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

22 de abril 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

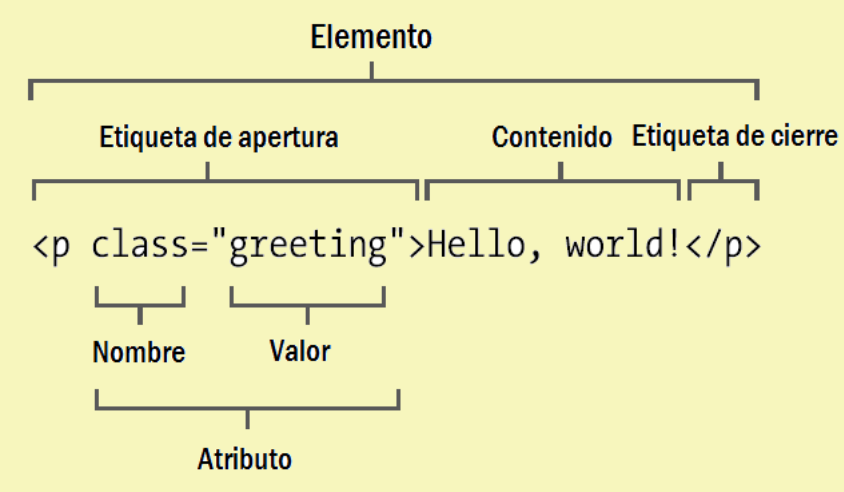


Figura 1.3.1.: Partes de un elemento HTML

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 esto permite al autor definir **restricciones y características** del documento. Ahora, vamos a profundizar en las partes que lo forman.

- **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

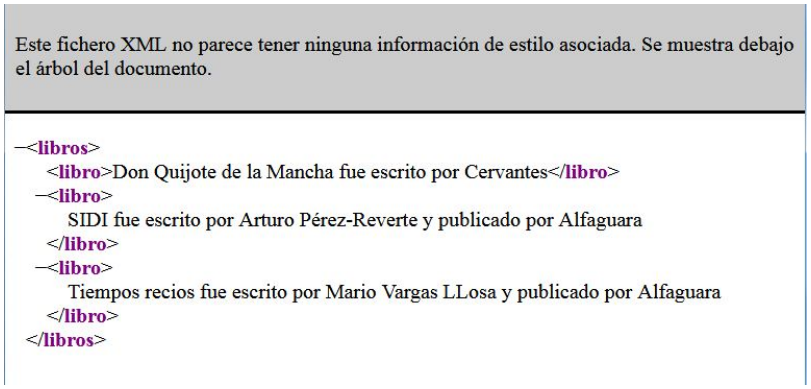


Figura 2.2.7.: Documento generado con entidades internas sustituidas

- **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.

2.3.7 Asociación con Documentos XML

Una vez que tenemos creado el **fichero XSD** debemos asociarlo con un **fichero XML**. El modo de asociar un esquema a un documento XMl es mediante un espacio de nombres al ejemplar del documento, donde se indica la ruta de la localización de los ficheros de esquemas mediante su URI, precedida del prefijo **xs:**.

Para que el documento XML siga las reglas no utilizaremos **<!DOCTYPE>**, sino que utilizaremos atributos especiales en el elemento raíz del documento XML.

Primero, al igual que en el documento XML Schema , necesitamos definir los dos espacios de nombres, el correspondiente al documento XML (que se suele usar sin abreviatura) y el espacio de nombres de XML Schema, que suele usar el prefijo xs.

Además es necesario indicar donde está el archivo XML Schema que contiene las reglas de aplicación para el documento. Esto se realiza mediante el atributo **schemaLocation**.

A continuación se muestra un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns="http://www.iesaguadulce.es/doc" xmlns:xs="http://w3.org/2001/XMLSchema-instance"
xs:schemaLocation="esquema.xsd">
....
</documento>
```

Figura 2.3.5.: Enlace de documento XML con el esquema XSD

Como vemos, se indica el **espacio por defecto de nombres**, que coincide con el declarado en el documento del esquema, se indica el **espacio de nombres del esquema**, que siempre será la misma url, y se asocia este espacio de nombres con el prefijo **xs**.

En el caso de que el esquema no definamos el espacio de nombres por defecto del documento, este quedaría de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns:xs="http://w3.org/2001/XMLSchema-instance"
           xs:noNamespaceSchemaLocation="esquema.xsd">
....
</documento>
```

2.3.8 Documentación del Esquema

Una vez que hemos visto como crear el esquema, vamos a ver como incorporar cierta documentación.

Podemos pensar que una forma de hacerlo es añadir comentarios, el problema con esto es que los analizadores no garantizan que los comentarios no se modifiquen al procesar el documento, y por tanto, que los datos añadidos no se pierdan.

En lugar de usar comentarios, XML Schema tiene definido un elemento **xs:annotation** que permite **almacenar información adicional**. Este elemento a su vez puede contener una combinación de otros dos, los cuales son:

- **xs:documentation**: ademas de contener elementos de esquema pueden contener elementos XML bien estructurados. También permite determinar el idioma del documento mediante el atributo **xs:lang**.
- **xs:appinfo**: se diferencia muy poco del elemento anterior, aunque lo que se pretendió es que xs:documentation fuera legible para los usuarios y que xs:appinfo guardase información para los programas de software. También se usa para generar una ayuda contextual para cada elemento declarado del esquema.

En la figura 2.3.6 se muestra un ejemplo de documentación de un esquema XML Schema. Como podemos ver, las anotaciones se pueden anidar dentro de los elementos, incluyendo a xs:appinfo para proporcionar información contextual de ese elemento concreto.

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

    <xs:annotation>
        <xs:documentation xml:lang ="es-es">
            Materiales para formación e-Learning
            <modulo>Lenguajes de marcas y sistemas de gestión de información.<modulo>
            <fecha_creación> 2011<fecha_creacion>
            <autor> Nuky La Bruji</autor>
        </xs:documentation>
    </xs:annotation>

    <xs:element name="lmsgi"  type=xs:string>
        <xs:annotation>
            <xs:appinfo>
                <texto_de_ayuda>Nombre completo del tema</texto_de_ayuda>
            <xs:appinfo>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

Figura 2.3.6.: Ejemplo de documentación del esquema XML

2.4 Herramientas de Creación y Validación

Al igual que hasta ahora, para crear y validad un documento XML basta con un editor de texto plano, aunque no es lo ideal. Hay herramientas que nos facilitan el trabajo permitiendo al usuario visualizar, validar y editar documentos en el lenguaje XML de forma mucho más rápida y cómoda. Algunas de estas aplicaciones son:

- **Notepad++**
- **Editix XML Edito**
- **XML Copy Editor**
- **NetBeans**
- **VSCode**

También tenemos herramientas de validación online, entre las que cabe destacar:

- https://www.xmlvalidation.com/
- xsdvalidation

2.5 Ejercicio Resuelto

En esta sección vamos a incluir un ejercicio resuelto en el que crearemos documentos tanto, DTD como XSD, así como un documento XML que valide a estos dos documentos.

2.5.1 Caso Práctico

La empresa Reggio tiene establecimientos por toda Italia, pero su sede central está en Cesena, al igual que el almacén donde se distribuye a todos los demás establecimientos. Esta empresa distribuye alpiste para pájaros, así como otros artículos ornitológicos.

Cada establecimiento tiene una tienda, así como un almacén, que pueden o no estar en la misma ubicación. Cuando se hace un pedido a la fábrica por parte de los establecimientos, éstas reciben los artículos en su almacén, y la documentación (albarán y pago) se remite a la tienda.

Cada **pedido** debe tener los datos siguientes:

- **Datos del establecimiento** que realiza el pedido (Nombre, dirección para envío y dirección almacén (si es la misma, sólo aparecerá una).
- **Código de pedido** (Cadena de caracteres formada por: Código establecimiento (1 letra y 2 números), número pedido (4 números), un guión y Año (dos números), por ejemplo: E180021-17.
- **Nombre del empleado** que realiza el pedido.
- **Fecha** de pedido.
- **Tipo de envío**, cuyos valores son: Normal o Urgente.

Respecto a los **artículos** del pedido, se guardarán los siguientes datos:

- **Código** del artículo (formado por tres letras y 3 números, por ejemplo: ZZZ134.
- **Número** de unidades.
- **Precio** por unidad.
- **Observaciones**.

