HTML es que la cantidad de documentos escritos en este lenguaje creció exponencialmente, muchos de los cuales no se ceñían a ningún estándar generando bastante caos. Como respuesta es ese problema, el **W3C** estableció en 1998 el estándar internacional **XML**, un lenguaje de marcas puramente estructural, que **no incluye información sobre el diseño**, y permite la creación de etiquetas adaptadas a las necesidades, convirtiéndose con rapidez en el estándar para intercambio de datos en la web.

XML es un **metalenguaje** con las siguientes características:

- Permitir definir etiquetas propias.
- Permitir asignar atributos a las etiquetas.
- Utilizar un esquema para definir de forma exacta las etiquetas y sus atributos.
- La estructura y el diseño son independientes.

En realizad XML es un **conjunto de estándares** relacionados entre sí y que comprende los siguientes:

- **XLS** (eXtensible Style Language): permite definir hojas de estilo para XML e incluye capacidad de transformación de documentos.
- **XML Linking Language**: determina aspectos sobre los enlaces entre documentos XML e incluye **Xpath**, **Xlink** y **Xpointer**.
- **XML Namespaces**: proveen de un contexto donde se aplican las marcas del documento XML y que se diferencian de otras con el mismo nombre válidas en otros contextos.
- **XML Schemas**: permiten definir restricciones que se aplicarán a un documento XML. Actualmente las mas utilizadas son **DTD**.

```
<?xml version="1.0" encoding="UTF-8x°x"?>
<!DOCTYPE biblioteca>

<biblioteca>
    <ejemplar tipo="libro" isbn="978-2-7460-4958-1" edicion="1">
        <titulo>XML practico</titulo>
        <editorial>Ediciones Eni</editorial>
        <autor>Sebastien Lecomte</autor>
        <autor>Thierry Boulanger</autor>
        <autor funcion="traductor">Ángel Belinchon Calleja</autor>
        <prestamos>
            <lector inicio="13/05/2014" devolucion="15/05/2014">Pedro López</lector>
            <lector inicio="13/07/2015" devolucion="15/07/2015">Ali Méndez</lector>
        </prestamos>
     </ejemplar>
</biblioteca>
```

Figura 1.2.3: Documento XML simple

1.2.4 Comparación XML y SGML

Aunque XML esta basado en SGML, estos tienen muchas diferencias. A continuación se muestra una tabla con las principales diferencias de estos dos lenguajes de marcas.

XML	SGML
Su uso es sencillo	Su uso es my complejo
Trabaja con documentos bien formados	Solo Trabaja con documentos válidos
Desarrollo de aplicaciones a bajo coste	Aplicaciones para procesarlo costosas
Muy utilizado en informática y otras áreas	Se utiliza en sectores muy específicos
Compatibilidad e integración con HTML	No hay compatibilidad con HTML
Formateo y estilos fáciles de aplicar	Formateo y estilo relativamente complejos

Como vemos en esta tabla, SGML es un un lenguaje mas complejo y costoso que XML, además de imponer mas restricciones, haciéndolo un lenguaje menos flexible que XML y mas orientado a sectores concretos. Para obtener más información sobre XML, podemos consultar el Estándar XML publicado por la W3C.

1.2.5 Comparación XML y HTML

Aunque tanto XML como HTML se crearon a partir de SGML y su sintaxis es similar, son lenguajes diferentes con propósitos diferentes, como podemos ver en la siguiente tabla.

XML	HTML
Es un perfil de SGML	Es una aplicación de SGML
Permite definir conjuntos de etiquetas	Aplica un conjunto limitado de etiquetas
Modelo de hiperenlaces complejo	Modelo de hiperenlaces simple
Navegador como plataforma de desarrollo	Navegador como visor de páginas
Compatibilidad e integración con HTML	No hay compatibilidad con HTML
Fin de las etiquetas propietarias	Problema de la no compatibilidad en navegadores

1.3 Etiquetas, Elementos y Atributos

Los lenguajes de marcas usan una serie de etiquetas intercaladas en un documento sin formato, las cuales serán posteriormente por el interprete del lenguaje.

Existen tres términos ampliamente utilizados en los lenguajes de marcas:

- **Etiquetas**: una etiqueta, también llamada **tag**, es un pequeño bloque de código que se escribe encerrado entre los símbolos **menor que** (**<**) y **mayor que** (**>**). Normalmente se utilizan **dos etiquetas**, una de **inicio** y otra de **fin**, para indicar que el efecto que queríamos conseguir ha finalizado, con la única diferencia que a la etiqueta de fin se le añade el carácter **/** antes del nombre.
- **Elemento**: representan estructuras mediante las que se organiza el contenido del documento, o acciones que se desencadenan cuando el navegador lo interpreta. Está **compuesto** de la **etiqueta de apertura**, la **etiqueta de cierre** y el **contenido entre ambas**.

- **Atributo**: es un par **nombre-valor**, que se encuentra al inicio de un elemento e indican diferentes propiedades asociadas a ese elemento

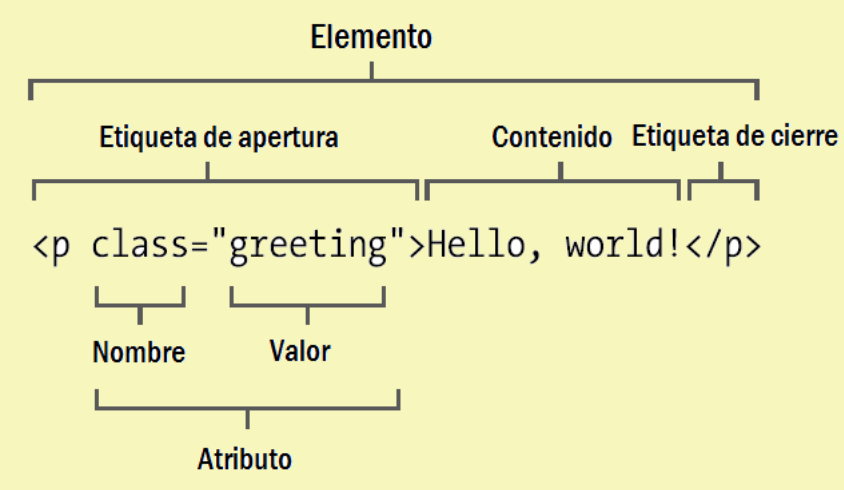

Figura 1.3.1: Partes de un elemento HTML

1.4 Herramientas de Edición

Para trabajar con XML es necesario, por un lado, editar los documentos, y por otro procesarlos. Por ello, necesitaremos dos tipos de herramientas para trabajar con él, estas son:

- **Editores XML**

  Una característica de los lenguajes de marcas es que se basan en la utilización de ficheros de **texto plano**, por lo que basta con usar cualquier editor de texto para construir un documento XML.

  Aunque podemos usar cualquier editor, cuando elaboramos documentos XML complejos es conveniente usar algún software de edición XML. Estos nos ayudan a crear estructuras y etiquetas de los elementos usados en XML, resaltan las etiquetas para diferenciarlas más cómodamente y ademas incluyen ayudas para la creación de otros elementos como DTD, hojas de estilo CSS o XLS,.. El W3C ha desarrollado un editor HTML, XHTML, CSS y XML gratuito llamado Amaya, pero existen otros muchos gratuitos como: Notepad++, VSCode, Sublime Text, Netbeans,..etc

- **Procesadores XML**

  Para interpretar un documento XML puede usarse cualquier navegador. Los procesadores XML permiten visualizar los documentos XML y acceder a su contenido y estructura. Un **procesador** es un conjunto de módulos de software entre los que se encuentra un **parser**, que comprueba que el documento cumple las normas establecidas para que pueda abrirse. Estás normas pueden corresponderse a las necesarias para trabajar con documentos de tipo válido o solo exigir que el documento este bien formado. A los primeros se le conocen como **procesadores validadores** y a los segundos como **procesadores no validadores**.

  Para publicar documentos XML en internet se usan **procesadores XSLT**, que permiten generar archivos HTML a partir de XML.

  Puesto que XML se usa para el intercambio de archivos entre aplicaciones, hay que recurrir a motores independientes como «XML para Java» de IBM, JAXP de Sun, etc

1.5 XML

XML, que significa *eXtensible Markup Language*, es un lenguaje que permite definir lenguajes de marcas desarrollado por el W3C y utilizado para almacenar datos de forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos. [3]

Su importancia radica en que permite **compartir datos** entre diferentes equipos y aplicaciones de forma **segura**, **fiable** y **sencilla**. El hecho de que diferentes equipos y aplicaciones puedan leer y generar archivos en este formato lo convierte en una herramienta muy útil para el envío de información a través de la Web. Aunque a veces suele confundirse con HTML, podemos decir que HTML esta diseñado para mostrar datos en nuestras pantallas mientras que XML esta diseñado para almacenar y compartir esos datos.

El XML ahorra tiempos de desarrollo y proporciona ventajas, dotando a webs y aplicaciones de un método muy potente para almacenar y compartir información. Por ello, se ha convertido en un formato universal usado por todo tipo de sistemas operativos y dispositivos.

1.5.1 Estructura y Sintaxis

Un documento XML es un documento de de texto, con la **extensión .xml**, compuesto de un **conjunto de etiquetas**, **estructuradas en árbol**, que describen la organización del documento y que es interpretado por un navegador Web.

La **características básicas** de XML son las siguientes:

- Es directamente **compatible** con protocolos usados en en la Web como **HTTP** y **URL**.
- Todo documento que verifique las reglas de XML esta **conforme con SGML**.
- **No se requieren conocimientos de programación** para realizar tareas sencillas en XML.
- Los documentos XML son **fáciles de crear**.
- **La difusión** de documentos XML **está asegurada**, ya que cualquier procesador de XML puede leer documentos XML.
- El marcado de XML es **legible para los humanos**.
- El diseño de XML es **formal y conciso**.
- XML es **extensible**, **adaptable** y **aplicable** a una gran variedad de situaciones.
- XML es **orientado a objetos**
- Todo documento XML se **compone** de **datos de marcado** y **datos carácter** entremezclados.

El **proceso de creación** de un documento pasa por varias etapas en las que el éxito de cada una de ellas se basa en la calidad de la anterior. Estas etapas son:

1. Especificación de requisitos.
2. Diseño de etiquetas.
3. Marcado de los documentos.

El **marcado** son etiquetas que se añaden para estructurar los documentos y permitir a los ordenadores interpretar los textos. Los **datos carácter** son los que componen la verdad información del documento.

Los documentos XML también pueden **contener comentarios**, que son interpretados por el intérprete XML y que comienzan con la cadena <!– y finalizan con –>. Pueden estar en cualquier posición del documento salvo en:

- El prólogo
- Dentro de una etiqueta

Los XML están **formados** por dos partes, **el prólogo** y **el ejemplar**, los cuales veremos a continuación.

1.5.2 El Prólogo

El prólogo es la primera parte que nos encontramos en cualquier documento XML y siempre debe preceder al ejemplar del documento. Éste se divide en dos partes, la **declaración XML** y la **declaración de la codificación** empleada, los cuales explicamos a continuación.

1. **La declaración XML**: es la primera linea del documento, de no ser así, se genera un error que impide que el documento sea procesado. En caso de que sea opcional, se permite el procesamiento de documentos HTML y SGML como si fueran documentos XML, si es obligatoria, estos deberán incluir la declaración de la versión XML. Esta declaración permite indicar de forma explicita que el documento es de tipo XML.

   El prologó puede tener **tres funciones**:

   a) **Declaración de la versión** utilizada de XML:

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

   b) **Declaración de la codificación empleada**:

   ```
   <?xml version= "1.0" encoding="iso-8859-1" ?>
   ```

   En este caso se usa el código código iso-8859-1 (Latin-1) que permite el uso de tildes o caracteres como la «ñ». Otro de los códigos a tener en cuenta es **UTF-8** (**unicode**). Esta codificación soporta más caracteres que iso-8859-1 y permite que estos se visualicen correctamente en más sistemas. Es la **codificación estándar** recomendada para todos los documentos y la usaremos siempre, salvo que por algún motivo no pueda ser empleada.