2.5.2 Creación del DTD

En primer lugar vamos a crear un documento DTD con las especificaciones pedidas y un documento XML que valide estas especificaciones.

```
<!ELEMENT pedido (establecimiento,empleado,fecha_pedido,articulos)>
<!ATTLIST pedido cod_pedido CDATA #REQUIRED>
<!ATTLIST pedido tipo (normal|urgente) #REQUIRED>

<!ELEMENT establecimiento (nomb_estab,direccion_envio,direccion_almacen?)>
<!ELEMENT nomb_estab (#PCDATA)>
<!ELEMENT direccion_envio (via,numero,localidad,provincia,cp)>
<!ELEMENT direccion_almacen (via,numero,localidad,provincia,cp)>
<!ELEMENT via (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT localidad (#PCDATA)>
<!ELEMENT provincia (#PCDATA)>
<!ELEMENT cp (#PCDATA)>

<!ELEMENT empleado (#PCDATA)>
<!ELEMENT fecha_pedido (#PCDATA)>

<!ELEMENT articulos (articulo+)>
<!ELEMENT articulo (unidades,precio,observaciones?)>
<!ATTLIST articulo cod_articulo CDATA #REQUIRED>
<!ELEMENT unidades (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
<!ATTLIST precio moneda CDATA #REQUIRED>
<!ELEMENT observaciones (#PCDATA)>
```

Figura 2.5.1.: DTD del ejercicio propuesto

Una vez que hemos creado el DTD, deberemos crear un XML que verifique estas restricciones. En nuestro caso, podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pedido SYSTEM "tarea04_01.dtd">

<pedido cod_pedido="E180021-17" tipo="urgente">

    <establecimiento>
        <nomb_estab>Manitoba</nomb_estab>
        <direccion_envio>
            <via>Vittorio</via>
            <numero>1</numero>
            <localidad>Cesena</localidad>
            <provincia>Cesena</provincia>
            <cp>00400</cp>
        </direccion_envio>
        <direccion_almacen>
            <via>Vittorio</via>
            <numero>5</numero>
            <localidad>Cesena</localidad>
            <provincia>Cesena</provincia>
            <cp>00400</cp>
        </direccion_almacen>
    </establecimiento>

    <empleado>Roberto Rossini</empleado>
    <fecha_pedido>2017-05-01</fecha_pedido>

    <articulos>
        <articulo cod_articulo="ZZZ134">
            <unidades>1</unidades>
            <precio moneda="EUR">1.05</precio>
            <observaciones>Tenere lontano dalla portata dei bambini</observaciones>
        </articulo>
        <articulo cod_articulo="ZZZ137">
            <unidades>2</unidades>
            <precio moneda="EUR">3.50</precio>
        </articulo>
        <articulo cod_articulo="ZZZ139">
            <unidades>5</unidades>
            <precio moneda="EUR">5.50</precio>
        </articulo>
    </articulos>

</pedido>
```

Figura 2.5.2.: Documento XML válido para el DTD

El documento XML podría haber sido diferente, siempre que respetemos el DTD.

2.5.3 Creación del XML Schema

Una vez que hemos creado el documento DTD y el XML correspondiente vamos a crear el XML Schema. En éste, podremos establecer más restricciones que en el DTD, como hemos visto a lo largo de este tema. También el documento será más complejo de crear.

En primer lugar vamos a crear el documento XSD. Como el documento es bastante largo, se ha divido en dos partes para que quepa bien en un página.

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

<!-- Definicion del vocabulario -->
    <xs:element name="pedido">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="establecimiento"/>
                <xs:element ref="empleado"/>
                <xs:element ref="fecha_pedido"/>
                <xs:element ref="articulos"/>
            </xs:sequence>
            <xs:attributeGroup ref="attlist.pedido"/>
        </xs:complexType>
    </xs:element>

    <xs:attributeGroup name="attlist.pedido">
        <xs:attribute name="cod_pedido" type="cod_ped" use="required"/>
        <xs:attribute name="tipo">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="normal"/>
                    <xs:enumeration value="urgente"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:attributeGroup>

    <xs:simpleType name="cod_ped">
        <xs:restriction base="xs:string">
            <xs:length value="10"/>
            <xs:pattern value="[A-Za-z][0-9]{2}[0-9]{4}-[0-9]{2}"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="establecimiento">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nomb_estab"/>
                <xs:element ref="direccion_envio"/>
                <xs:element ref="direccion_almacen"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="nomb_estab" type="xs:string" />
    <xs:element name="direccion_envio">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="via"/>
                <xs:element ref="numero"/>
                <xs:element ref="localidad"/>
                <xs:element ref="provincia"/>
                <xs:element ref="cp"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="via" type="xs:string" />
    <xs:element name="numero" type="xs:string" />
    <xs:element name="localidad" type="xs:string" />
    <xs:element name="provincia" type="xs:string" />
    <xs:element name="cp">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:length value="5" />
                <xs:pattern value="[0-9]{5}"></xs:pattern>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
```

Figura 2.5.3.: Documento XML Schema (Parte 1)

```
    <xs:element name="direccion_almacen">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="via"/>
                <xs:element ref="numero"/>
                <xs:element ref="localidad"/>
                <xs:element ref="provincia"/>
                <xs:element ref="cp"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="empleado" type="xs:string" />
    <xs:element name="fecha_pedido" type="xs:date" />

    <xs:element name="articulos">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="articulo"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="articulo">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="unidades"/>
                <xs:element ref="precio"/>
                <xs:element ref="observaciones" minOccurs="0" />
            </xs:sequence>
            <xs:attributeGroup ref="attlist.articulo"/>
        </xs:complexType>
    </xs:element>

    <xs:attributeGroup name="attlist.articulo">
        <xs:attribute name="cod_articulo" type="cod_articulo" use="required"/>
    </xs:attributeGroup>

    <xs:simpleType name="cod_articulo">
        <xs:restriction base="xs:string">
            <xs:length value="6"/>
            <xs:pattern value="[A-Za-z]{3}[0-9]{3}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="unidades" type="xs:positiveInteger"/>
    <xs:element name="precio">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attributeGroup ref="attlist.precio" />
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>

    <xs:attributeGroup name="attlist.precio">
        <xs:attribute name="moneda" type="xs:NMTOKEN" fixed="EUR" use="required"/>
    </xs:attributeGroup>
    <xs:element name="observaciones" type="xs:string"/>
</xs:schema>
```

Figura 2.5.4.: Documento XML Schema (Parte 2)

Una vez creado el documento XML Schema, creamos el XML que lo valide, que es similar al visto en el punto anterior, cambiando solo el modo de enlazar el documento con el esquema.

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

    <pedido xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
            xsi:noNamespaceSchemaLocation="reggio.xsd"
            cod_pedido="E180021-17"
            tipo="urgente">

        <establecimiento>
            <nomb_estab>Manitoba</nomb_estab>
            <direccion_envio>
                <via>Vittorio</via>
                <numero>1</numero>
                <localidad>Cesena</localidad>
                <provincia>Cesena</provincia>
                <cp>00400</cp>
            </direccion_envio>
            <direccion_almacen>
                <via>Vittorio</via>
                <numero>5</numero>
                <localidad>Cesena</localidad>
                <provincia>Cesena</provincia>
                <cp>00400</cp>
            </direccion_almacen>
        </establecimiento>
        <empleado>Roberto Rossini</empleado>
        <fecha_pedido>2017-05-01</fecha_pedido>

        <articulos>
            <articulo cod_articulo="ZZZ134">
                <unidades>1</unidades>
                <precio moneda="EUR">1.05</precio>
                <observaciones>Tenere lontano dalla portata dei bambini</observaciones>
            </articulo>
            <articulo cod_articulo="ZZZ137">
                <unidades>2</unidades>
                <precio moneda="EUR">3.50</precio>
            </articulo>
            <articulo cod_articulo="ZZZ139">
                <unidades>5</unidades>
                <precio moneda="EUR">5.50</precio>
            </articulo>
        </articulos>
</pedido>
```

Figura 2.5.5.: Documento XML que valida el XML Schema

2.6 Documentación de Apoyo

En esta última sección vamos a incluir una serie de enlaces con documentación extra, incluyendo vídeos, para completar el contenido de este tema y para ver como se realizan los documentos de validación paso a paso. Los enlaces son los siguientes:

- Ejercicio resuelto de DTD y XML Schema - Parte 1 (Vídeo)
- Ejercicio resuelto de DTD y XML Schema - Parte 2 (Vídeo)
- Resumen DTD (PDF)
- Ejercicios DTD (Web)
- DTD y XML Schema (PDF)
- Estructura XML Schema (Vídeo)

- Mi primer DTD (Vídeo)
- DTD: Elementos (Vídeo)
- Creación de DTD sencillo - Parte 1 (Vídeo)
- Creación DTD sencillo - Parte 2 (Vídeo)
- XML Schema: Data Types (PDF)

Tema 3

Utilización de Lenguajes de Marcas en Entornos Web

Tema 4

Utilización de Lenguajes de Marcas en la Sindicación de Contenidos

En este tema vamos a estudiar como podemos utilizar los lenguajes de marcas para la **sindicación de contenidos**, un método que nos permite distribuir contenidos en la Web de forma que los usuarios puedan acceder a esta información basándose solo en los contenidos de su interés.

4.1 Sindicación de Contenidos

La sindicación de contenidos, desde el punto de vista Web, permite a **un sitio Web** utilizar los **servicios o contenidos** ofertados por otra web diferente. Estos servicios, juntos con los **metadatos** que tienen asociados a ellos en el sitio original, constituyen lo que conocemos como **feeds** o **canales de contenido**.

Esta redifusión suele realizarse bajo una **licencia de normas de uso**, en lugar de mediar un contrato para regular los derechos de los contenidos.

En la actualidad, la **redifusión Web** consiste en ofrecer en una página contenidos desde otra fuente web, permitiendo que los usuarios obtengan actualizaciones sobre dichos contenidos. Las **fuentes** suelen codificarse en lenguaje **XML**, aunque es válido hacerlo en cualquier lenguaje que pueda transportar el protocolo http.

4.1.1 Características

Para realizar esta sindicación de contenidos deberemos conseguir que la información, almacenada por ejemplo en un fichero local de un servidor, sea mostrada a los usuarios/as que quieran leerla. En el caso de que la información se haya codificado en un documento HTML, esto se llevará a cabo actualizando dicho documento en el directorio adecuado del servidor.

Actualmente es habitual el uso de un **CMS** (Content Management System), en cuyo caso los datos se encontrarán en un **repositorio** y antes de ser servidos al cliente sufren alguna transformación para pasarlos al formato adecuado.

Esta **transformación** puede corresponder a alguno de los siguientes casos:

- Documento XML ->Transformación XSLT ->Documento XHTML
- Bases de Datos ->Script en Perl ->Documento HTML

- Texto plano ->ASP ->Documento HTML
- Mente del autor ->Editor de textos ->Documento HTML

Al utilizar cualquier tipo de CMS la transformación puede replicarse. Además, podemos tener mas de una entrada para la información así como **varias salidas**, pudiendo, por ejemplo, generar tanto **ficheros HTML** como **canales RSS**, tal y como se muestra en la siguiente imagen.

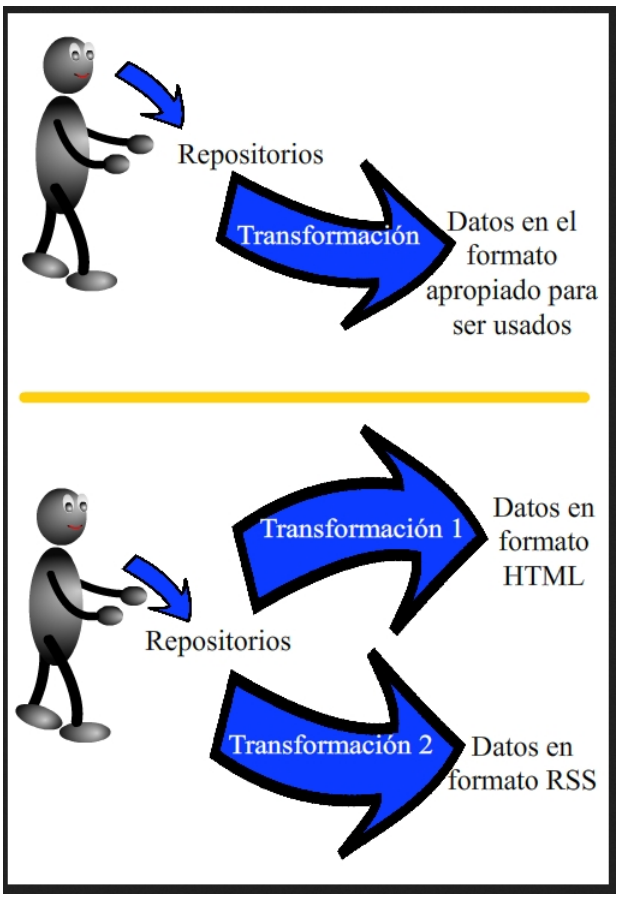


Figura 4.1.1.: Generación de salidas HTML o RSS

Publicar en la Web puede ser visto como un **flujo de información**. Para que una web sea **suministradora** de un canal en su cabecera **<head>** hay que incluir un **enlace** al canal de contenidos, incluyendo un elemento **<link>** con sus correspondientes atributos **rel**, **href**, **type** y **title**.

Además tendremos que especificar el enlace para acceder al contenido, por lo que en algún lugar el elemento **<body>** de nuestro archivo HTML deberemos añadir un enlace usando el elemento **<a>** y completar sus atributos **href**, **type** y **rel**.

A continuación mostramos un ejemplo de un archivo HTML suministrador de un canal de noticias en formato Atom y RSS:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <title>RSS y ATOM</title>
    <link rel="stylesheet" href="style.css">
    <!--************************* NOTICIAS ******************************-->
    <link rel="alternate" href="feed/canal.rss" type="application/rss+xml" title="RSS">
    <link rel="alternate" href="feed/canal.atom" type="application/atom+xml" title="Atom">
    <!--************************* NOTICIAS ******************************-->

</head>

<body>
    <header>
        <h1>Unidad 3: Sindicación de contenidos web.</h1>
        <h2>Ejemplo práctico: Enlazar archivos RSS y Atom a una web</h2>
    </header>

    <main>
        <h3>Texto de relleno</h3>
        <p>
            Lorem, ipsum dolor sit amet consectetur adipisicing elit.
            Voluptatibus est sit distinctio ad et culpa maiores ipsam sunt
            dolores nihil, beatae consequuntur, illum mollitia! Veritatis, suscipit
            sapiente! Laboriosam, mollitia dolor.Ipsa expedita exercitationem
            voluptatum id eveniet ipsam nemo eos alias culpa hic. Non, distinctio.
            Alias, necessitatibus deleniti ipsum ullam aliquam ex. Nam cum tempora nisi
            adipisci vel voluptatum officiis asperiores.Dicta, nostrum optio. Facilis
            porro odit distinctio quas, debitis doloremque maiores eveniet, temporibus
            accusantium cum omnis quibusdam! Aperiam, debitis ipsam! Vitae dolorum beatae
            recusandae a laudantium velit eligendi, odio natus.Facere optio voluptate ut,
            ducimus deserunt iusto culpa. Blanditiis, rem. Reiciendis, sapiente.
            Alias necessitatibus recusandae voluptatibus minus beatae dolorum velit vero
            sequi voluptatum optio qui dignissimos, nemo voluptatem iste aliquid.
        </p>
        </main>

    <footer>
        <p>
            <hr>
            <!--************************* NOTICIAS ******************************-->
            <a rel="alternate" href="feed/canal.rss" type="application/rss+xml">
                <img src="img/valid-rss.png" alt="enlace RSS">
            </a>
            <a rel="alternate" href="feed/canal.atom" ype="application/atom+xml">
                <img src="img/valid-atom.png" alt="enlace Atom">
            </a>
            <!--************************* NOTICIAS ******************************-->
            <hr>
        </p>
    </footer>

</body>

</html>
```

Figura 4.1.2.: Documento HTML con canal RSS y Atom

4.1.2 Ventajas de la Redifusión de Contenidos

El uso de la redifusión de contenidos de otros propietarios nos puede aportar un número de ventajas, siendo las principales las siguientes:

- **Aumentar el tráfico** de nuestro sitio Web.
- Ayuda a que los **usuario/as visiten frecuentemente** nuestro sitio web.
- Favorece el **posicionamiento** del sitio en **buscadores**.
- Ayuda a **establecer relaciones** entre **diferentes sitios web** dentro de la comunidad.
- Permite a **otras personas** añadir características a los **servicios del sitio web**, como por ejemplo, notificaciones de actualizaciones de contenido, mensajes instantáneos, etc.., aunque esto requiere tecnologías adicionales.
- Enriquece internet impulsando la **tecnología semántica** y **fomenta la reutilización**.

4.2 Ámbitos de Aplicación

La redifusión web no es sólo un fenómeno vinculado a los **weblogs**, aunque ha contribuido mucho a su popularización. Siempre se han sindicado contenidos y se ha compartido todo tipo de información usando documentos XML.

De esta forma, podemos ofrecer contenidos propios que sean mostrados en otras web de forma integrada, lo que aumenta el valor de la página web que muestra el contenido y también nos genera a nosotros más valor, ya que normalmente la redifusión web enlaza con los contenidos originales. Además, ésta puede aplicarse a cualquier tipo de contenido, ya sea **texto**, **audio**, **vídeo** e **imágenes**.

Desde el punto de vista de los **suscriptores**, la redifusión web permite, entre otras cosas, la actualización profesional. Mediante la suscripción a sitios relevantes, el usuario/a puede estar al día de temas relevantes relativos a su profesión, recibiendo noticias e información en su blog o su programa agregador de noticias.

4.3 Tecnologías de Creación de Canales de Contenido

Hay un conjunto de tecnologías que se usan para la creación de canales de contenido, o canales RSS. Los principales **estándares** utilizados son los siguientes:

- **RSS** (Really Simple Syndication): este estándar forma parte de la familia de formatos XML, desarrollado para compartir información entre sitios web que se actualiza con frecuencia. Además, se usa en la conexión de sistema de mensajería instantánea, y permite la conversión de mensajes RSS en correos electrónicos o la transformación de los enlaces favoritos del navegador a RSS. Ha sido desarrollado por tres organizaciones diferentes lo que ha dado lugar a **siete formatos** diferentes:
    - **RSS 0.90**: es el estándar que creo la empresa **Netscape** en 1990. Se basa en la especificación **RFD de metadatos**, con la intención de que su proyectos, My Netscape, estuviera formado por titulares de otras web.
    - **RSS 0.91**: es la versión simplificada de RSS 0.90 que Netscape lanzó con posterioridad. Esta versión tuvo poco éxito y su desarrollo se detuvo, aunque la empresa UserLand Software decidió usar esta versión para desarrollar blogs.
    - **RSS 1.0**: creado a partir del estándar **RSS 0.90**, es más estable y permite definir una cantidad mayor de datos que el resto de versiones RSS.
    - **RSS 2.0**: la empresa UserLand Software rechazó el estándar RSS 1.0 por considerarlo complejo, y continuó con el desarrollo de RSS 0.91, publicando diferentes versiones, las cuales

tenían una sintaxis incompleta y no cumplían con las normas XML. La versión 2.0 se publico para subsanar estos problemas.

- **Atom**: fue publicado como un estándar por el grupod de trabajo **Atom Publishing Format and Protocol** de la **IETF** en el **RFC4287**. Se desarrollo como una alternativa a RSS, con el fin de evita la confusión creada por la existencia de estándares similares para la sindicación de contenidos, entre los que existían diferentes incompatibilidades. En lugar de sustituir estos, se creó un nuevo estándar que convive con ellos. Se caracteriza por su flexibilidad, ya que permite tener un mayor control sobre la cantidad de información que se muestra en los agregadores.

En los siguientes enlaces, puedes consultar las especificaciones de estos estándares:

- RSS 1.0
- RSS 2.0
- Atom

4.4 Estructura de un Canal de Contenido

Para crear una canal de contenidos es necesario crear un fichero siguiendo la especificaciones de **RSS** o **Atom** que están basadas en XML. Este **fichero**, se publicará en uno de los directorios del sitio web.

Los **elementos básicos** de estos ficheros son los siguientes:

- **Declaración del documento XML** y definición de la **codificación empleada** en el documento. Esta última será, preferentemente, **UTF-8**, que es la más ampliamente utilizada y una de las que permite mayor rango de caracteres.
- **Un canal** en el que se determina el sitio web asociado a la fuente web a la que hace referencia el fichero. Este estará formado por:
  - La propia **definición del canal**.
  - **Secciones**: cada una de estas hará referencia a la web que contiene uno de los servicios que se van a ofrecer. En un canal pueden incluirse tantas secciones como se quiera, lo que hace que una canal pueda tener un tamaño enorme si contiene un gran número de enlaces independientes.