   Para consultar más información sobre codificación de caracteres en sistemas informáticos y cuales son los más empleados podemos visitar página de Wikipedia sobre codificación de caracteres.

   c) **Declaración de autonomía del documento**:

   Indica si el documento necesita de otro para su interpretación. Para declararlo, hay que definir el prólogo completo:

   ```
   <?xml version= "1.0" encoding="iso-8859-1" standalone="yes" ?>
   ```

   La opción *standalone* indica al procesador XML si un documento es independiente (*standalone=ɛes"*), o si depende de un documento externo, como declaraciones de marcas externas o una DTD externa (*standalone="no"*). Por defecto el documento se considera independiente.

2. **La declaración del tipo de documento**:

   Esta declaración define el tipo de documento que estamos creando para que sea procesado correctamente. Toda declaración de tipo de documento comienza con la cadena:

```
<!DOCTYPE Nombre_tipo ...>
```

1.5.3 El Ejemplar

Es la parte más importante de un documento XML ya que contiene los **datos reales** del documento. Es el **elemento raíz** de un documento XML y este se nombrará igual que la declaración del tipo de documento (*!DOCTYPE*). Suele estar compuesto de **elementos anidados**.

En el siguiente ejemplo, el ejemplar es el elemento <*libro*>, que a su vez esta compuesto de los elementos <*titulo*>, <*autoria*>, <*editorial*>, <*isbn*>, <*edicion*> y <*paginas*>.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE libro>

<libro>
  <titulo>XML práctico</titulo>
  <autoria>
      <autor>Sebastien Lecomte</autor>
      <autor>Thierry Boulanger</autor>
  </autoria>
  <editorial>Ediciones Eni</editorial>
  <isbn>978-2-7460-4958-1</isbn>
  <edicion>1</edicion>
  <paginas>347</paginas>
</libro>
```

Figura 1.5.1: Ejemplo del *ejemplar* en un documento XML

Es recomendable **establecer un criterio** y mantenerlo durante todo el documento, por ejemplo, que las etiquetas vayan escritas **siempre en minúsculas**.

Por otro lado, es conveniente anidar los elementos para una visualización óptima del documento, esto se hará **indentando** o **tabulando** el código. A continuación se muestran dos figuras, una sin indentación (errónea) y otra con indentación (correcta).

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE libro>

<libro>
<titulo>XML práctico</titulo>
<autoria>
<autor>Sebastien Lecomte</autor>
<autor>Thierry Boulanger</autor>
</autoria>
<editorial>Ediciones Eni</editorial>
<isbn>978-2-7460-4958-1</isbn>
</libro>
```

Figura 1.5.2: Código XML sin indentación (incorrecto)

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE libro>

<libro>
    <titulo>XML práctico</titulo>
    <autoria>
        <autor>Sebastien Lecomte</autor>
        <autor>Thierry Boulanger</autor>
    </autoria>
    <editorial>Ediciones Eni</editorial>
    <isbn>978-2-7460-4958-1</isbn>
</libro>
```

Figura 1.5.3: Código XML con indentación (correcto)

Como podemos observar, en la segunda figura se reconoce más fácilmente la estructura del documento y se facilita la lectura de este.

Por último, es recomendable **anidar grupos de datos relacionados**, como se ha hecho en el ejemplo anterior con los elementos <*autor*> dentro del elemento <*autoria*, ya que el documento que **mas limpio** y **ordenado**.

1.5.3.1 Elementos

Todos los datos de un documento XML deben **pertenecer** a un **elemento del mismo**.

Los elementos son los diferentes **bloques de información** que permiten definir la **estructura** del documento XML. Estos está delimitados por una **etiqueta de apertura** y una **etiqueta de cierre**. Pueden estar **compuestos** por **otros elementos**.

Los **nombres** de las etiquetas deben ser **autodescriptivos**, es decir, que ilustren su contenido. Por ejemplo, si estamos con datos relativo a un libro, una etiqueta no debería ser <*caracteristicas*>, ya que es demasiado ambiguo, deberíamos utilizar nombres como <*titulo*>, <*isbn*, etc...

La **formación de elementos** debe cumplir ciertas **normas** para que queden bien definidos y que el documento XML al que pertenecen pueda ser procesado sin generar ningún error fatal. Estas normas son las siguientes:

- En todo documento XML debe existir **un documento raíz** y **solo uno**.
- **Todos** los elementos tienen una **etiqueta de inicio** y **otra de cierre**. En caso de que en el documento existan **elementos vacíos**, se pueden sustituir las etiquetas de apertura y cierre por una de elemento vacío. Esta se construye como una etiqueta de inicio pero añadiendo **/**, es decir, <*elemento/*> en vez de <*elemento*>. Esta considerada una etiqueta de apertura y cierre.
- Al anidar hay que tener en cuenta que **no puede cerrarse** ningún **elemento** que **contenga otro elemento** que aún **no se haya cerrado**.
- Los **nombres** de las **etiquetas de apertura y cierre** deben ser **idénticos**, respetando las mayúsculas y minúsculas. Pueden ser cualquier cadena alfanumérica que no contenga espacios ni tildes, y que no comience por dos puntos (**:**), ni por la cadena **xml**.
- El contenido de los elementos **no puede contener** la cadena «***]]>***», por compatibilidad con SGML. Además, no se pueden utilizar los caracteres **mayor que** (**>**), **menor que** (**<**), **ampersand** (**&**), **dobles comillas** (**"**) y **apostrofe** (**‘**).

En caso de tener que utilizar alguno de estos caracteres, se deben sustituir por las siguientes cadenas:

Caracteres	Cadena
>	>
<	<<
&	&

Caracteres	Cadena
"	"
’	'

- Para **utilizar caracteres especiales**, como £, ©, ®,... hay que usar las expresiones ***&#D;*** o ***&#H;***, donde D y H se corresponden con el número decimal o hexadecimal, respectivamente, correspondiente al carácter que se quiere representar en código **UNICODE**.

  En los siguientes enlaces puedes consultar tanto los códigos ASCII y su equivalente en HTML, como el código correspondiente a cada carácter en UNICODE.

  - ASCII - https://www.asciitable.com/
  - UTF-8 - https://www.charset.org/utf-8

1.5.3.2 Atributos

Las etiquetas pueden tener **atributos**, que **permiten definir propiedades** a los elementos de un documento. Los atributos, a diferencia de los elementos, no puede organizarse en ninguna jerarquía o estructura de árbol, no pueden contener ningún otro elemento, no pueden contener valores múltiples y no reflejan ninguna estructura lógica. En definitiva, los atributos **no se podrán extender fácilmente** en futuros cambios.

Un elemento puede tener varios atributos, pero ninguno de estos puede estar vacío, además todos los atributos dentro de un elemento **deben ser únicos**. Los atributos se codifican de la siguiente forma:

```
<etiqueta atributo="valor_atributo"></etiqueta>
```

No debe usarse un atributo para almacenar información susceptible de ser dividida, sino para proporcionar información adicional sobre el elemento. A continuación vamos a ver un ejemplo de la utilización de atributos en un documento XML.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<centro_educativo>
    <alumno sexo="Varón" fecha_nacimiento="05/06/1990">
        <nombre>Pablo</nombre>
        <apellido>Pérez</apellido>
        <telefono tipo="Móvil">666666666</telefono>
        <direccion tipo="Calle">Policar, 34</direccion>
    </alumno>
 </centro_educativo>
```

Figura 1.5.4: Uso de atributos en documentos XML

Por norma general, intentaremos **evitar el uso** de atributos o procurar **no abusar de ellos**. Normalmente **los usaremos** para **metadatos** o información que no sea relevante para los datos. Se puede ver como una manera de incorporar características o propiedades a los elementos, como en el siguientes

ejemplo: *<perro raza="Pastor">Lolo</perro>*. También se suelen para especificar unidades de medida y similares: *altura unidad_altura="cm">178</altura>*. Hay que destacar que toda la información que se almacena en atributos se podría almacenar igualmente en elementos.

Lo que si es recomendable es que **una vez elegido un estilo**, mantenerlo dentro de todo el documento XML, teniendo en cuenta que los **nombres de los atributos** tienen que cumplir las **mismas normas** que los **elementos**, y no pueden contener el carácter menor que "<".

1.6 Documentos XML bien formados

Los documentos XML deben de estar **bien formados**, es decir, ser documentos **válidos**.

Los **documentos bien formados** son aquellos que cumplen las reglas sintácticas de creación de documentos XML ya mencionados en los puntos anteriores, como por ejemplo, que usan caracteres válidos para la creación de nombres de etiquetas o atributos, que las etiquetas estén cerradas correctamente, etc...

Los **documentos válidos** son aquellos que, ademas de estar bien formados, cumplen los requisitos de una definición de estructura (DTD, Schema,..), que veremos en la siguientes unidad.

Por lo tanto, para que un documento esté bien formado, debe **verificar** las **reglas sintácticas** que define la recomendación de la W3C para el **estándar XML**. Estas normas básicas se pueden resumir en las siguientes:

- El documento ha de tener **definido** una **declaración XML** en el prólogo:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
  ```

  Hay que tener en cuenta que aunque es posible omitir el prólogo, hay navegadores que nos pueden devolver un error al procesar el documento, por lo que siempre hay que incluirlo para evitar problemas. En caso de omitirlo, los valores por defecto son los mostrados encima de este párrafo.

- Existe un **único elemento raíz** por cada documento: es un solo elemento en el que todos los demás contenidos y elementos se encuentran anidados.

- Los elementos se organizan entre sí en un **estructura jerárquica** y **no se permite** el **solapamiento** de éstos.

- Hay que **cumplir reglas sintácticas** a la hora de definir el nombre de elementos y atributos. Estas normas se pueden resumir en:

  - El **nombre de elementos** puede contener como **primer carácter** los siguientes: *[A-Z]*, *[a-z]* y *"_"*. Para el resto de caracteres, ademas de estos, se pueden emplear: *[0-9]*, *"-"* y *"."*.

  - Las **etiquetas de apertura y cierre** deben ser **idénticas**, teniendo en cuenta que XML es sensible a mayúsculas y minúsculas.

  - Los **valores de los atributos** se escribirán siempre entre **comillas dobles o simples**. Si quisiéramos usar alguno de estos caracteres dentro del nombre usaríamos " (") y ' ('). También hay que tener en cuenta que hay nombres reservados para el uso del lenguaje.

  - **Los comentarios** en XML se escriben así: *<!– Comentario –>*

1.6.1 Espacios de Nombres

Los **espacios de nombres** permiten definir la pertenencia de los elementos y nombres a un contexto de vocabulario XML. De esta forma se resuelven las ambigüedades que se pueden producir al juntar dos documentos donde os autores han usado nombres similares para diferentes elementos.

Los espacios de nombres, también conocidos como **XML namespaces**, permiten dar nombre único a un elemento, indexándolo según un nombre de vocabulario adecuado. Además, están asociados a una **URI**.

En el documento, las etiquetas ambiguas se sustituyen por otras con el nombre precedido de un **prefijo**, que determina el contexto al que pertenece dicho nombre:

```
<prefijo:nombre_etiqueta></prefijo:nombre_etiqueta>
```

Esta etiqueta de denomina **nombre cualificado**. Al definir el prefijo hay que tener en cuenta que no se pueden usar espacios ni caracteres especiales y que no puede comenzar por un dígito.

Antes de poder usar el espacio de nombres hay que declararlo, es decir, asociar un índice con la URI asignada al espacio de nombres, mediante un atributo **xmlns**. Esto se hace entre el prólogo y el ejemplar y tiene la siguientes sintaxis:

```
<conexion>://<direccionservidor>/<apartado1>/<apartado2>/...
```

Si queremos consultar más información acerca de los espacios de nombres, podemos consultar el estándar en la recomendación en XML de la W3C.

1.7 Sistemas de Gestión Empresarial

Lo primero que debemos de entender antes de meternos de lleno en los sistemas de gestión empresarial es como fluye el flujo de información en una empresa, es decir, conocer sus recursos empresariales y gestionarlos eficientemente. Estos flujos de información son los siguientes:

- Entre los empleados de la empresa.
- Entre los empleados y la empresa.
- Entre la empresa con sus clientes y proveedores.

Estos **flujos de información** se puede clasificar en dos tipos:

- Informales y no estructurados.
- Formales y estructurados, que se centran en información de procesos críticos de la empresa.

Para facilitar estos flujos de información es conveniente tener instalado un **Sistema de Información**, que facilite el conocimiento propio de la empresa para mejorar la **planificación**, la **gestión** y el **control**.

Un **Sistema de Información** se define como un conjunto organizado, de elementos relacionados, orientados al tratamiento y administración de la información. Esta compuesto por elementos físicos, humanos y por un conjunto de normas y protocolos.

La utilización de sistema de información ofrece **ventajas competitivas** a las empresas, mejorando su eficiencia, calidad del producto o mejorando los servicios ofrecidos a los clientes. También ayudan, entre otras cosas, a la captación de clientes.

La automatización de los flujos de información cambió sustancialmente con el surgimiento de los **ERP** y los **CRM**, en los que se integran las diferentes aplicaciones que soportan los procesos de la empresa.

1.7.1 ERP

En una empresa, salvo que esta sea muy pequeña, es necesario usar algún sistema automatizado de gestión para controlar todos los recursos empresariales y procesos. Aquí es donde entran en juego los ERP.

Un **ERP** es un **sistema de gestión de la información** que se caracteriza por ser **una aplicación** donde hay **varias partes integradas** y se **especializa en manejar** todos los **datos relevantes** para la continuidad de la empresa. Estas partes gestionan diferentes procesos, por ejemplo producción, ventas, compras, pedidos, nóminas, etc...

Un **CRM** es un tipo de ERP que se centra en la **relación con los clientes** que tiene una empresa, es decir, información de contacto orientada a ventas.

Los **objetivos principales** de un ERP son los siguientes:

- **Optimizar** los procesos empresariales.
- **Acceder** a la información de forma confiables y precisa.
- **Permitir** compartir información entre los componentes de la organización.
- **Eliminar** los datos y operaciones innecesarias.

Las **características** que diferencia a los ERPs de otras aplicaciones, es que deben ser **sistemas integrales**, **modulares** y **adaptables**. Sus principales características son:

- Ser un programa con **acceso de una base de datos**.
- Sus **componentes interactúan** entre sí.
- Las **datos** deben ser **consistentes**, **completos**.

Suelen ser **sistemas complejos** de implantar ya que necesitan un desarrollo personalizado para las necesidades de la empresa a partir del paquete básico. Estas adaptaciones, suelen estar a cargo de **consultorías**.

Las **consultorías** en materia de **ERP** suelen ser de dos tipos:

- **Consultoría de negocios**: estudia los procesos de negocio de la compañía y personaliza el ERP para ajustarlo a las necesidades de la organización.
- **Consultoría técnica**: estudia los recursos tecnológicos existente.

En la actualidad, la mayoría de sistemas ERP poseen **interfaz web** que mejora la accesibilidad al sistema desde prácticamente cualquier lugar y dispositivo.

1.7.1.1 Características

Las características que distinguen a un sistema de gestión empresarial son las siguientes:

- **Integración**: un sistema ERP **integra todos los procesos de la empresa**, considerándolos como un serie de áreas que se relacionan entre sí para conseguir una mayor eficiencia, reduciendo tiempo y costes.

- **Modularidad**: **cada módulo** de un sistema ERP se corresponde con un **área funcional** de la empresa. Gracias a una **base de datos centralizada**, todos los módulos pueden compartir información entre sí, facilitando la adaptabilidad, personalización e integración. Cada módulo suele usar un software específico para su funcionalidad.
- **Adaptabilidad**: aunque las dos características anteriores facilitan la adaptabilidad del ERP a los requisitos de la empresa, a veces se utiliza una solución más genérica, para abaratar costes, y se modifican algunos procesos para adaptarlos al sistema ERP.

Gracias a la modularidad y la capacidad de integración de las funcionalidades, los ERPs se pueden adaptar fácilmente a las necesidades de cada empresa. Los diferentes módulos se interconectan entre sí de forma que una única herramienta ERP puede adaptarse a diferentes empresas cambiando el conjunto de módulos activos y sus relaciones.

Los **módulos principales** que forman un ERP son los siguientes:

- **Ventas/Marketing**: interfaz pública que interactúa con los clientes, pedidos, estrategias de ventas, precios, promociones, publicidad, etc..
- **Finanzas**: es la base de cada ERP, donde se almacenan las transacciones facilitando las auditorías.
- **Inventario/Logística**: controla el stock y los flujos de entrada y salida.
- **Recursos Humanos**: gestión de personal, nómina, productividad, incentivos, etc..
- **Producción**: en núcleo que se encarga de los movimientos físicos del producto, planificación de materiales, etc..

Estos módulos son considerados los básicos, pero se pueden añadir muchos más como proyectos, planificación de ventas, configuración de productos a medida, etc..

No todos los **trabajadores** accederán al ERP de la misma forma, ya que cada grupo tiene su **rol**, que será supervisado por un administrador, por lo que dependiendo de este rol el trabajador tendrá unos u otros módulos habilitados.

Mención especial requieren los **CRM** (Customer Relationship Management), que surgen como consecuencia de la aplicación especifica de los ERP a las interacciones con los clientes. Están centrados en mantener, crear y potencias las relaciones con clientes sirviendo de apoyo a las políticas de marketing de la empresa.

En la actualidad los sistemas globales de CRM se pueden dividir en:

- **Aplicaciones** electrónicas para los **canales de distribución** de la empresa.
- Centros de **atención telefónica**.
- **Autoservicio** para los clientes.
- **Gestión** electrónica de las **actividades** que afecten a los clientes.
- **Ventas**.

Entres sus **principales características** se encuentran la facilidad de **toma de decisiones** en **tiempo real**, **incremento de la rentabilidad** del clientes gracias a que se obtiene información muy útil a través de datos complejos, por ejemplo, identificando a los clientes que compran o que no están interesados y actuar en consecuencia.

1.7.1.2 Ventajas e Inconvenientes

Contar con un **sistema ERP personalizado**, permite a la empresa tener integradas diferentes utilidades que facilitan la gestión de la información. Aunque estos sistemas ofrecen muchas ventajas para una empresa, también tiene sus inconvenientes, como podemos ver en la siguiente lista.

- **Ventajas**
  - **Aumento** de la **información** que tiene la empresa sobre sus **potenciales clientes**. Los ERP que incluyen CRM aportan beneficios relacionados con la gestión de clientes de la empresa. Algunos incluyen control de calidad de los productos, permitiendo enfocar la oferta en las necesidades y deseos de los clientes, con la consecuente mejora de la satisfacción de estos.
  - **Aumento** de las **ventas**.
  - Permiten **resolver problemas** relacionados con el **tratamiento de la información** derivado del uso de sistemas anteriores.
  - **Aumenta** la **eficiencia operativa**.
  - **Facilitan** el **acceso de la información** y constituyen una mejora en las herramientas de tratamiento de la misma.
  - **Reducción** de **costes empresariales**, especialmente los relacionados con las operaciones de las tecnologías de la información y comunicaciones.
  - Permiten mayor **facilidad de configuración** de los sistemas de la empresa.
  - Mejoran el **entorno de integración** de todas sus acciones.
- **Inconvenientes**
  - El ERP ha de ser usado y realizado por **personal capacitado**.
  - La **instalación** del ERP es **muy costosa**.
  - Los ERP son vistos como **sistemas muy rígidos** y difíciles de adaptar al modo de trabajo de las empresas.
  - Son sistemas que sufren de problemas de **cuello de botella**, es decir, todos los usuarios se pueden ver afectados por la ineficiencia en uno de los departamentos.
  - Altos **coste de modificación** de un ERP ya implantado.

1.7.1.3 ERP de Software Libre

Dentro de los ERP de software libre encontramos una gran variedad de aplicaciones para la gestión empresarial. Entre ellas podemos destacar **Openbravo**, que es una iniciativa de origen español y **OpenERP**, actualmente conocido como **Odoo**, de origen belga y que se carateriza por tener una gran cantidad de módulos.

Openbravo es una aplicación de código abierto de planificación de recursos empresariales. Utiliza una arquitectura **cliente/servidor** web y esta escrita en **Java**. Se ejecuta sobre un servidor web y ofrece **soporte** para la diferentes bases de datos **Oracle** y **PostgreSQL**. Consta de dos versiones:

- **Openbravo Community Edition**: versión libre y gratuita desde la que no se puede acceder a los módulos comerciales. Tiene licencia **OBPL**.

- **Openbravo Network Edition**: versión bajo licencia **OBCL** que ofrece actualizaciones de código y si ofrece acceso a los módulos comerciales.

Además de estas dos versiones, Openbravo ofrece dos soluciones diferentes:

- **Suite de comercio Openbravo**: solución de comercio para minoristas.
- **Suite de negocio Openbravo**: solución global para empresas.

Por otro lado tenemos **Odoo** (anteriormente **OpenERP**) que resuelve problemas complejos haciendo uso de soluciones sencillas. Esta escrita en **Python** y hace uso de la base de datos **PostgreSQL**.

OpenERP fue creado en el año 2005 por un joven informático belga llamado **Fabien Pickaers**, causando gran sorpresa de que creara un programa de estas características y lo pusiera de forma gratuita en internet, mientras otras empresas venden sus productos a precios desorbitados. El modelo de Odoo esta basado en los servicios prestados en torno al software y tiene colaboradores alrededor de todo el mundo. En este enlace podemos ver el motivo del cambio de nombre de OpenERP a Odoo, así como la respuesta a algunas preguntas interesantes.

A continuación se muestran algunos enlaces de interese sobre estos ERPs:

- **Página oficial Openbravo** - `www.openbravo.com/es/`
- **Wiki sobre OpenBravo** - `http://wiki.openbravo.com/wiki/Main_Page`
- **Página oficial de Odoo** - `http://www.openerp.com/es`
- **Doc. de usuario de Odoo** - `https://www.odoo.com/documentation/8.0/`
- **Doc. ténica de Odoo** - `https://www.odoo.com/documentation/user/`

1.7.1.4 Instalación

Para realizar la instalación de paquete de gestión empresarial, primero tenemos que definir **cuales** son las **necesidades que debe cubrir** el software y buscar aquel que se ajuste mejor.

En general, las **tareas** implicadas en la **instalación e implantación** de un ERP son las siguientes:

- **Diseño de la instalación**: antes de iniciar la instalación deberá hacerse una análisis de cuales son las necesidades de la empresa y como las resuelve el ERP.
- **Instalación de equipos servidores y clientes**: será necesario revisar y actualizar el hardware de la empresa para que cumpla los requisitos el ERP.
- **Instalación del software**: instalación del ERP y del software que este necesite para su correcto funcionamiento.
- **Adaptación y migración del programa**: una vez instalado, será necesario configurarlo y adaptarlo a la empresa del cliente.
- **Migración de datos**: este proceso es de gran importancia ya que los datos son vitales para el correcto funcionamiento de la empresa. En ocasiones, si no hay forma de automatizar el proceso, deberá hacerse de forma manual.
- **Realización de pruebas**: la instalación del nuevo software puede conllevar un tiempo de transición en el que convivirá con el anterior software de gestión de la empresa. Durante este periodo de tiempo de deberán realizar pruebas para comprobar que el ERP funciona correctamente y los resultados son satisfactorios.

- **Documentación del sistema**: en esta fase se deben elaborar los manuales y documentos necesarios y ponerlos a disposición de la organización mediante los medios de difusión interna de los que disponga, como correo electrónico, tablón de anuncios, etc..
- **Formación de usuarios**: por último hay que formar a los usuarios en la utilización del ERP, empezando por los responsables de proyecto y siguiendo por los usuarios finales.

En la mayoría de los casos el sistema ERP correrán en una plataforma **cliente-servidor**, pero también pueden correr en un **servidor Web** o utilizar **tecnologías SaaS**.

Independientemente del sistema operativo que tengamos instalado en nuestra empresa o que decidamos usar para implementar nuestro ERP, tenemos que **tener en cuenta** lo siguientes:

- Tener un **servidor** donde instalaremos nuestros ERP.
- Instalar nuestra **base de datos** y conectarla con nuestro ERP.
- Instalar los **módulos necesarios ERP** que hayamos decidido adquirir.
- **Configurar** los diferentes **clientes** para que accedan al servidor y puedan realizar peticiones al ERP.

Hay que tener en cuenta que es **imprescindible** tener una **base de datos** instalada en nuestra empresa, ya que el sistema ERP se basa en la utilización de una base de datos para realizar los informes y las consultar. Así mismo, si decidimos incorporar el ERP a la **intranet** de la empresa será necesario disponer de un **servidor web** con soporte para el lenguaje de script en el que se haya programado la aplicación.

Los **tipos de instalación** de un sistema ERP/CRM dependerá de la plataforma con la que vayamos a trabajar con él. Los mas habituales son los siguientes:

- **Instalación en máquina virtual**: la aplicaciones y programas necesarios se proporcionarán en una máquina virtual lista para su utilización. Esta opción **no es apta** para entornos de **producción**, sino que se suele usar para probar el ERP.
- **Instalación de paquetes en entorno gráfico**: en este caso la instalación se hace mediante el entorno gráfico del sistema operativo, usando asistentes que instalan y resuelven las dependencias de los paquetes. Aunque esta opción se puede usar en producción. los **paquetes** pueden **no estar actualizados** a la última versión.
- **Instalación personalizada**: si queremos instalar la versión mas reciente de la aplicación podemos descargarnos los paquetes desde la pagina web que los contenga y instalarlos de forma manual mediante comandos. Esta opción permite un mayor control sobre los paquetes que se instalan aunque también es mas compleja de llevar a cabo.