En un principio, **no existe** ninguna **restricción** a la cantidad de canales que se pueden ofrecer desde un servicio web.

4.4.1 RSS

En esta sección vamos a crear un **canal de contenido** empleando el estándar **RSS**. Esto no servirá para explicar todos los elementos que debe contener un documento XML para la creación de dicho canal y las diferentes opciones y etiquetas que podemos emplear.

Para crear este canal, vamos a seguir los siguientes pasos:

1. En primer lugar, vamos a crear un fichero con **extensión .rss**, con un editor de texto o un IDE de nuestra preferencia.
2. Una vez creado el documento, vamos a comenzar a añadir los diferentes elementos necesarios para la creación del canal y que el documento sea válido, comenzando por la **declaración XML**,

donde especificaremos la versión de XML a usar así como la **codificación del documento**, que en nuestro caso va a ser **UTF-8**.

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

Figura 4.4.1.: Declaración XML en el documento RSS

3. A continuación, debemos de indicar el ejemplar. En este caso, el elemento raíz sera **<rss>**, donde además deberemos indicar, mediante el atributo **version**, la versión de RSS que vamos a emplear, en este caso, la 2.0.

```
<rss version="2.0">
...
</rss>
```

Figura 4.4.2.: Elemento raíz del documento RSS

4. Una vez creado el elemento raíz, debemos definir el canal empleando el elemento **<channel>**. Para su definición hay una serie de elementos obligatorios y otros opcionales. Los elementos obligatorios para definir el canal son los siguientes:
   - **<title>**: es el título del canal de noticias, suele ser el mismo que el de la página Web.
   - **<link>**: elemento con la dirección web de la página asociada al canal de noticias.
   - **<description**: este elemento contiene una breve descripción del canal de noticias.

   Estos elementos son obligatorios y siempre deben aparecer en en la definición de un canal, pero tenemos **muchos más elementos**, lo cuales no son obligatorios, pero que nos proporcionan diferentes funcionalidades, siendo los principales los siguientes:
   - **<language>**: Determina el idioma utilizado en el canal de noticias, en el caso del español su valor será es. Para indicar español de España: es-es.
   - **<item>**: Definirá cada una de las noticias del canal.
   - **<copyright>**: Licencia de los contenidos del canal de noticias.
   - **<managingEditor>**: Correo electrónico del editor de contenidos del canal de noticias.
   - **<webMaster>**: Correo electrónico del responsable técnico del canal de noticias.
   - **<pubDate>**: Fecha de publicación del canal de noticias. La fecha se debe escribir en formato: Sat, 07 Sep 2002 0:00:01 GMT
   - **<lastBuildDate>**: Fecha del último cambio en los contenidos del canal. También en formato: Sat, 07 Sep 2002 0:00:01 GMT
   - **<category>**: Categoría del canal de noticia. Puede estar asociada a una o más categorías.

- **<generator>**: En caso de que el canal de noticias sea generado, nombre del programa que se utiliza para hacerlo.
- **<docs>**: URL de la documentación sobre el formato usado en el archivo RSS.
- **<cloud>**: Permite que los procesos se registren en una nube para recibir notificaciones de las actualizaciones del canal de noticias.
- **<ttl>**: "Time to live", minutos que el canal de noticias esta en cache antes de actualizarse.
- **<image>**: Especifica una imagen GIF, JPEG o PNG que puede ser mostrada en el canal de noticias. Tiene tres atributos obligatorios: **url**, **title** y **link**. También se puede indicar el alto (height), ancho (width) y una descripción (description) con los atributos opcionales.
- **<textInput>**: Especifica una caja de texto de entrada que puede ser mostrada en el canal de noticias.
- **<skipHours>**: En este elemento pueden definirse subelementos <hour>que representan una hora GMT en las cuales los agregadores de noticias no se actualizarán.
- **<skipDays>**: En este elemento pueden definirse subelementos <day>que representan los días de la semana en los cuales los agregadores de noticias no se actualizarán.

Como vemos por la lista anterior, tenemos una gran variedad de elementos que nos permiten añadir información y configurar nuestro canal de contenidos. En nuestro caso, hemos añadidos los que se pueden ver en la siguiente figura:

```
<channel>
   <title>Canal de noticias</title>
   <link>http://www.noticias.com</link>
   <description>En este canal tendrás noticias actualizadas a diario</description>

   <language>es</language>
   <pubDate>Mon, 7 May 2018 17:32:06 GMT</pubDate>
   <managingEditor>paco@noticas.com (Paco López)</managingEditor>
   <image>
       <url>http://www.noticias.com/img/logo.png</url>
       <title>Canal de noticias</title>
       <link>http://www.noticias.com</link>
   </image>
   <copyright>Creative Common</copyright>
</channel>
```

Figura 4.4.3.: Creación del elemento channel y subelementos

En primer lugar hemos puesto los elementos obligatorios y a continuación hemos agregado algunos otros como language, pubDate, image o copyright.

5. Por último, vamos a agregar una cuantas secciones dentro de la definición del canal con el elemento **<item>**. Un elemento **<channel>** puede contener cualquier número de elementos **<item>**, definiendo una sección del canal cada uno. Estas secciones pueden estar completas o contener una sinopsis y un enlace a la sección completa.

   Todos los subelementos del elemento <item> son opcionales, aunque al menos debe aparecer un **título** y una **descripción** de la sección. <description>

   Los principales **subelementos** de <item> que existen son los siguientes:

- **<title>**: Título de la noticia.
- **<link>**: URL de la noticia enlazada, que ha de pertenecer al dominio establecido en el canal.
- **<description>**: Un resumen de la noticia.
- **<author>**: Correo electrónico del autor de la noticia.
- **<category>**: Categoría de la noticia. Puede estar asociada a una o más categorías.
- **<comments>**: URL del hilo de los comentarios relacionados con la noticia.
- **<enclosure>**: Descripción de objetos multimedia asociados a la noticia. Tiene tres atributos obligatorios: **url**, **length** y **type**.
- **<guid>**: Cadena única que identifica la noticia.
- **<pubDate>**: Fecha de publicación de la noticia. La fecha se debe escribir en formato: Sat, 07 Sep 2002 0:00:01 GMT.
- **<source>**: Canal RSS del que proviene la noticia. Tiene un atributo obligatorio **url**.

En nuestro ejemplo, nosotros hemos añadido 3 secciones dentro del elemento <channel> con los elementos que podemos ver a continuación:

```
<item>
   <title>Primera noticia de actualidad</title>
   <description>Esta noticia es un ejemplo de noticia de actualidad</description>
   <link>https://www.americannews.com/</link>
   <guid>http://www.americannews.com/today</guid>
   <author>John@americannews.com (John Johnson)</author>
   <category>Actualidad</category>
   <pubDate>Sun, 6 May 2018 12:24:26 GMT</pubDate>
</item>

<item>
   <title>Segunda noticia de actualidad</title>
   <link>https://www.titular.es/hoy/segunda.html</link>
   <guid>http://www.titular.es/segunda.html#titular</guid>
</item>

<item>
   <description>Otra noticia de ejemplo</description>
   <guid>http://www.quepais.com/actualidad/primera.html#titular</guid>
   <author>maria@actualidad.quepais.com (María Martín)</author>
   <pubDate>Sun, 6 May 2018 12:24:26 GMT</pubDate>
</item>
```

Figura 4.4.4.: Secciones del canal RSS

Como podemos ver, el número de elementos presentes en cada elemento <item>puede variar según nuestra preferencia o la cantidad de información que queramos añadir.

No todos los elementos que hemos visto se han empleado en el ejemplo, habiendo algunos como el elemento **<enclosure>** que sería conveniente revisar la documentación oficial para ver más detalladamente como se usa cuales son sus atributos. Al final de esta sección, se incorporarán algunos enlaces hacia la documentación oficial.

6. Por último, ponemos todos los elementos juntos y ya tendremos creado nuestro canal RSS, siendo el documento final resultante el que podemos ver en la siguiente figura.

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

<rss version="2.0">

    <!--CANAL DE NOTICIAS-->

    <channel>
        <title>Canal de noticias</title>
        <link>http://www.noticias.com</link>
        <description>En este canal tendrás noticias actualizadas a diario</description>

        <language>es</language>
        <pubDate>Mon, 7 May 2018 17:32:06 GMT</pubDate>
        <managingEditor>paco@noticas.com (Paco López)</managingEditor>
        <image>
        <url>http://www.noticias.com/img/logo.png</url>
        <title>Canal de noticias</title>
        <link>http://www.noticias.com</link>
        </image>
        <copyright>Creative Common</copyright>

        <!--LISTADO DE NOTICIAS-->
        <!--1º Noticia-->
        <item>
            <title>Primera noticia de actualidad</title>
            <description>Esta noticia es un ejemplo</description>
            <link>https://www.americannews.com/</link>
            <guid>http://www.americannews.com/today</guid>
            <author>John@americannews.com (John Johnson)</author>
            <category>Actualidad</category>
            <pubDate>Sun, 6 May 2018 12:24:26 GMT</pubDate>
        </item>

        <!--2º Noticia-->
        <item>
            <title>Segunda noticia de actualidad</title>
            <link>https://www.titular.es/hoy/segunda.html</link>
            <guid>http://www.titular.es/segunda.html#titular</guid>
        </item>