- **No instalar y acceder a la aplicación online**: algunos ERP nos permiten usar el software sin necesidad de instalarlo, accediendo a ir mediante internet conectándonos a un servidor que tiene todos los datos y programas de aplicación. Esta es la opción utilizada por los proveedores de ERP que ofrecen el servicio SaaS.

En **Odoo** podemos realizar diferentes tipos de instalaciones, desde paquetes, donde no tendremos que hacer prácticamente nada, hasta instalaciones paso a paso donde tendremos que instalar y configurar cada componente. Desde la **versión 8.0**, el **servicio de acceso** de **cliente de escritorio** no existe, y su acceso se **realiza** a través de un **cliente web**. Esta es la forma de acceso a la mayoría de los ERP actualmente.

1.7.1.5 Personalización

Los sistemas ERP actuales permiten la incorporación de diferentes módulos predefinidos que facilitan la personalización de la aplicación.

Un **módulo** es una aplicación que se crea para realizar una determinada función. Existen unos **módulos básicos** que se pueden cargar durante la instalación y otros que pueden instalarse posteriormente. La **integración** de estos módulos pueden realizarse durante la instalación del sistema o posteriormente durante su **ampliación**.

La integración de estos módulos se puede realizar en el momento de la instalación o posteriormente. Éstos ofrecen una una gran variedad de recursos, como creación de informes avanzados, servicios de comunicación para plataformas móviles, etc.. Hoy en día la mayoría de ERP también incluyen un modulo CRM integrado.

Para instalar los módulos hay que acceder al ERP con un usuario que tenga permisos de administrador y usar el cargador de módulos que tienen estos paquetes.

Los **principales módulos** que nos podemos encontrar en un ERP son:

- **Gestión contable y financiera**: se encarga de recoger todas las operaciones contables de la compañía, centralizándolas para su consulta, publicación y control. Es módulo necesita estar integrado con un módulo de ventas para evitar duplicidades y poder acceder a los datos en tiempo real.
- **Compras, ventas y almacén**: el módulo de compras y ventas se encarga de registrar todas las operaciones de solicitudes de presupuestos, a proveedor, recepción de precios y creación de pedidos de compra. El módulo almacén permite gestionar las existencias de productos en almacén.
- **Facturación**: se encarga de generar todo tipo de información generado con la facturación de productos y servicios.
- **Gestión de Personal**: es módulo lleva a cabo la planificación y realización de las nominas, contratos, altas, bajas, control de horarios y datos del personal, además del sistema de pago a los empleados.
- **Gestión de relaciones con el cliente**: estos módulos, los **CRM**, permiten registrar todo lo relativo a la relación comercial con los clientes o posibles clientes.

En los siguientes enlaces, podemos consultar los diferentes módulos que tiene los dos ERPs que hemos tratado en esta unidad.

- Módulos de Odoo - https://openerpspain.com/funcionalidades-de-odoo/
- Módulos de Openbravo - https://www.openbravo.com/es/soluciones

1.7.1.6 Seguridad: Planificación, Usuarios y Roles

En primer lugar, para **aumentar la seguridad** de nuestro ERP, debemos analizar los tipos de riesgos a los que está sometido. Estos se pueden clasificar en **dos tipos**:

- **Riesgos Físicos**: cuando ocurre un fallo en un componente electrónico de nuestro sistema. Pueden fallar principalmente por agresiones externas, como altas temperaturas, incendios, inundaciones,etc...

- **Riesgos Lógicos**: sucede cuando no hay una política adecuada en los sistemas informáticos y se producen accesos no autorizados, bugs, errores en el sistema operativo o en el software, intrusiones externas, etc...

Para mitigar estos riesgos, los sistemas ERP implementan una serie de **medidas de seguridad** que se basan en los siguientes aspectos:

- **Niveles de acceso configurables para los usuarios según su rol**: en función de las tareas que deba realizar un usuario, éste debe contar con una serie de políticas que le permitan acceder a los datos que necesita, quedando otros datos reservados para usuarios con un nivel de toma de decisiones mas elevado.
- **Auditoría de cada transacción**: se controla cada envío de datos, lo que garantiza las operaciones realizadas.
- **Soporte para la conexión segura con https**: para garantizar la comunicación segura entre los clientes y el servidor se usa una conexión segura. Especialmente durante el proceso de autenticación de los usuarios.

Uno de los puntos más importantes es la **buena asignación de roles**, ya que esto permite que los usuarios solo puedan acceder a los datos que necesitan dejando, restringiendo su acceso a datos más sensibles solo disponibles a usuarios con mayores privilegios. Esta asignación se realiza mediante un **usuario administrador**.

Los paquetes básicos de ERP y módulos suelen tener varios **usuarios disponibles**:

- **Administrador**: es el usuario con **mayores privilegios**, tanto en el acceso a datos como en la gestión del sistema (creación de usuarios, instalación de módulos,...)
- **Usuario Normal**: usuario que no tiene privilegios de gestión pero si tiene acceso a **toda la información** almacenada.
- **Usuario Grupo**: se crean para recibir correo entrante para distribución.
- **Usuario de Portal**: permite a los usuarios **acceder a los portales** creados en el entorno pero **no a la aplicación**.

Hay aplicaciones en las que los usuarios no pueden borrarse directamente, sino que hay que hacerlo desde la base de datos.

Además de los diferentes usuarios, es necesario conocer las características de los **roles** que hay definidos para los usuarios de una aplicación. Estos roles definen **ciertos privilegios** a la hora de realizar **tares específicas**.

Las **características** de los **roles** son las siguientes:

- Un **rol** es un **grupo particular de privilegios**.
- El rol solo **tiene validez** si esta **asignado a un usuario**.
- Un **usuario** puede tener asignados **varios roles**, en este caso prevalece el más restrictivo.
- Los **cambios** realizados en los roles no son **efectivos** hasta que no se **inicia sesión** de nuevo.
- Si un rol **niega el acceso** a un módulo, se pierde la opción de ver cualquier subpanel perteneciente a éste.

1.8 Enlaces de Interés

En esta última sección solo vamos a incluir unos enlaces para completar y aclarar algunos de los conceptos que hemos visto en esta unidad, así como para expandir un poco el contenido.

- Introducción a XML - https://youtu.be/bCd2xaQrTAo
- Mi primer documento XML - https://youtu.be/LljEw3Std1Y
- Documentos XML bien formados - https://youtu.be/s1RgyhO9F2I
- Introducción a XML Namespaces - https://youtu.be/YGFcD3-WO6c
- Resumen teórico y práctico . https://youtu.be/YGFcD3-WO6c

Tema 2

Definición de Esquemas y Vocabulario XML

En este tema vamos a estudiar como establecer y definir la estructura de un documento XML. Para ello, vamos a emplear dos herramientas como son **Document Type Definition** (**DTD**) y **XML SChema Definition** (**XSD**).

Tanto DTD como XSD nos permitirán definir restricciones tanto de tipos como de estructura para un documento XML. Cada uno tiene sus ventajas he inconvenientes, las cuales estudiaremos en los siguientes puntos. También veremos algunas herramientas que nos ayudarán tanto en la creación como en la validación de documentos XML, así como de los documentos escritos en DTD o XDS.

2.1 Documento XML: Estructura y Sintaxis

Hasta ahora solo hemos trabajado con documentos XML muy básicos, los cuales podemos considera incompletos ya que solo hemos declarado que tipo de documento vamos a definir, pero no que características tiene.

Si recordamos la estructura de un documento XML que vimos en el tema anterior, este se compone de las siguientes partes:

- **Prólogo**: informa al intérprete encargado de procesar el documento de todos aquellos datos que necesita para realizar su trabajo. Este consta de:
  - **Definición de XML**: aquí se indica la versión de XML que se utiliza, el código de los datos a procesar y la autonomía del documento. Este ultimo dato, hasta ahora, siempre ha tenido la opción "*yes*", ya que los documentos que hemos definido hasta ahora han sido independientes.
  - **Declaración del tipo de documento**: hasta ahora somo hemos dicho que esta parte se compone del nombre del ejemplar precedido de la declaración "*!DOCTYPE*" y separado de esta al menos por un espacio y seguido de ">".
- **Ejemplar**: contiene los datos del documento que se quiere procesar. Es el **elemento raíz** del documento y ha de **ser único**. Esta compuesto de elementos con una estructura de árbol en la que el elemento raíz es el ejemplar y la hojas los elementos terminales, es decir, aquellos que no contienen más elementos. Los elementos, a su vez, pueden estar compuestos por atributos.

Una vez que hemos recordado la estructura básica de un documento XML, vamos a profundizar y expandir las definiciones que tenemos sobre los elementos de esta estructura y a ampliar la opciones que nos proporcionan para personalizar un documento XML.

2.1.1 Declaración de Tipo de Documento

Ya habíamos visto que esta parte nos permite establecer restricciones sobre el documento, aunque no lo habíamos profundizado en las partes que lo forman. Estas partes, se componen de los siguientes elementos:

- **Declaración del tipo de documento**: comienza con el texto que indica el nombre del tipo, precedido por la cadena "**!DOCTYPE**" separado del nombre del tipo por al menos un espacio. El nombre del tipo debe ser idéntico al del ejemplar del documento XML en el que se ésta trabajando.
- **Definición del tipo de documento**: permite asociar al documento una definición de tipo **DTD**, la cual se encarga de definir las cualidades del tipo. Es decir, define los tipos de elementos, atributos y notaciones que se pueden usar en el documento, así como las restricciones del documento, valores por defecto, etc.

  Para formalizar todo esto, XML esta provisto de ciertas estructuras llamadas **declaraciones de marcado**, las cuales pueden ser internas o externas. En este último caso, deben declararse en un documento donde encontrar las declaraciones, además de indicar en la declaración de XML que el documento no es autónomo. Las diferencias entre este tipo de declaraciones de marcado dan lugar a dos subconjuntos, el **interno** y el **externo**. Durante el procesado de un documento XML, siempre se procesa primero el conjunto interno y después el externo, lo que permite sobrescribir declaraciones externas compartidas entre varios documentos y ajustar el DTD al documento específico.

  Las principales de características de los dos subconjuntos son las siguientes:

  - **Subconjunto interno**: contiene las declaraciones que pertenecen a un **único documento** y no es posible compartirlas. Se localizan dentro de unos corches que siguen a la declaración del tipo de documento.
  - **Subconjunto externo**: están localizadas en un documento con extensión **dtd** que puede situarse en el mismo directorio del documento XML. Habitualmente son declaraciones que pueden ser compartidas entre múltiples documentos XML que pertenecen al mismo tipo. En este caso, la declaración de documento autónomo ha de ser negativa, ya que es necesario el fichero con el subconjunto externo para su correcta interpretación. Esto implica que el procesado del documento se hará más lento, ya que antes de procesarlo el procesador debe obtener todas las entidades.

    Para declarar el documento externo, deberá realizarse una declaración explicita de subconjunto externo que podrá realizarse de las siguientes dos formas:

    - **<!DOCTYPE nombre_ejemplar "URI">**

      Se especifica una URI donde podrán localizarse las declaraciones.

    - **<!DOCTYPE nombre_ejemplar PUBLIC "id_publico" "URI">**

      En este caso, también se especifica un identificador, que puede ser utilizado por el procesador para generar una URI alternativo, posiblemente basado en alguna tabla. Como se puede observar, también es necesario incluir una URI.

Ya que hemos visto como declarar el tipo de documento, en el siguiente punto veremos con se declara la sintaxis de un documento XML.

2.1.2 Definición de Sintaxis de XML

En los documentos de lenguajes de marcas, la distribución de los elementos esta jerarquizada según un estructura de árbol, lo que implica que es posible anidarlos pero no entrelazarlos. En este caso, el orden es significativo, pero no es así para los atributos, cuyo orden no importa, aunque si cabe recordar que no puede haber dos atributos con el mismo nombre.

Sabemos que los atributos no pueden tener nodos que dependan de ellos, por lo tanto solo pueden corresponder con las hojas de la estructura de árbol que jerarquiza los datos. Pero no solo los atributos pueden ser hojas de esta estructura, sino que los elementos pueden serlo también.

Por lo tanto, ¿que criterios podemos utilizar para decidir si un dato debe ser representado como un elemento o un atributo? Bueno, aunque no siempre se respetan, podemos tener en cuenta los siguientes criterios:

- El **dato** será **un elemento** si cumple algunas de las siguientes condiciones:
  - Contiene otras subestructuras.
  - Es de tamaño considerable.
  - Su valor cambia frecuentemente.
  - Su valor va a ser mostrado a un usuario o aplicación.
- Un **dato** sera un **atributo**. si cumple:
  - El dato es de pequeño tamaño y su valor raramente cambia, aunque hay situaciones en las que este caso se puede representar como un elemento también.
  - El dato solo puede tener unos cuantos valores fijos.
  - El dato guía el procesamiento de XML pero no se muestra.

A la hora de decidir si un dato debe ser representado como atributo o elemento, por tanto, podemos comprobar si cumple unas u otras características, aunque como hemos comentado, esto no debe ser tomado como una "guía absoluta", ya que puede haber datos que aunque cumplan características de un atributo, pueden, o deben, ser representados como un elemento, y viceversa.

2.1.2.1 XML Namespace

Como sabemos, no puede haber atributos dentro de un mismo elemento con el mismo nombre, así como no podemos nombrar dos elementos diferentes con el mismo nombre. Para eso, XML nos proporciona los **namespaces** o **espacios de nombres**.

Un **espacio de nombre** se usan para proveer de nombres únicos a los elementos y atributos de XML, siendo su uso una recomendación de la W3C. [4] En esencia, son una URI que referencia a una definición de vocabulario y nos permite:

- Diferenciar entre los atributos y elementos de diferentes vocabularios y con diferente significado que comparten nombre.
- Agrupar todos los elementos y atributos de una aplicación XML para que el software pueda reconocerlos con facilidad.

Los espacios de nombres se pueden declaran usando el atributo **xmlns**, y se pueden definir de la siguiente forma.

- **xmlns:"URI_namespace"**