        <!--3º Noticia-->
        <item>
            <description>Otra noticia de ejemplo</description>
            <guid>http://www.quepais.com/actualidad/primera.html#titular</guid>
            <author>maria@actualidad.quepais.com (María Martín)</author>
            <pubDate>Sun, 6 May 2018 12:24:26 GMT</pubDate>
        </item>

    </channel>
</rss>
```

Figura 4.4.5.: Documento RSS completo

Como hemos visto, la creación de un canal RSS puede ser bastante simple, aunque como hemos comentado, no hemos empleado todos los elementos en este ejemplo y puede que haya algunos que requieran ver ejemplos concretos para comprender bien como funcionan, por eso, se incluyen los siguiente enlaces para obtener información más detallada sobre los elementos y sus atributos.

- RSS 2.0 en W3C
- Validador RSS 2.0 del W3C
- Especificación del estándar RSS 2.0

4.4.2 Atom

En la sección anterior ya creamos una canal de contenido usando el estándar RSS 2.0, en esta sección, vamos a crear un canal pero empleando el **estándar Atom**, para ellos, vamos a enumerar los pasos para su creación al igual que hicimos en la sección anterior.

Como veremos, algunos pasos son prácticamente iguales, mientras que otros difieren tanto en los elementos como en sus atributos.

1. En primer lugar creamos un documento con extensión **.atom** con nuestro editor o IDE favorito.
2. A continuación, al igual que con RSS, vamos a agregar la **declaración XML** y la **codificación** que vamos emplear en el documento, que como vemos en la siguiente figura, no difiere del documento anterior.

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

Figura 4.4.6.: Declaración XML en el documento Atom

3. Lo siguiente definir el canal. A diferencia de RSS, con Atom definimos directamente el canal como elemento raíz empleando la etiqueta **<feed>**, a la que tendremos que añadir el estándar Atom usado, mediante el atributo **xmlns** y el lenguaje utilizado en el fichero con el atributo **xml:lang**, como podemos ver en la siguiente figura:

```
<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="es-es">
...
</feed>
```

Figura 4.4.7.: Elemento feed del canal Atom

4. Al igual que con los canales de RSS, en Atom tenemos ciertos **elementos obligatorios** que todo canal debe incluir, entre los que nos podemos encontrar los siguientes:
   - **<title>**: Es el título del canal de noticias.
   - **<id>**: Identificador del canal de noticias. Habitualmente su URI.
   - **<updated>**: Fecha de publicación del canal de noticias. La fecha se debe escribir en el formato: CCYY-MM-DDTHH:MM:SSZ, donde T es el separador entre la fecha y la hora y Z indica que la hora hace referencia al sistema de tiempo universal, esto es la hora zulú, o la hora del meridiano de Greenwich. Ejemplo: 6 de febrero de 2010 a la 17:15 hora española tenemos que poner: 2010-02-06T16:15:00Z

   Además de estos elementos obligatorios, tenemos dos **elementos recomendados** que si bien no es indispensable que aparezcan, es una buena práctica influirlos en cada canal. Estos son:

   - **<link>**: Identificación del sitio web. Tiene un atributo obligatorio **href** y varios opcionales: **rel**, **type**, **hreflang**, **title** y **length**.

     Para el atributo **rel** deben indicarse dos valores, cada uno en un elemento link diferente:

- **self**: Referenciando la ubicación del fichero .atom.
- **alternate**: Referenciando la web.

- **<author>**: Cada elemento feed debe tener al menos un **author** a menos que todas las noticias **entry** tengan **author**. Los datos de autor se podrán recoger en los subelementos: **name**, **email** y **uri**.

Al igual que con los canales RSS, tenemos un conjunto de **elementos opcionales** que nos ayudarán a aportar información al canal o a incluir imágenes, logos, etc... Estos elementos son:

- **<category>**: Categoría del canal de noticia. Puede estar asociada a una o más categorías. Tiene un atributo obligatorio **term**.
- **<contributor>**: Datos de los contribuidores al canal de noticias. Sigue la misma estructura que **author**.
- **<generator>**: Identifica el software usado para generar el canal de noticias. Tiene dos atributos opcionales **uri** y **version**.
- **<icon>**: Identifica una pequeña imagen para ser usada de icono. La imagen debe ser cuadrada.
- **<logo>**: Identifica una imagen como referencia del canal de noticias. La imagen debe ser el doble de ancha que de alta.
- **<rights>**: Información sobre los derechos de autor. Permite el atributo **type** para indicar la codificación del texto (text, html, xhtml)
- **<subtitle>**: Descripción o subtitulo para el canal de noticias.

Nosotros hemos agregado, además de las etiquetas obligatorias, las recomendadas y algunas opcionales, como podemos ver en la siguiente figura.

```
<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="es-es">

   <title type="text">Canal de noticias</title>
   <id>http://www.noticias.com/</id>
   <updated>2018-05-07T17:32:06Z</updated>


   <link rel="self" type="application/atom+xml" href="feed/canal.atom" />
   <link rel="alternate" type="text/html" href="http://www.noticias.com" />
   <author>
       <name>Paco López</name>
       <email>paco@noticias.com</email>
   </author>

   <category term="novedades" />
   <generator version="1.0">vim</generator>
   <icon>img/icon.jpg</icon>
   <logo>img/logo.png</logo>
   <rights>Creative Common</rights>
   <subtitle type="text">Noticias de actualidad.</subtitle>
</feed>
```

Figura 4.4.8.: Elementos del canal feed en Atom

5. A continuación vamos a definir las secciones. En Atom, esto se lleva a cabo con el elemento **<entry>**. A diferencia del elemento **<item>** de RSS que no tenía **elementos obligatorios**, el elemento <entry>si los tiene, siendo estos similares a los del elemento <feed>, como podemos ver a continuación:

   - **<title>**: Es el título del canal de noticias.
   - **<id>**: Identificador del canal de noticias. Habitualmente su URI.
   - **<updated>**: Fecha de publicación del canal de noticias. La fecha se debe escribir en el formato: CCYY-MM-DDTHH:MM:SSZ, donde T es el separador entre la fecha y la hora y Z indica que la hora hace referencia al sistema de tiempo universal, esto es la hora zulú, o la hora del meridiano de Greenwich. Ejemplo: 6 de febrero de 2010 a la 17:15 hora española tenemos que poner: 2010-02-06T16:15:00Z

   También tenemos unos **elementos recomendados** a la hora de definir una noticia con **<entry>**, que son los siguientes:

   - **<link>**: Relacionado con el sitio web. Tiene un atributo obligatorio **href** y varios opcionales: **rel**, **type**, **hreflang**, **title** y **length**. Si no existe elemento **content** en la noticia debe haber un elemento **link** con el atributo **rel a alternate** que indique la página web de la noticia en el atributo href.
   - **<content>**: Contenido o enlace al texto completo de la noticia. Debe existir un elemento **content** si no hay elemento **link** con el valor **rel=“alternative”** o no hay un elemento **summary** en la noticia.
   - **<author>**: Cada elemento entry debe tener al menos un **author** a menos que haya un elemento author en el **elemento feed** o como contenido del elemento **source**. Los datos de autor se podrán recoger en los subelementos: **name**, **email** y **uri**.
   - **<summary>**: Breve resumen del contenido de la noticia. Este elemento debe añadirse si no hay elemento content en la noticia.

   Por último, vamos a listar los **elementos opcionales**, que en este caso no son tan numerosos con en los anteriores:

   - **<category>**: Categoría de la noticia. Puede estar asociada a una o más categorías. Tiene un atributo obligatorio **term**.
   - **<contributor>**: Datos de los contribuidores de la noticia. Sigue la misma estructura que **author**.
   - **<published>**: Fecha y hora de la creación o primera publicación de la noticia. La fecha se debe escribir en el formato: CCYY-MM-DDTHH:MM:SSZ
   - **<rights>**: Información sobre los derechos de autor. Permite el atributo **type** para indicar la codificación del texto (text, html, xhtml)
   - **<source>**: Contiene metainformación sobre el origen del canal de noticias si esta noticia está copiada de otro lugar.

   Nosotros, al igual que en el ejemplo de RSS hemos **añadido 3 secciones** con diferentes elementos opcionales.

6. Así, tras añadir todos los elementos que hemos descrito durante estos pasos, el documento resultante lo podemos ver en la siguiente figura:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- *************************** CANAL ******************************-->

<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="es-es">
    <title type="text">Canal de noticias</title>
    <id>http://www.noticias.com/</id>
    <updated>2018-05-07T17:32:06Z</updated>

    <link rel="self" type="application/atom+xml" href="feed/canal.atom" />
    <link rel="alternate" type="text/html" href="http://www.noticias.com" />
    <author>
        <name>Paco López</name>
        <email>paco@noticias.com</email>
    </author>

    <category term="novedades" />
    <generator version="1.0">vim</generator>
    <icon>img/icon.jpg</icon>
    <logo>img/logo.png</logo>
    <rights>Creative Common</rights>
    <subtitle type="text">Noticias de actualidad.</subtitle>

    <!--************************ NOTICIAS ******************************-->

    <entry>
        <title>Primera noticia de actualidad</title>
        <id>https://www.americannews.com/</id>
        <updated>2018-05-06T12:24:26Z</updated>

        <author>
            <name>John Johnson</name>
            <email>john@americannews.com</email>
        </author>
        <link rel="alternate" type="text/html" href="http://www.americann.com/today" />
        <summary>Estado del tiempo de la ciudad de Frankfurt</summary>

        <category term="actualidad"/>
        <contributor>
        <name>Ana Gómez</name>
        </contributor>
        <published>2018-05-10T09:12:30Z</published>
        <rights type="text">Copyright</rights>
    </entry>

    <entry>
        <title>Segunda noticia de actualidad</title>
        <link rel="alternate" type="text/html" href="https://www.ideal.es/hoy/1.html" />
        <updated>2019-12-18T19:25:00Z</updated>

        <id>https://www.titular.es/hoy/primera.html#titular</id>
        <summary>Esta es una noticia local de actualidad</summary>

        <category term="noticias locales"/>
    </entry>

    <entry>
        <title>Otra noticia de ejemplo</title>
        <link rel="alternate" type="text/html" href="http://www.pais.com/hoy/2.html" />
        <updated>2019-12-18T19:27:00Z</updated>

        <id>http://www.quepais.com/actualidad/segunda.html#titular</id>
        <summary>Aqui podras encontrar todo tipo de noticias de actualidad</summary>
        <author>
            <name>María Martín</name>
            <email>maria@actualidad.quepais.com</email>
        </author>

        <published>2018-05-06T12:24:26Z</published>
        <rights type="text">Copyright</rights>
    </entry>
</feed>
```