```
<nombre xmlns="https://educacionadistancia.es/EspacioNombres">Ejemplo</nombre>
```

- **xmlns:prefijo="URI_namespace"**

```
<EN:nombre xmlns:EN="https://educacionadistancia.es/EspacioNombres">Ejemplo</EN:nombre>
```

En el segundo ejemplo se usa un prefijo, que nos informa de cuál es el vocabulario a la que esta asociada la definición. En ambos casos, **URI_namespace** hace referencia al conjunto de vocabulario del espacio de nombres.

2.2 Definiciones de Tipo de Documento (DTD)

Una **definición de tipo de documento** o **DTD**, es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la definición de una estructura de datos, para usar un diseño común y mantener la consistencia entre los diferentes documentos que usen el mismo DTD. De esta forma, los documentos pueden ser validados, conociendo la estructura de los elementos y la descripción que trae consigo cada documento.

Así, dos o mas documentos que tengan el mismo DTD se construyen de forma similar, tienen el mismo tipo de etiquetas, en el lugar y orden que especifica el DTD.

En el tema anterior se explicó cuando un documento XML estaba bien formado o era correcto, pero un documento bien formado no es necesariamente un documento válido. Para que un documento XML sea válido tiene primero que estar bien formado, y después seguir las especificaciones dictadas por la DTD.

Las DTD **están formadas** por una **relación precisa** de **que elementos** pueden aparecer o no en el documento y **donde**, así como **el contenido** y los **atributos** de los mismos. Garantiza que los datos de un documento XML cumplen las restricciones que se le ha impuesto en el DTD, ya que éstas permiten:

- **Especificar** la **estructura** del documento.
- Reflejar una **restricción de integridad referencial** mínima utilizando ID e IDREF.
- Utilizar unos pequeños mecanismos de abstracción comparables a las macro, las **entidades**.
- Incluir **documentos externos**.

Aunque como vemos, DTD tiene muchas ventajas, también tiene sus **inconvenientes**, siendo los principales los siguientes:

- Su sintaxis **no es XML**.
- **No** soporta **espacio de nombres**.
- **No define tipos** para los datos, solo hay un tipo de elementos terminales, que son los datos textuales.

- **No permite** las **secuencias no ordenadas**.
- **No** es posible **formar claves** a partir de **varios atributos o elementos**.
- Una vez que se define un DTD, **no** es posible **añadir más vocabularios**.

A continuación se muestra un ejemplo de un documento XML que queremos validar, y cual sería el resultado de aplicar un DTD que lo valide.

```
<?xml version="1.0"?>
<pelicula>
<titulo>Titanic</titulo>
</pelicula>
```

Figura 2.2.1: Documento XML que queremos validar

Con "**!DOCTYPE pelicula /[**" comienza la DTD y finaliza con "**]>**". El nombre que aparece a continuación debe ser la raíz del documento, es decir, **pelicula**, que a su vez indica que la etiqueta **titulo** esta dentro de la etiqueta **pelicula**. En la siguiente figura vemos el DTD que deberemos crear para validar este documento.

```
<?xml version="1.0"?>
<!DOCTYPE pelicula [
<!ELEMENT pelicula (titulo)>
<!ELEMENT titulo (#PCDATA)>
]>

<pelicula>
<titulo>Titanic</titulo>
</pelicula>
```

Figura 2.2.2: DTD para validar el documento

2.2.1 Declaración de la DTD

Existen dos formas de definir la DTD que describirá la estructura de un documento XML. Se puede incluir dentro del mismo documento o incluirlo en un documento externo e indicar su ubicación.

- **DTD incrustada**: es posible incluir la DTD en el mismo documento, como hemos visto en el punto anterior. En la siguiente figura podemos ver un ejemplo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pelicula [
<!ELEMENT pelicula (titulo)>
<!ELEMENT titulo (#PCDATA)>
]>
<pelicula>
<titulo>Titanic</titulo>
</pelicula>
```

Figura 2.2.3: Declaración DTD incrustada

Cuando se declara de esta forma, se puede proporcionar una ayuda al analizador de XML, si a través de instrucciones de proceso presentes en el código, se indica que el documento es independiente y que todo lo que necesita está contenido en el mismo. Para ello, basta con añadir el atributo **standalone="yes"**, como puede verse a continuación:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

El valor por defecto del atributo standalone es "yes", por lo que no sería estrictamente necesario incluirlo en este caso.

- **DTD externa**: otra opción es separar la declaración DTD del documento XML, realizando ésta en un documento externo e indicando donde se puede encontrar este documento. En el siguiente ejemplo, cargamos un fichero externo, *cine.dtd* con la declaración DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> <!-- Enlace a la DTD -->
<!DOCTYPE pelicula SYSTEM "cine.dtd">
<pelicula>
<titulo>Titanic</titulo>
</pelicula>
```

Figura 2.2.4: Documento SGML simple

Aunque las dos formas de declarar un documento DTD son igual de válidas, cabe apuntar que realizar un **declaración externa** tiene ciertas **ventajas** respecto a hacerla incrustada. Algunas de estas ventajas son estas:

- Si la DTD que se va a incluir es compartida por muchos documentos XML, es preferible que se encuentre en un archivo independiente, ya que si hay que hacer algún cambio en el DTD, solo tendrá que hacerse en el archivo donde este declarado, y no en cada archivo XML.
- La DTD puede ubicarse en un servidor web, de forma que cualquier persona con acceso a internet puede validar el documento XML que está creando, lo que garantiza que todos los documentos creados usen la última versión de la DTD. Para declarar que la DTD se encuentra en una servidor web, se puede especificar de la siguiente forma:

```
 <!DOCTYPE cine SYSTEM "http://cine.com/filmoteca.dtd">
```

Aunque sería más correcto si el archivo se pusiera de forma pública. Como se muestra en el siguiente código.

```
<!DOCTYPE cine PUBLIC "filmoteca" "http://cine.com/filmoteca.dtd">
```

Siendo **"filmoteca"** el nombre de la DTD.

2.2.2 Tipos de Elementos Terminales

Los **tipos de elementos terminales** son aquellos elementos que dentro de la estructura de árbol de un documento XML serían representados por la hojas.

La declaración de tipos de elementos esta formada por la cadena "**<!ELEMENT**", separada por un espacio del nombre del elementos XML que se declara y seguido de la declaración de contenido de dicho elemento. En el caso de los elementos terminales, es decir, aquellos que no tienen más elementos anidados, esta declaración de contenido puede tomar los siguientes valores:

- **EMPTY**: indica que el elemento no es un contenedor, es decir, que el elemento está vacío y no puede tener contenido. Para definir un elemento de este tipo se usa la siguiente definición:

```
<!ELEMENT ejemplo EMPTY>
```

  - XML asociado **correcto**:

```
<ejemplo></ejemplo> ó <ejemplo />
```

  - XML asociado **incorrecto**:

```
<ejemplo>Esto es un ejemplo</ejemplo> ó <ejemplo><a></a></ejemplo>
```

- **(#PCDATA)**: indica que los datos son analizados en busca de etiquetas, resultando que el elemento no puede contener otros elementos, es decir, solo puede contener datos de tipo carácter, exceptuando **<**, **[**, **&**, **]**, **>**. El elemento también podrá estar vacío. Un elemento de este tipo tendrá un definición así:

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

  - XML asociado **correcto**:

```
<ejemplo>Esto es un ejemplo</ejemplo> ó <ejemplo />
```

  - XML asociado **incorrecto**:

```
<ejemplo><a></a></ejemplo>
```

- **ANY**: permite que el contenido de un elemento sea cualquier cosa, ya sea vacío, texto u otro elemento. **No** es **recomendable** es uso de este tipo de elemento. Para definir un elemento como ANY se usa la siguiente sentencia:

```
<!ELEMENT ejemplo ANY>
```

El problema con este tipo de dato es que no pone prácticamente restricciones al elemento, pudiendo este adoptar diferentes formas lo que hace que esta declaración de tipos sea muy poco específica. Por ejemplo, si tenemos en cuenta la siguiente declaración DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mascota[
  <!ELEMENT mascota ANY>
  <!ELEMENT nombre (#PCDATA)>
  <!ELEMENT tipo(#PCDATA)>
   <!ELEMENT raza(#PCDATA)>
]>
```

Todos los elementos que vemos a continuación, serían válidos:

```
<mascota>
<nombre>Coco</nombre> es mi mascota.
Es una <tipo>chinchilla</tipo> <raza>blanca</raza>.
</mascota>



<mascota>
Coco es mi mascota.
Es una <tipo>chinchilla</tipo> <raza>blanca</raza>.
</mascota>


 <mascota>
<nombre>Coco</nombre>.
</mascota>

<mascota/>
```

Por lo tanto, aunque su uso es correcto y hay determinadas situaciones en las que se debe usar, no se recomienda de forma general declarar elementos de este tipo.

2.2.3 Elementos No Terminales

Una vez que hemos visto como declarar los elementos terminales, es decir, la hojas del árbol de la estructura, ahora vamos a ver como declarar los elementos **no terminales**, es decir, **elementos formados** por **otros elementos**, lo que serían las **ramas** del árbol de la estructura de un documento XML.

Para definir estos elementos, debemos hacer referencia a las elementos que las componen, como vemos en este ejemplo:

```
<!ELEMENT A(B C)>
```

En este caso, se ha definido **un elemento A**, que esta **formado** por un **elemento B** seguido de un **elemento C**. Cuando un elemento aparece varias veces en un documento, también deberemos indicarlo. Para ello, se usan los siguientes operadores, que nos permiten definir la **cardinalidad** de un elemento:

- **Operador opción** (**?**): este operador indica que un elemento es opcional, por lo que podrá o no aparecer en la estructura del elemento:

```
<!ELEMENT telefono (trabajo? casa)>
```

- **Operador uno-o-más** (**+**): define un componente que aparece al menos una vez. En el siguiente ejemplo se define un elemento formado por el nombre de una provincia y otro grupo, que puede aparecer una o varias veces:

```
<!ELEMENT provincia (nombre, (cp, ciudad)+ )>
```

- **Operador cero-o-más** (*****): este operador permite definir un elemento que aparece cero, una o más veces. Siguiendo con el ejemplo anterior, en este caso el grupo (cp, ciudad) podría no aparecer, o hacerlo varias veces:

```
<!ELEMENT provincia (nombre, (cp, ciudad)* )>
```

- **Operador de elección** (|): cuando se utiliza para sustituir las comas en la declaración de grupos indica que para formar el documento XML hay que elegir entre los elementos separados por este operador. En el siguiente ejemplo, el elemento *provincia* estará formado por el elemento *nombre* y el elemento *cp* ó el elemento *ciudad*.

```
<!ELEMENT provincia (nombre, (cp | ciudad) )>
```

Ya hemos visto como se realizan las declaraciones tanto de elementos terminales como no terminales, el siguiente paso será ver como se declaran los atributos que tienen estos elementos.

2.2.4 Atributos de los Elementos

En esta sección vamos a ver como se declaran los atributos que lleva un elemento, independientemente de si es terminal o no terminal. Para ellos, se usa la cadena **<!ATTLIST** seguida del nombre del elemento asociado al atributo que se declara, luego el nombre del atributo seguido de su tipo y modificador.

```
<! ATTLIST nombre_elemento nombre_atributo tipo_del_atributo "Valor_por_defecto">
```

Este elemento puede usarse para declarar una lista de atributos asociados a un elemento, o repetirse el número de veces necesario para asociar a dicho elemento esa lista de atributos, pero individualmente. Si un elemento tiene más de una atributo se puede expresar de la siguiente forma:

```
<! ATTLIST nombre_elemento nombre_atributo1  tipo_del_atributo1  "Valor_por_defecto"
                           nombre_atributo2  tipo_del_atributo2  "Valor_por_defecto"
                           nombre_atributo3  tipo_del_atributo3  "Valor_por_defecto"
           ..................................................................>
```

Al igual que los elementos, no todos los atributos son del mismo tipo. Los tipos más destacados con los que podemos definir un atributo son los siguientes:

- **Enumeración**: con el tipo, el atributo solo podrá tomar uno de los valores especificados dentro del paréntesis, los cuales irán separados por el operador |. En el siguiente ejemplo, vemos que el atributo *dia_semana* solo puede tomar como valor unos de los siete días de la semana:

```
<!ATTLIST fecha dia_semana (lunes|martes|miércoles|jueves|viernes|sábado|domingo)>
```

- **CDATA**: este tipo se usa para especificar que un atributo es una cadena de texto. En el siguiente ejemplo se define el atributo *color* del elemento ejemplo como CDATA:

```
<!ATTLIST ejemplo color CDATA #REQUIRED>

<ejemplo color="verde" />
```

- **ID**: permite declarar que el valor del atributo debe ser único y no se puede repetir en otros elementos ni atributos. Hay que tener en cuenta que los números no son nombres válidos en XML, por tanto no son un identificado legal. Para resolverlo, suele incluirse un prefijo en los valores y separarlo con un guión o letra, como en el siguiente ejemplo:

```
<!ATTLIST libro codigo ID #REQUIRED>

<libro codigo="Q1">El Quijote</libro>
```

- **IDREF**: permite hacer referencia a un identificador. En esta caso, el valor del atributo ha de corresponder con el identificador de algún elemento del documento XML.
- **NMTOKEN**: permite definir que el valor de un atributo ha de ser solo una palabra compuesta por los caracteres permitidos en XML, es decir, letras, números y los caracteres: "**:**", "_", "**-**" y "**.**".

Además de estos tipos, debemos declarar si el valor de un **atributo** es **obligatorio** o no. También podemos indicar cual será el valor por defecto de un atributo. Para todo ésto, se pueden emplear las siguiente cadenas:

- **#IMPLIED**: indica que el atributo sobre el que se aplica es opcional.

```
<!ATTLIST ejemplo color CDATA #IMPLIED>

<ejemplo color="verde" /> ó <ejemplo color="" />
```

- **#REQUIRED**: indica que el atributo sobre el que se aplica es obligatorio.

```
<!ATTLIST ejemplo color CDATA #REQUIRED>
```

  - XML no válido:

```
    <ejemplo color="" />
```

  - XML válido:

```
    <ejemplo color="verde" />
```

- **#FIXED**: permite definir un valor fijo para un atributo independientemente de que ese atributo se defina explícitamente en una instancia del elemento del documento XML.

```
<!ATTLIST ejemplo color CDATA #FIXED "verde">
```

  - XML no válido:

```
      <ejemplo color="rojo" />
```

  - XML válido:

```
      <ejemplo color="verde" />
```

- **Literal**: asigna por defecto a un atributo el valor indicado entre comillas, aunque este atributo también podrá tomar otros valores.

```
<!ATTLIST ejemplo color (rojo|verde|amarillo) "verde">

<ejemplo color="verde" />
```

Ya conocemos como se declaran los elementos y atributos en un DTD. En la siguiente sección veremos que también podemos declarar entidades, que nos ayudarán a la hora de crear este tipo de documentos.

2.2.5 Entidades

Las entidades nos permite **declarar valores constantes** dentro de un documento XML. Cuando se emplean dentro de un documento XML se limitan por "**'&**" y "**;**". Por ejemplo: **&entidad**. El intérprete, al procesar el documento XML, sustituirá todas las apariciones de la entidad por el valor que se le haya asignado en el DTD. La entidades **no admiten recursividad**, es decir, una entidad no puede hacer referencia a sí misma.

Las entidades puede ser de dos tipos, **generales** y de **parámetro**.

- **Entidades Generales**: dentro de la entidades generales podemos encontrar las **internas** y las **externas**:
  - **Internas**: son las que se declaran en el DTD y existen algunas predefinidas que podemos ver en la siguiente tabla.

Entidad	Carácter
<	<
>	>
&guot;	"
'	'
&	&

Figura 2.2.5: Entidades predefinidas en XML

Además de las entidades predefinidas, podemos definir las nuestras propias. Para ello emplearemos la estructura "**<!ENTITY nombre_entidad "valor entidad">**, donde:

- **nombre_entidad** es el nombre que recibe la entidad.
- **"valor_entidad"** es el valor que toma dicha entidad.

Para hacer referencias a las entidades creadas usaremos **&nombre_entidad**.

En la siguiente figura se puede ver un ejemplo de la definición de entidades internas propias. En este caso se declaran las entidades **autor** y **editorial**, que posteriormente se usan en el texto dentro de los elementos del documento.

En la figura 2.2.7 podemos ver el resultado, y como el intérprete a sustituido cada ocurrencia de las entidades por sus valores en el momento de generar el documento.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libros [
<!ELEMENT libros (libro)+>
<!ELEMENT libro (#PCDATA)>

<!ENTITY autor "Miguel de Cervantes">
<!ENTITY editorial "Alfaguara">
]>


<libros>
<libro>Don Quijote de la Mancha fue escrito por &autor;</libro>
<libro>SIDI fue escrito por Arturo Pérez-Reverte y publicado por &editorial;</libro>
<libro>Tiempos recios fue escrito por Mario Vargas y publicado por &editorial;</libro>
</libros>
```

Figura 2.2.6: Declaración de entidades internas en DTD

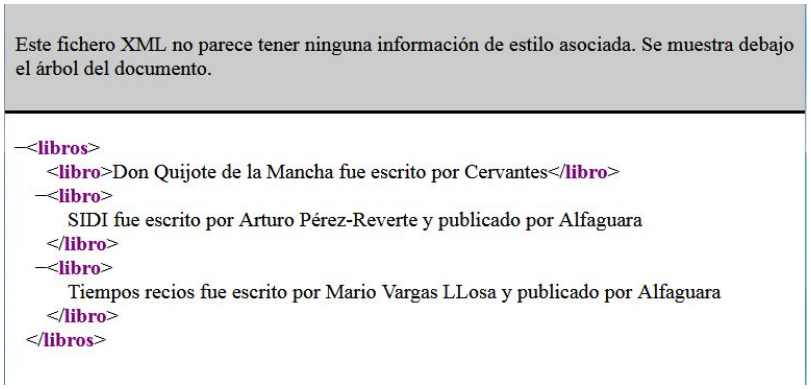

Figura 2.2.7: Documento generado con entidades internas sustituidas

- **Externas**: permiten establece una relación entre el documento XML y otro documento a través de la URL de este último.

```
<!ENTITY nombre_entidad SYSTEM "http://localhost/docsxml/fichero_entidad.xml">
```

En este caso el contenido de los ficheros es analizado por lo que deben seguir la sintaxis XML. Cuando es necesario incluir ficheros binarios, se utiliza la palabra reservada **NDATA** en la definición de la entidad, para que el fichero no sea analizado, y habrá que asociar a dicha entidad una declaración de notación , tal y como veremos en la siguiente sección.

En el caso de que la entidad externa vaya a ser utilizada por varias aplicaciones, deberemos declararla de la siguiente forma:

```
<!ENTITY nombre_entidad PUBLIC "identificador público formal" "camino hasta la DTD (uri)">
```

- **Entidades de Parámetro**: al igual que la generales, éstas pueden ser **internas** y **externas**.

- **Internas**: permiten dar nombre a partes de un DTD y hacer referencias a ellas a lo largo del mismo. Son especialmente útiles cuando varios elementos del DTD comparte listas de atributos o especificaciones de contenido. Se denotan por **%entidad**. Aquí mostramos un ejemplo de su uso.

```
<!ENTITY % direccion "calle, numero?, ciudad, cp">

<!ELEMENT almacen (%direccion;, web)>
<!ELEMENT oficina (%direccion;, movil)>
<!ELEMENT central (%direccion;, telefono)>
<!ELEMENT tienda (%direccion;, fax)>
```

- **Externas**: permite incluir en un DTD elementos externos, lo que se aplica al dividir la definición DTD en varios documentos.