Figura 4.4.9.: Documento del Canal con Atom completo

Al igual que con el canal creado usando RSS, hay elementos y atributos que no se han usado en este ejemplo y que sería conveniente consultar más a fondo, para ver ejemplos de su uso y los valores que pueden tomar sus atributos. En los **siguiente enlaces** podemos ampliar la información y ver cada elemento de forma más detallada:

- Atom 1.0 en W3C
- Syndication | Atom Enabled
- Especificación de Atom 1.0
- Validador de Atom 1.0 de la W3C

4.5 Validación

Una vez creado nuestro fichero, ya sea RSS o Atom, el siguiente paso será validarlo. En internet podemos encontrar múltiples lugares que ofrecen este servicio.

Para **validar un documento**, ya sea de Atom o RSS, solemos tener dos opciones, una es **introducir la URI** en la que esta alojado el fichero, en cuyo caso el validador comprobará que el documento alojado en esa dirección es válido, o bien **introduciendo el código** en el validador, el cual nos lo validará directamente.

Una vez introducido el documento, ya sea con un método u otro, el validador nos dirá si éste es válido. En caso de que no sea, nos mostrará cuales son los errores. También nos puede mostrar **sugerencias**, para por ejemplo, hacer el documento compatible con otro estándar incluyendo un enlace.

Una véz validado, nos ofrecerán la posibilidad de incluir un **banner con el resultado** de la validación en nuestra página web, mediante la inclusión de un enlace que nos proporcionan.

Uno de los validadores más empleados es el validador de la W3C, que nos permite validar documentos tanto de RSS como de Atom mediante una URI o introduciendo directamente el código.

4.6 Utilización de Herramientas

Aunque podemos crear un documento RSS o Atom prácticamente con cualquier editor, hay editores e IDE‘s que nos simplifican el trabajo añadiendo diferentes funcionalidades que no tiene un editor simple.

Un ejemplo de este tipo de herramientas es **PSPad Editor**, un editor de textos con licencia freeware orientado a la programación. Algunas de las funcionalidades que nos ofrece este editor son las siguientes:

- Trabajar con distintos estándares.
- Importar documentos de texto con CSV y HTML..
- Editar HTML.
- Editar documentos XML e imágenes.
- Actualizar las fuentes por vía FTP.
- Exportar documentos RSS a HTML, CSV y Javascript.

Aunque hemos mencionado PSPad, hay una gran variedad de editores e IDE‘s que nos pueden ofrecer estas funcionalidades y otras muchas más, como pueden set **Netbeans** o **VSCode** entre otros.

4.7 Directorios de Canales de Contenido

Los **directorios de canales de contenido** permiten que un fichero RSS esté disponible para cualquiera, además de facilitar a los usuarios/as finales la búsqueda de información, ya que estos directorios también clasifican los ficheros RSS.

Para usarlos, es necesario **registrar** un **fichero RSS** en un **directorio RSS**. El proceso es similar a registrar un sitio en un motor de búsqueda. Si tienes una web con un feed, es recomendable incluirlo en un directorio, ya que esto ayudará a incrementar las visitas de la web. Además, es totalmente gratuito.

Cabe mencionar, que es posible que a la hora de sindicar un canal puedan surgir problemas debido al uso de tildes o ñ en los contenidos del canal, por lo que siempre es recomendable usar la codificación **UTF-8** y las entidades XML que les sustituyen, es decir:

- **&aacute**, en lugar de à, **&eacute**, en lugar de é, ...etc.
- **&lt** en lugar de <y **&gt** en lugar de >.

4.8 Agregación

Los clientes que quieran usar un servicio RSS o Atom tendrán que utilizar un **agregador de contenidos** para poder leer estos canales de información.

Un **agredador** o **lector de fuentes** es una aplicación software que permite suscribirse a fuentes RSS y Atom. El agregador avisa al usuario de que webs han incorporado contenido nuevo desde la última lectura y cual es este contenido nuevo. Para ello hay que indicar la dirección web de cada archivo fuente en el agregador, ya sea en formato RSS o Atom, para que pueda acceder al contenido.

Existen **varios tipos de agregadores**, siendo los más comunes los siguientes:

- **Agregadores RSS Web o En Línea**:

  Son aplicaciones que residen en determinados **sitios web** y que se ejecutan a través de la propia web. Los lectores RSS online cumplen la misma función que los programas que se instalan en el ordenador, aunque en estos todos se hace a través de la página web. Para ello, hay que **darse de alta** en la web que ofrece el servicio. A partir de ese momento, se puede acceder en cualquier momento al lector web introduciendo el nombre de usuario y contraseña. Son recomendables cuando el usuario/a no accede a internet siempre desde el mismo ordenador.

  Algunos ejemplos de estas aplicaciones son:

  - **Netvibes**: `https://validator.w3.org/feed/`
  - **Feedly**: `https://feedly.com/`
  - **Inoreader**: `https://www.inoreader.com/`

- **Agregadores RSS en Navegador Web o Gestor de Correo**:

  Son lectores de RSS que se instalan en un **navegador web** o en un **gestor de correo**. Antes los navegadores llevaban esta opción por defecto, aunque en la actualizad se le puede añadir esta funcionalidad mediante plugins. Algunos de los navegadores y gestores de correo más usados que tienen esta funcionalidad son **Mozilla Firefox**, **Outlook Express**, **Google Chrome** o **Mozilla Thunderbird**.

  Aquí tienes un articulo relacionado con el tema, en concreto con la eliminación de la opción por defecto en Firefox y donde se nos recomiendan diferentes extensiones.

- **Agregadores RSS de Escritorio**:

  Son aplicaciones que se instalan en el ordenador del usuario/a. Su uso es aconsejable para quienes acceden a internet siempre desde el mismo ordenador. Su interfaz suele ser parecida a la de los gestores de correo con un panel donde se agrupan suscripciones y otro donde se accede a la entradas individuales para su lectura.

  Algunos de estos programas son:

  - **Feedreader**: `http://feedreader.com/download`
  - **RSSReader**: `http://www.rssreader.com/download.htm`

- **Agregadores de Podcast o Podcasting**:

  Un **Podcast** consiste en la **distribución de archivos multimedia**, por normal general **vídeo** y **audio**, de larga duración, mediante un sistema de **redifusión RSS** y que presenta la posibilidad de suscribirse y usar programas de descarga para que los usuarios los escuchen y vean. Suele haber dos tipos principales:

  - **Podcast de Audio**: distribución de archivos de audio, en general en formato MP3 o AAC, con un tamaño pequeño y fáciles de manejar.
  - **Podcast de Video o Vodcast**: distribución de archivos de video que requieren conexiones de un gran ancho de banda, debido a su tamaño. En general suelen distribuirse en formato MP4 o M4V.

  Algunos de los agregadores de podcast más usados son **Spotify**, **Podimo** o **Google Podcast**, los cuales permiten no solo escucharlos o verlos en directo, sino también descargarlos.

Tema 5

Conversión y Adaptación de Documentos XML

En esta unidad vamos a estudiar diferentes técnicas de adaptación y conversión de documentos XML. Para llevar a cabo esto, vamos a usar principalmente dos tecnologías ligadas a XML, por una lado **XPath**, un que permite realizar la consulta de elementos en en un documento XML y **XSL**, un lenguaje que nos permitirá realizar diferentes tipos de transformaciones a un documento XML.

5.1 Técnicas de Transformación de Documentos XML

Una de las principales herramientas para realizar transformaciones en un documento XML es **Extensible Stylesheet Language** (XSL), un lenguaje que interpreta hojas de estilo. Una **hoja de estilo XSL** describe como debe mostrarse un documento XML, algo parecido a lo que hace un archivo **CSS** (Cascade Style Sheets) con los documentos HTML.

La especificación XSL define unas características y una sintaxis que se puede agrupar en tres partes:

- **XSL Transfornation Language** (XSLT): es un lenguaje para la transformación de documentos XML que nos permite realizar diferentes tipos de transformaciones.
- **XML Path Language** (XPath): un lenguaje de consulta de elementos XML.
- **XSL Formatting Object** (XSL-FO): un vocabulario XML para especificar la semántica del formato, es decir, indica donde debe aparecer cada elemento, con que espaciado, que color, etc...

Para realizar las transformaciones se utilizan unos programas denominados **procesadores XSL** o **XSL parsers**. A estos programas se les debe indicar los archivos de entrada XSL y XML. Si los archivos no tienen errores, se generará el archivo deseado en el formato indicado en las instrucciones XSL.

XSLT fue diseñado para usar realizar transformaciones usando el vocabulario XML especificado por XSL-FO, pero también puede usarse con independencia de XSL y dicho vocabulario. A nivel académico podemos realizar transformaciones de documentos XML a documentos HTML u otros XML de estructura diferente sin usar XSL-FO.

En esta unidad, usaremos **XSLT** y **XPath**, para evitar de esta forma la complejidad añadida por XSL-FO.

5.2 XML Path Language (XPath)

XML Path Language (XPath) es un lenguaje para la consulta de elementos en un documento XML. Diseñado originalmente para ser usado con **XSLT** (eXtensible Stylesheet Language for Transformations) y **XPointer** (XML Pointer Language), en versiones posteriores también es usado con **XQuery** como veremos en próximas unidades.

El consorcio W3C (The World Wide Web Consortium) aprobó su primera versión **XPath 1.0** en **noviembre de 1999**. XPath usa una sintaxis compacta y sin etiquetas, por lo que no se asemeja a XML. Su nombre surge por el uso de las rutas en la navegación a través de las estructuras jerárquicas dentro de un documento XML.

Posteriormente siguieron las **versiones**:

- **XPath 2.0** (Enero de 2007)
- **XPath 2.O Second Edition** (Diciembre 2010)
- **XPath 3.0** (Abril de 2014)
- **XPath 3.1** (Marzo de 2017)

Para obtener más información puedes ver la recomendación completa en la web de W3C.

5.2.1 Árbol de Nodos

XPath modela los documentos empleando un **árbol de nodos**.

Un **árbol** es una estructura abstracta muy empleada en informática que representa una serie de elementos, **nodos**, unidos por líneas o **vértice**. Además, entre dos nodos cualesquiera solo existe un camino único y ninguno de los caminos forma un bucle.

El nombre de esta estructura le viene porque tiene forma de copa de árbol invertida. Partiendo de un nodo raíz, van añadiéndose ramas que contienen los nodos hijo. A los nodos que **tienen** al menos **un hijo** se les llama **nodos padre**. A los nodos que **no tienen** ningún hijo se les llama **nodos hoja**. Los nodos que **comparten padre** se les llama **nodos hermanos**.

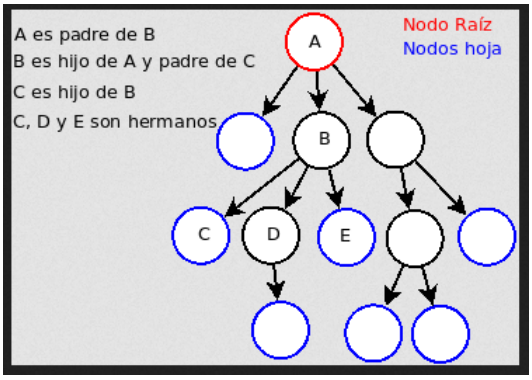


Figura 5.2.1.: Árbol de nodos

En **XPath** existen distintos **tipos de nodos** que recogen diferentes tipos de información en los documentos XML. Los **siete tipos distintos** de nodos que podemos encontrar en XPath son los siguientes:

- **Nodo raíz**: es el primer nodo del árbol y el único que no tiene padre. No hay que confundir el nodo raíz con el elemento raíz del documento XML. El nodo raíz tiene como hijos al elemento raíz y en su caso, los comentarios y las instrucciones de procesamiento que formen el prólogo del documento XML.
- **Nodo Elemento**: hay un nodo elemento por cada elemento XML en el documento. Todos los nodos elemento tienen un sólo padre que puede ser otro elemento o el nodo raíz. Los nodo elementos pueden tener un identificador único (ID), aunque para ello deben estar declarados como tipo ID en el documento DTD o en el XMl Schema.
- **Nodo atributo**: estos nodos almacenan los atributos de un documento XML. Un nodo atributo estará asociado a un único elemento que será el padre de este, pero desde el punto de vista del elemento no se considera a los nodo atributo como sus hijos. Los nodo atributo son siempre nodos hoja, es decir, no pueden tener nodos hijo.
- **Nodo texto o contenido**: estos nodos almacenan los valores alfanuméricos de los elementos en un documento XML. Estos nodos son nodos hoja, al igual que nodos atributo. Nótese que los valores de los nodos atributo no se almacenan en nodos texto, sino en los propios identificadores de dichos nodos.
- **Nodo comentario**: son los nodos que almacenan los comentarios de un documento XML. El nodo almacenará el contenido del comentario sin incluir las cadenas de inicio (<!–) ni fin (–>) del comentario.
- **Nodo espacio de nombres**: cada nodo elemento puede tener un conjunto asociado de nodos espacio de nombres, uno para cada uno de los distintos prefijos de espacios de nombres, incluyendo si si es el caso el espacio de nombres por defecto. Tienen un funcionamiento parecido a los nodos atributo. Un nodo espacio de nombres tiene un único elemento como nodo padre, pero desde el punto de vista del elemento estos nodos no son considerados nodos hijo. Estos nodos siempre serán nodos hoja.
- **Nodo instrucción de procesamiento**: hay un nodo por cada instrucción de procesamiento. No se incluye la cadena de terminación de la orden (?>).

Además se establece un **orden entre los nodos** que se corresponde con el orden en el que está escrito el documento, de modo que la raíz será el primer nodo y los nodos elemento aparecerán antes que sus hijos. Por tanto, el orden del documento establece un orden para los nodos según el orden de aparición de las etiquetas de apertura de dichos elementos en el documento XML. Los nodos atributos y los nodos espacio de nombres aparecen antes que los hijos del elemento, apareciendo los nodos espacio de nombres antes que los nodos atributo.

El nodo raíz y los nodos elemento tiene un lista ordenada de nodos hijo. Además, los nodos nunca comparten hijo, es decir, cada nodo tiene un único padre, excepto del nodo raíz que no tiene padre.

En el **siguiente ejemplo** de un documento XML que almacena una agenda podemos ver el árbol de nodos que crea la aplicaicón **BaseX**. Podemos observar a la izquierda el documento XML y a la derecha el árbol creado. Esta aplicación no resalta los diferentes nodos de ninguna forma especial, mostrando todos con su identificador.

En la siguiente figura podemos ver una captura del documento y su correspondiente árbol.

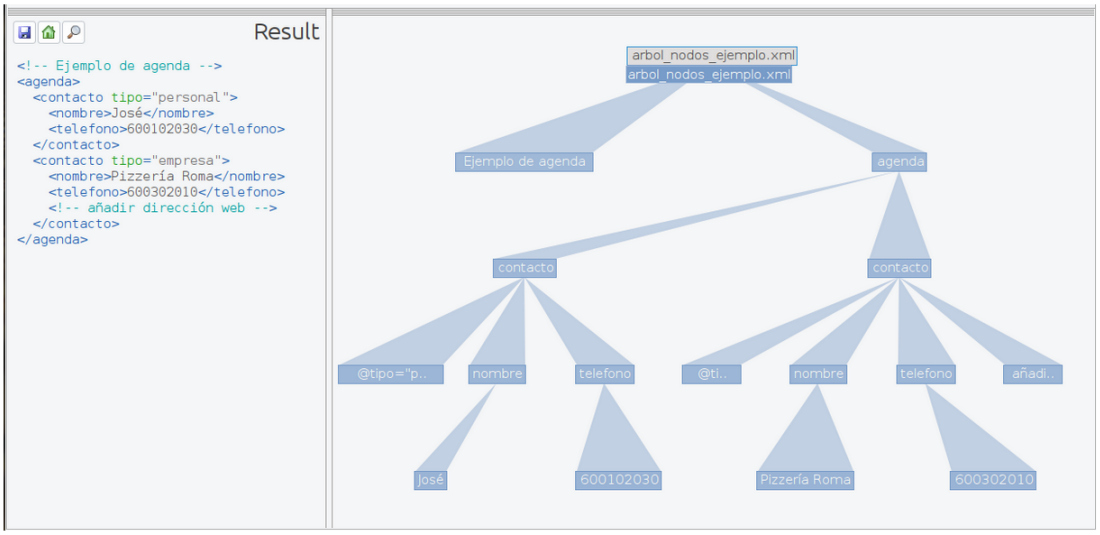


Figura 5.2.2.: Arbol de un documento XML

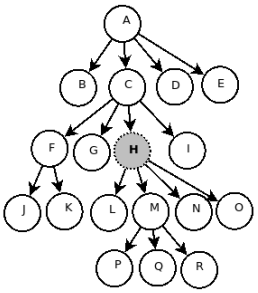
- El **nodo raíz** esta indicado con el nombre del documento **XML:arbol_nodos_ejemplo.xml**. Nótese que no se corresponde con el elemento raíz que en esta caso sería 'agenda'. Es importante **no confundir** el nodo raíz con el elemento raíz. El primero ser refiere al árbol de nodos y el segundo al documento XML.
- Los **nodos elementos** son los más abundantes: **agenda**, **contacto**, **nombre** y **teléfono**.
- Los **nodos texto** son los contenidos de los elementos **nombre** y **teléfono**, es decir: **Jóse**, **600102030**, **Pizzería Roma** y **600302010**. Estos nodos siempre serán nodos hoja.
- Los **nodos atributo** recogen el identificado y el valor. En nuestro caso, **tipo="personal"** y **tipo="empresa"**.
- También podemos ver en el ejemplo los **nodos comentario**, uno al inicio y otro dentro del segundo elemento **contacto**, con el contenido **Ejemplo de agenda y añadir dirección web**.

5.2.2 Relaciones entre Nodos

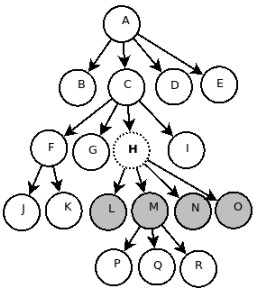
Cuando nos **posicionamos en un nodo** dentro de un árbol de nodos se establecen **distintas relaciones** con otros nodos del arbol. El nodo en el que nos posicionamos para establecer la relación se denomina **nodo contexto** o **nodo contextual**. Las distintas relaciones que podemos tener serán utilizadas después como ejes en los pasos de localización.