```
<!ENTITY %persona SYSTEM "persona.dtd">
```

Como vemos, las entidades son bastante útiles a la hora de crear documentos DTD, y nos ayudan a establecer valores fijos para ciertos parámetro ó incluso a no tener que repetir código.

2.2.6 Declaración de Notación

En esta sección vamos a ver un tipo de declaración muy concreto que se aplica cuando queremos incluir un **fichero binario**. En este caso, deberemos indicar al intérprete que aplicación es la que se tiene que hacer cargo de procesar dicho fichero.

Para especificar la aplicación que deberá hacerse caso de ese fichero, usamos la siguiente **notación**:

```
<!NOTATION nombre SYSTEM aplicacion>
```

Por ejemplo, en el caso de que quisiéramos incluir un archivo de tipo **gif**, donde indicáramos un editor que se encargar de visualizar este tipo de imagen, lo haríamos de la siguiente forma:

```
<!NOTATION gif SYSTEM "gifEditor.exe">
```

En el caso de que quisiéramos asociar una entidad externa no analiza, bastaría con declarar dicha asociación de la siguiente manera:

```
<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>
```

Ya hemos visto todo lo relacionado con las declaraciones en DTD, por último, en el siguiente tema veremos las secciones condicionales y habremos concluido la parte dedicada a DTD.

2.2.7 Secciones Condicionales

Las **secciones condicionales** nos permiten incluir o ignorar partes de la declaración de un DTD. Para ellos se usan los dos siguientes tokens:

- **INCLUDE**: permite que esta parte de la declaración sea visible. Su sintaxis es:

```
<![INCLUDE [Declaraciones visibles] ] >
```

- **IGNORE**: permite ocultar esa sección de declaraciones dentro de DTD. Su sintaxis es:

```
<![IGNORE [Declaraciones visibles] ] >
```

El uso de la secciones condicionales suele **estar ligado** al uso de **entidades paramétricas**.

2.3 XML Schema

Los **DTD** nos permiten definir el **vocabulario** de un fichero XML, pero estos no permiten definir los **tipos** de datos que vamos a emplear en cada en elemento. Para ello tenemos **XML Schema**. Estos, a diferencia de DTD que usa una sintaxis similar a SGML, son documentos **XML** y también se especifican en ficheros de **texto plano** que se denominan **XSD** (XML Schema Definition).

Los elementos XML que se utilizan para generar un esquema han de pertenecer al espacio de nombres de XML Schema, que es: `https://www.w3.org/2001/XMLSchema`. En esta especificación se usa el prefijo **<xsd:schema>**, aunque para abreviar se suele utilizar **<xs:schema>**.

Las estructuras que se definen en XML Schema definen a su vez numeroso atributos para uso directo en cualquier documento. Estos se encuentra en un espacio de nombres diferente, en concreto en XML Schema Instance, que está definido en `https://www.w3.org/2001/XMLSchema-instance`. En esta especificación se usa el prefijo **<xsi:schema>**, aunque como en el anterior, para abreviar, se suele emplear **<xs:schema>**.

De esta manera, los prefijos **<xsd:schema>**, **<xsi:schema>** y **<xs:schema** se pueden usar indistintamente para definir el esquema, teniendo en cuenta que si se usa uno de los tres, deberemos usar este en todo el documento.

El ejemplar de estos ficheros es **<xs:schema**, que contiene declaraciones para todos los elementos y atributos que pueden aparecer en un documento XML asociado válido. Los elementos hijos de este ejemplar se denominan **<xs:element>**, y nos permiten crear un globalmente un elemento. Esto significa que el elemento creado puede ser el ejemplar del documento XML asociado.

El elemento **<xs:schema>** puede tener algunos atributos. Un ejemplo de declaración de este elemento sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8">

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="https://www.w3schools.com"
           xmlns="https://www.w3schools.com"
           elementFormDefault="qualified">
...

...

</xs:schema>
```

Figura 2.3.1: Declaración del ejemplar xs:schema

En esta declaración podemos ver varios elementos los cuales vamos e explicar a continuación.

- **xmlns:xs="http://www.w3.org/2001/XMLSchema"**

  Esta declaración indica que los elementos y tipos de datos usados en el esquema vienen del espacio de nombres "http://www.w3.org/2001/XMLSchema". También indica que los elementos y los tipos de datos que vengan de ese espacio de nombres tienen que tener el prefijo **xs:**. Este fragmento es el único **obligatorio** para que la definición sea correcta.

- **targetNamespace="https://www.w3schools.com"**

  Indica que los elementos definidos en el esquema pertenecen al espacio de nombres, es decir, el espacio de nombres de destino. En este caso es "https://www.w3schools.com".

- **xmlns="https://www.w3schools.com"**

  Indica cual es el espacio de nombres por defecto, en este caso "https://www.w3schools.com".

- **elementFormDefault="qualified"**

  Indica que cualquier elemento usado en una instancia XML que haya sido declarada con este esquema debe ser identificado con el espacio de nombres. Por defecto, toma este valor.

Aunque hay mas atributos que puede contener el elemento **xs:schema**, estos son algunos de los principales.

2.3.1 Tipos de Elementos en XML Schema

Los **elementos** se usan para especificar las etiquetas válidas de un documento XML. Todos los elementos que se vayan a utilizar en el ejemplar XML deben estar declarados en el esquema. Su etiqueta es **<xs:element** y su declaración de elementos en XML Schema tiene una estructura diferente dependiendo de si son **simples** o **complejos**. A saber:

- **Tipo simple**: no pueden contener otros elementos o atributos. Su estructura es la siguiente:

```
<xsd:element name="nombreElemento"
             ref="elementoReferenciado"
             type="tipoDato"
             minOccurs="valor"
             maxOccurs="valor"
             fixed="valor"
             default="valor"/>
```

Donde:

- **name**: es el nombre del elemento.
- **ref**: el elemento al que hace referencia esta declarado en otro lugar. No puede aparecer junto con **name**, ni si el elemento padre es **<xs:squema>**.
- **type**: el tipo de dato del elemento. No puede aparecer si usamos **ref**.
- **minOcurrs/maxOccurs** (opcionales): estos atributos indican el mínimo y máximo numero de ocurrencias del elemento respectivamente. El **valor por defecto** en ambos casos en **1**.

  Para especificar que el elemento puede aparecer un **número indeterminado** de veces el atributo **maxOccurs** toma el valor **"unbounded"**. Para especificar que el elemento **no puede aparecer**, el atributo **minOccurs** toma el valor **0**.

  Ninguno de los atributos puede aparecer si el padre del elemento es **<xs:schema>**.
- **fixed** (opcional): especifica un valor fijo para el elemento.
- **default** (opcional): especifica un valor por defecto para el elemento.

Podemos **crear nuevos elementos** "simpleType" a partir de uno ya existente, añadiendo condiciones a alguno de los tipos ya predefinidos en XML Schema. Para ello se utiliza el elemento **<xs:restriction**, como podemos ver en este ejemplo:

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

Figura 2.3.2: Uso del elemento xs:restriction

- **Tipo complejo**: estos elementos pueden estar compuestos por otros elementos y/o atributos. Sus elementos están definidos entre las etiquetas de inicio y final del elemento.
  - **<xs:schema>**: es el ejemplo básico de elemento compuesto y contiene la definición del esquema.
  - **<xs:complexType**: este elemento se usa para definir los elementos de tipo complejo entre su etiqueta de inicio y cierre. Pueden estar formados por subelementos predefinidos en XML Schema como:
    - **Secuencias** (**<xs:sequence**): permite construir elementos complejos mediante la enumeración de los elementos que lo forma en orden correcto. Si se altera dicho orden en el documento XML, éste no será valido.
    - **Alternativa** (**xs:choice**): representan alternativas, teniendo en cuenta que es la elección es exclusiva, es decir, especifica un lista concreta de elementos de los que solo puede aparecer uno.
    - **Secuencias no ordenadas** (**xs:all**): representa todos los elementos que conforman el elemento compuesto sin un orden específico. Dichos elementos podrán aparecer en cualquier orden dentro del documento XML.

- **Contenido mixto**: se define estableciendo en atributo **mixed** de un elemento a t**true**, es decir, **<xs:complexType mixed="true»**, y permite mezclar texto con elementos hijo. Los hijos se definen con las opciones anteriores, xs:sequence, xs:choice o xs:all.
- **Elemento vacío**: el elemento no puede contener texto ni otros subelementos, solo atributos. En el siguiente ejemplo vemos la definición de un elemento vacío y un XML válido para esa definición.

```
<xsd:element name="asignatura">
     <xsd:complexType>
         <xsd:attribute name='codigo' type='xs:integer'/>
     </xsd:complexType>
</xsd:element>

<!-- XML válido -->

<asignatura codigo='MAT-1920'/>
```

- **Referencias**: tal y como sucede en otro lenguajes, en un documento xsd podemos definir elementos de forma global y luego hacer referencias a estos desde otros elementos. Es es muy útil si a lo largo del documento se repiten determinados elementos. A continuación vemos un ejemplo del uso de referencias.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Direccion">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="via"/>
                <xsd:element ref="numero"/>
                <xsd:element ref="poblacion"/>
                <xsd:element ref="provincia"/>
                <xsd:element ref="cp"/>
            </xsd:sequence>
             </xsd:complexType>
      </xsd:element>

     <!-- Declaración de los elementos -->

      <xsd:element name="via" type="xsd:string"/>
      <xsd:element name="numero" type="xsd:integer"/>
      <xsd:element name="poblacion" type="xsd:string"/>
      <xsd:element name="provincia" type="xsd:string"/>
      <xsd:element name="cp" type="xsd:string"/>
 </xsd:schema>
```

En esta sección hemos visto como se declaran los elementos en un esquema XML Schema. Como vemos, aunque es más complejo que en DTD también nos aporta más potencia. En la siguiente sección veremos la declaración de atributos.

2.3.2 Atributos en XML Schema

El **elemento "atributo"** permite definir los atributos de los elementos en el documento xsd para usarlos adecuadamente en el documento XML. Estos elementos solo pueden aparecer en los elementos de tipo compuesto y su declaración debe realizarse siempre al final de la definición del elemento del que es atributo, es decir, justo antes del cierre **</xs:compleType>**. A continuación vemos un ejemplo:

```
<xsd:attribute name="nombreAtributo"
               ref="atributoReferenciado"
               type="tipoAtributo"
               use="valor"
               fixed=”valor"
               default=”valor”/>
```

Figura 2.3.3: Definición del elemento atributo en

Los diferentes atributos de este elemento son los siguientes:

- **name**: indica el nombre del atributo.
- **ref**: el atributo referenciado se encuentra definido en otra parte del esquema. No puede aparecer al mismo tiempo que **name**.
- **type**: indica el tipo del atributo. Tampoco puede aparecer al mismo tiempo que **ref**.
- **use** (opcional): indica si la aparición del atributo es opcional (**optional**), obligatoria (**required**) o prohibida (**prohibited**). Por **defecto**, toma el valor **“optional”**.
- **default** (opcional): valor que tomará el atributo por defecto al ser procesado si en el documento XML no se le asigna ningún valor. Solo se puede usar con tipos de datos **cadena de caracteres**. No puede aparecer si el atributo **fixed** esta presente.
- **fixed**: valor fijo que toma que el atributo. No puede aparecer si esta presente el atributo **default**.

Cabe mencionar que podemos usar el elemento **xs:attributeGroup** para agrupar atributos, lo que nos facilitará añadirlos después como un grupo a algún tipo de datos complejo.

2.3.3 Tipos de Datos

En esta sección vamos a ver los **tipos de datos**, que son los valores que puede tomar el atributo **type** cuando declaramos un atributo o un elemento y que determina el tipo de dato que tendrá el elemento o atributo asociado.

Los principales tipos de datos que tenemos en XML Schema son los siguientes:

- **String** (*xs:string*): se corresponde con una cadena de caracteres UNICODE. Puede incluir caracteres, saltos de línea y tabulaciones.

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

<poblacion>La Puebla de Vícar</poblacion>
```

- **Boolean** (*xs:boolean*): representa valores lógicos, pudiendo tomar por tanto solo los valores **true** o **false**.

```
<xs:attibute name="cancelado" type="xs:boolean"/>

<vuelo cancelado="true">LK345</vuelo>
```

- **Integer** (*xs:integer*): permite representar un número entero positivo o negativo.

```
<xs:element name="precio" type="xs:integer"/>

<precio>94</precio>
```

- **Positive Integer** (*xs:positiveInteger*): se usa para representar un entero positivo.
- **Negative Integer** (*xs:negativeInteger*): se usa para representar una entero negativo.
- **Decimal** (*xs:decimal*): para representar un subconjunto de los números reales que pueden ser representados por un número decimal, por ejemplo, 8,79.

```
<xs:element name="precio" type="xs:decimal"/>

<precio>8,97</precio>
<precio>8</precio>
```

- **DateTime** (*xs:dateTime*): se emplea para representar una fecha y horas absolutas. Tiene el formato "**YYYY-MM-DDThh:mm:ss**" y sólo es válido si se especifican todos sus componentes.

```
<xs:element name="fecha" type="xs:dateTime"/>

<fecha>2020-05-20T08:20:00</fecha>
```

- **Duration** (*xs:duration*): representa una duración de tiempo expresada en meses, años, días, horas, minutos y segundos. El formato utilizado es "**PnYnMnDTnHnMnS**". Para indicar una duración negativa se pone el signo negativo (-) precediendo a la P.

```
<xs:element name="periodo" type="xs:duration"/>

<!-- Duración de 2 años, 4 meses, 3 días, 5 horas, 6 minutos y 7 segundos -->
<periodo>P2Y4M3DT5H6M7S</periodo>

<!-- Se pueden omitir los valores nulos, luego una duración de 2 años será -->
<periodo>P2Y</periodo>
```

- **Time** (*xs:time*):permite representar la hora con el formato "**hh:mm:ss**".
- **Date**: (*xs:date*): permite representar una fecha con el formato "**YYYY-MM-DD**".
- **gYearMonth** (**xs:gYearMonth**): representa un mes de un año mediante el formato "**YYYY-MM**.

```
<xs:element name="fecha" type="xs:gYearMonth"/>

<fecha>2020-05</fecha>  Mayo de 2020
```

- **gYear** (*xs:gYear*): permite representar un año gregoriano usando el formato "**YYYY**".

- **gMonthDay** (*xs:gMonthDay*): permite representar un día de un mes mediante el formato "**–MM-DD**".

```
<xs:element name="fecha" type="xs:gMonthDay"/>

<!-- XML válido  -->
<fecha>--05-19</fecha> 19 de  Mayo

<!-- XML no válido -->
<fecha>05-19</fecha>

fecha>--05-32</fecha>
```

- **gDay** (*xs:gDay*): representa el **ordinal** de un día del mes mediante el formato "**–DD**", por ejemplo, el dia 4º del mes sería **–04**.
- **gMonth** (*xs:gMonth*): representa el **ordinal** del mes mediante el formato "**–MM**". Como en el ejemplo anterior, el mes 4º sería **–MM**.
- **anyURI** (*xs:anyURI*): representa una URI.

```
<xs:element name="web" type="xs:anyURI"/>

<!-- XML válido  -->
<web>www.iesaguadulce.es</web>

<web>www.iesaguadulce.es#texto</web>

<!-- XML no válido -->
<web>www.iesaguadulce.es#texto#texto1</web>
```

- **Language** (*xs:language*): representa los identificadores de un lenguaje, tal y como están definidos en el RFC 1766.
- **ID** (*xs:ID*): permite declarar que el valor del atributo debe ser único y no puede repetirse en otros elementos. Al igual que con los DTD, el valor no puede empezar por un número, sino por un carácter.
- **IDREF** (*xs:IDREF*): permite hacer referencia a un ID de otro elemento.
- **ENTITY** (*xs:ENTITY*): permite representar una entidad, las cuales vimos en el apartado 2.2.5.
- **NOTATION** (*xs:NOTATION*): permite representar una notación, tal y como vimos en el apartado 2.2.6.
- **NMTOKEN** (*xs:NMTOKEN*): representa que el valor es una cadena compuesta solo por los valores permitidos en XML.

Además de estos tipos, hay otros tanto primitivos como derivados que acepta XML Schema, si queremos ampliar la información, podemos consultar podemos consultar la Recomendación de la W3C sobre los tipos de datos de XML Schema.

2.3.4 Facetas de los Tipos de Datos

Las **facetas** nos permiten aplicar restricciones sobre los tipos de datos. Estás solo pueden aplicarse sobre tipos de datos simples utilizando el elemento **xs:restriction**, que tiene el atributo **base** en el que

se indica el tipo de datos sobre el que se quiere realizar la restricción.

Las facetas se expresan como un elemento dentro de una restricción y se pueden combinar para restringir más el valor del elemento. Entre otras, nos podemos encontrar las siguientes:

- **enumeration**: restringe a un determinado número de valores.

```
<xs:simpleType name="estado">
    <xs:restriction base="xs:string">
        <xs:enumeration value="conectado"/>
        <xs:enumeration value="ocupado"/>
    </xs:restriction>
<xs:simpleType>
```

- **length, minlength, maxlentgh**: restringen la longitud del tipo de datos

```
<xs:simpleType name="estado">
    <xs:restriction base="xs:string">
        <xs:maxLength value="9"/>
    </xs:restriction>
<xs:simpleType>
```

- **whitespace**: define el tratamiento de los espacios en blanco. Sus opciones pueden ser: **preserver, replace** o **collapse**.

```
<xs:simpleType name="nombre">
    <xs:restriction base="xs:string">
        <xs:whitespace value="preserve"/>
    </xs:restriction>
<xs:simpleType
```

- (**maxInclusive/maxExclusive**)(**minInclusive/maxInclusive**): límites superiores e inferiores del tipo de dato. Cuando son **Inclusive** el valor que se determina es parte del conjunto de valores válidos para el dato, mientras que si son **Exclusive**, el valor no pertenece al conjunto de valores válidos.

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

- **totalDigits**, **fractionDigits**: número de dígitos totales y decimales de un número decimal. Si lo combinamos con el minInclusive, maxInclusive, etc.., podemos aplicar bastantes restricciones a los números de tipo decimal.

```
<xs:simpleType name="calificaciones">
    <xs:restriction base="xs:integer">
        <xs:totalDigits value="2"/>
        <xs:minExclusive value="0"/>
        <xs:maxInclusive value="10"/>
    </xs:restriction>
</xs:simpleType>
```

- **pattern**: permiten construir patrones que han de cumplir los datos de un elemento, mediante el uso de **expresiones regulares**. En la siguiente tabla se muestran algunos de los patrones que podemos emplear a la hora de construir las expresiones regulares:

Patrón	Significado
[A-Z a-z]	letra
[A-Z]	letra mayúscula
[a-z]	letra minúscula
[0-9]	dígitos decimales
\D	cualquier carácter excepto dígitos
(A)	cadena que coincide con A
A | B	cadena que coincide con A o con B
AB	concatenación de las cadenas A y B
A?	cero o una vez la cadena A
A+	una o más veces la cadena A
A*	cero o más veces la cadena A
[abcd]	alguno de los caracteres entre corchetes
[âbcd]	ninguno de los caracteres que esta entre corchetes

Figura 2.3.4: Patrones para expresiones regulares

Las expresiones regulares son una herramienta muy potente y nos ayudan a restringir de forma muy específica la cadena de texto que permitimos en lo datos. Si queremos obtener más información podemos consultar la página de IBM sobre expresiones regulares, aunque realizando una búsqueda en internet podremos encontrar cientos de documentos sobre el tema, desde los más básicos a textos más avanzados.

A continuación se muestra un ejemplo del uso de expresiones regulares, en el que se establece un patrón que restringe el formato al de un DNI:

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

2.3.5 Extensión de Datos Simples

Como hemos comentado en el punto 2.3.1, XML Schema permite trabajar tanto con **datos simples** como con **datos complejos**, es decir, compuesto por el anidamientos de otros datos simples o compuestos.

Aunque ya hemos visto como se define un datos simple, en este apartado vamos a ver las tres diferentes maneras que existen de extender un tipo de dato simple:

- **Restricción**: se hace una restricción sobre un tipo de dato **XSD** ya definido y se establece el rango de valores que puede tomar. Las restricciones, como hemos visto en el punto anterior, son conocidas como **facetas**.

```
<xs:simpleType name="edad">
    <xs:restriction base="xsd:positiveInteger">
        <xs:maxExclusive value="19"/>
        <xs:minInclusive value="12"/>
    </xs:restriction>
</xs:simpleType >
```

- **Unión**: consiste en combinar dos o más tipos de datos en uno único.

```
<xs:simpleType name="LongitudInternacional">
    <xs:restriction base="xs:string">
        <xs:enumeration value="cm" />
        <xs:enumeration value="m" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LongitudSajona">
    <xs:restriction base="xs:string">
        <xs:enumeration value="pulgada" />
        <xs:enumeration value="pie" />
        </xs:restriction>
</xs:simpleType>

<!-- Unimos los dos tipos -->

<xs:simpleType name="Longitud">
    <xs:union memberTypes="LongitudInternacional  LongitudSajona">
</xs:simpleType>
```

- **Lista**: permite asignar a un elemento un número de valores válidos separados por espacios en blanco. Puede ser creada de forma similar a la unión con la diferencia de que solo puede contener un tipo d elemento.

```
<xs:simpleType name="LongitudSajona">
    <xs:restriction base="xs:string">
        <xs:enumeration value="pulgada" />
        <xs:enumeration value="pie" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Longitud">
    <xs:list itemType="LongitudSajona">
</xs:simpleType>
```

2.3.6 Defnición de Datos Complejos

Aunuqe en el punto 2.3.1 hemos visto los tipos de datos complejos, en esta sección vamos a verlos con mas detalle. Si recordamos la definición que hemos dado, los **datos de tipo complejo** son aquellos que están compuestos por otros elementos y/o atributos. Su contenido esta definido entre las etiquetas de apertura y cierre del elemento.

Dentro de los tipos de datos complejos nos podemos encontrar, entre otros, los siguientes:

- **xs:eschema**: contiene la definición del esquema, es el elemento de tipo complejo básico. Contiene el atributo **xmlns** (XML Namspace), que indica el espacio de nombres usado para el esquema.

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

- **xs:complexType**: entre sus etiquetas de apertura y cierre se definen los elementos de un dato complejo. Pueden estar formados por elementos predefinidos en XML Schema, como los siguientes:
  - **xs:sequence**: permite construir elementos complejos mediante la enumeración de los elementos que los forman en un orden concreto. Si se altera dicho orden en el documento XML, éste no será correcto.

```
<xs:element name="Direccion">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="calle" type="xs:string" />
            <xs:element name="poblacion" type="xs:string" />
            <xs:element name="provinvia" type="xs:string" />
            <xs:element name="codigo_postal" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>

<!-- XML Válido -->

<Direccion>
    <calle>Lago de Enol, nº 32</calle>
    <poblacion>Aguadulce</poblacion>
    <provincia>Almería</provincia>
    <codigo_postal>04720</codigo_postal>
</Direccion>
```

  - **xs:choice**: representa una alternativa, teniendo en cuenta que es exclusiva, es decir, especifica un lista de elementos de los cuales solo puede aparecer uno:

```
<xs:element name="FormaPago">
  <xs:complexType>
      <xs:choice>
          <xs:element name="paypal" type="xs:string" />
          <xs:element name="transferencia" type="xs:string" />
      </xs:choice>
  </xs:complexType>
</xs:element>
```

- **x:all**: representa a todos los elementos que componen el elemento de tipo compuesto en cualquier orden. A diferencia de **xs:sequence**, los elementos dentro de este tipo pueden aparecer en cualquier orden y el documento XML seguirá siendo válido.
- **Contenido Mixto**: se hace dando valor **true** al atributo **mixed** del elemento **xs:complexType** y se usa para poder mezclar texto y otros elementos como hijos. Los elementos hijos se definen con las opciones anteriores, xs:sequence, xs:choice y xs:all.

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

- **Elemento Vacío**: define un elemento que no puede contener ni texto ni hijos, únicamente atributos.

```
<xs:element>
    <xs:complexType>
        <xs:attribute name='codigo' type='xs:integer'/>
    </xs:complexType>
</xs:element>
```

- **Elemento simple con atributo**: es un elemento simple porque no tiene otros elementos, solo datos, pero es de tipo complejo (complexType) porque tiene un atributo:

```
<xs:element name="cuota">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:float">
                <xs:attribute type="xs:string" name="moneda"/>
            </xs:extension>
        </xs:simpleContent>
     5</xs:complexType>
</xs:element>
```

Con esta sección terminamos de ver como se definen en XML Schema los tipos de datos simples y complejos, así como los atributos. Además hemos visto formas de expandir los datos simples y agrupar los datos complejos y los atributos. En la siguiente sección veremos como enlazar un esquema con el documento XML que queremos validar.

Bibliografía

[1] Wikipedia - Tex. ttps://es.wikipedia.org/wiki/TeX.

[2] Wikipedia - ASCII. https://es.wikipedia.org/wiki/ASCII.

[3] Wikipedia - XML. https://es.wikipedia.org/wiki/XML.

[4] Wikipedia - XML_namespace. https://es.wikipedia.org/wiki/XML_namespace.