Las **relaciones** que se pueden establecer son la siguientes:

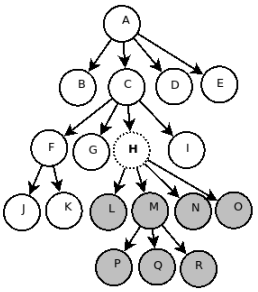
- **self**: contiene solo el nodo contexto.



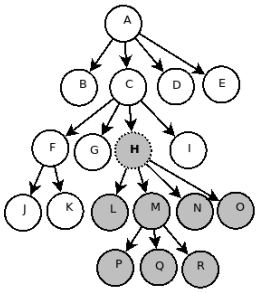
- **child**: contiene los hijos del nodo contexto.



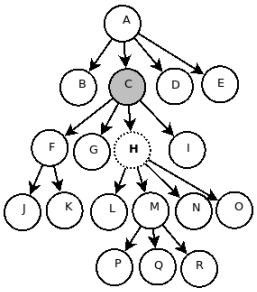
- **descendant**: contiene los descendientes del nodo contexto, es decir, los hijos y los hijos de éstos, etc.. Nunca contiene nodos atributo o espacio de nombres.



- **descendant-or-self**: contiene todos los nodos que contiene la relación descendant además del nodo contexto mismo.

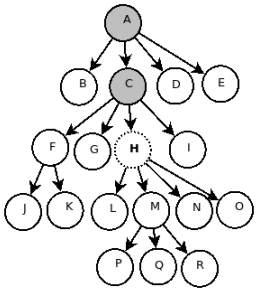


- **parent**: contiene el padre del nodo contexto, si lo hay.

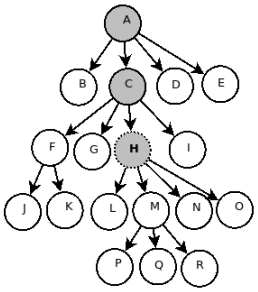


- **ancestor**: contiene los ancestros del nodo contexto, es decir, el padre del nodo contexto y el padre

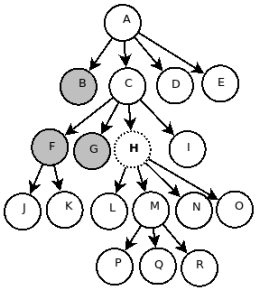
de este, etc.. Esta relación siempre incluirá al nodo raíz, salvo que el nodo contexto sea el mismo nodo raíz.



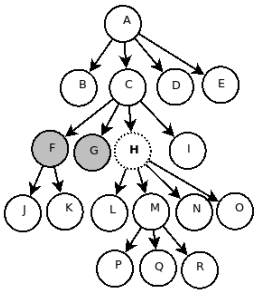
- **ancestor-or-self**: contiene los mismo nodos que la relación ancestor, pero además incluye al nodo contexto mismo.



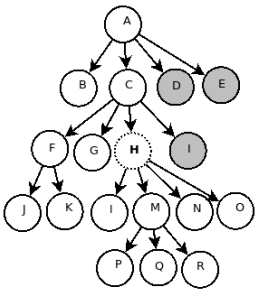
- **preceding**: contiene todos los nodos del mismo documento que el nodo contexto que están antes que él, según el orden de documento, excluyendo a los ancestros y los nodos atributos y espacio de nombres.



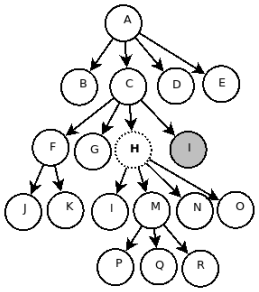
- **preceding-siblin**: contiene todos los hermanos precedentes del nodo contexto. Si el nodo contexto es un nodo atributo o espacio de nombres el eje preceding-sibling estará vacío.



- **following**: contiene todos los nodos del mismo documento que el nodo contexto que preceden a este según el orden del documento, excluyendo a los descendientes de este y los nodos atributo y espacio de nombres.



- **followinf-sibling**: contiene todos los elementos siguiente al nodo contexto. Si el nodo contexto es un nodo atributo o espacio de nombres, el eje following-sibling estará vacío.



No se han incluido **nodos atributo** ni **nodos espacios de nombres** en estos gráficos para simplificarlos un poco, pero las **relaciones** que podemos encontrar con estos nodos son las siguientes:

- **attribute**: contiene todos los nodos atributos del nodo contexto. Esta relación estará vacía si el elemento contexto no es un nodo elemento, ya que es el único que puede tener atributos.
- **namespace**: contiene todos los nodos de espacio de nombres del nodo contexto. Al igual que la relación attibute, ésta estará vacía si el nodo contexto no es un nodo elemento.

5.2.3 Sintaxis

La sintaxis de XPath no usa etiquetas, por lo que no se asemeja a XML. Como su utilidad es hacer búsquedas en el árbol de nodos, se utiliza una sintaxis similar a la que se usa en los árboles de directorios y archivos del sistema operativo.

Un ejemplo de sintaxis de XPath puede ser el siguiente '**camino de localización**' (location paths):

```
descendant-or-self::curso[position()=1]/child::grupo[position()=2]/child::alumno[last()]/
    child::nombre/child::node()
```

Aunque siempre que se pueda, se suele emplear su **versión simplificada**. El ejemplo anterior, quedaría de la siguiente forma en esta versión:

```
//curso[1]/grupo[2]/alumno[last()]/nombre/text()
```

Como podemos ver, nos hemos ahorrado muchos caracteres, y la consulta se ve mucho más limpia y fácil de entender. Por lo que es altamente recomendable usar la versión simplificada siempre que se pueda.

Además de las expresiones XPath, se pueden emplear **llamadas a funciones**, **operaciones matemáticas simples** y **operaciones lógicas**.

Cuando un programa **evalúa** un **camino de localización** devolverá el resultado en uno de los siguientes **tipos básicos**:

- Un **conjunto de nodos** (Node-Set): una colección desordenada de nodos sin duplicados. Nótese que este conjunto de nodos no puede ser vacío ni contener un único nodo.
- Un **valor lógico** (booleano): el valor verdadero o falso.
- Un **valor numérico**: un número en punto flotante.
- Una **cadena de caracteres**.

Las **palabras reservadas** de XPath son las siguientes:

- **Ejes**: ancestor, ancestor-or-self, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling y self.
- **Selectores de nodos**: node(), text(), comment() y procesing-instruction().
- **Operaciones lógicas**: and, or, not().
- **Operaciones matemáticas**: div, mod.

Además, hay un **conjunto de símbolos** con funcionales definidas en el lenguaje, a saber:

- **Agrupación** de **operaciones** con paréntesis: **()**
- **Predicados** con corchetes: **[]**
- Abreviación **nodo actual** con el punto: **.**
- Abreviación del **nodo padre** con dos puntos: **..**
- Abreviación del **atributo** con la arroba: **@**
- **Todos** los **tipos de nodos** con el asterisco: *****
- **Separado eje-selector** con los dos puntos: **:**
- **Separador** de los **pasos** de localización con la barra: **/**
- Abreviación del **paso** de localización *descendant-or-self::node()* con dos barras: **//**
- Referencia a **una variable** con el símbolo del dolar: **$**
- **Unión** de conjuntos con las barra vertical: **|**
- **Operaciones lógicas**: **=**, **!=**, **<**, **>**, **<=** y **>=**
- **Operaciones matemáticas**: **+**, **-** y *****

El lenguaje también usa:

- **Nombres cualificados** de los identificadores de XML.
- Los **nombres de las funciones** que se desean utilizar.
- **Nombres de las referencias a variables** anteponiendo el símbolo $.
- **Numeros y literales**, estos últimos entrecomillados con comillas simples o dobles. Pueden anidarse alternando el tipo de comillas.

A continuación, mostramos **un ejemplo** de como se realizarían varias consultas dado un documento XML. En la siguiente imagen, podemos ver el documento XML que se usará de ejemplo.

Figura 5.2.3.: XML de ejemplo para XPath

Partiendo de este documento XML, vamos a realizar la siguientes consultas:

1. Localizar al alumnado de la clase 2 de ESO A:

```
/child::colegio/child::curso[@nivel="2"]/child::grupo[@orden="A"]/child::alumno

<!-- Versión simplificada -->

/colegio/curso[@nivel="2"]/grupo[@orden="A"]/alumno
```

2. Localizar a todas las alumnas llamadas Ana:

```
descendant-or-self::alumno[nombre="Ana"]

<!-- Versión simplificada -->

//alumno[nombre="Ana"]
```

3. Nombre del alumno con la nota más alta:

```
descendant-or-self::alumno[child::nota_media=max(/descendant-or-self::nota_media)]/
    child::nombre

<!-- Versión simplificada -->

//alumno[nota_media=max(//nota_media)]/nombre
```

5.2.3.1 Caminos de Localización

Aunque los **caminos de localización** no son la construcción gramatical más general del lenguaje XPath, si es la **más importante**.

Todo camino de localización se **puede expresar** utilizando la sintaxis compleja, aunque algo más extensa. Existen también ciertas abreviaturas sintácticas que permiten expresar los casos más frecuentes de forma más concisa.

Siguiendo con el ejemplo anterior:

```
/descendant-or-self::curso[position()=1]/child::grupo[position()=2]
```

A estas expresiones se le llama **camino de localización** (location paths). Estos caminos están compuestos por trozos separados por barras (**/**). Cada uno de estos trozos se denominan **pasos de localización** (location steps).

A su vez, cada **pasado de localización** esta separado en dos partes por dos puntos dobles (**::**). A la **primera parte** se le denomina **eje** (axe) y a la **segunda parte** se le llama **selector de nodos** (node test).

En la siguiente imagen podemos ver de forma más gráfica las diferentes partes de una expresión de camino de localización.

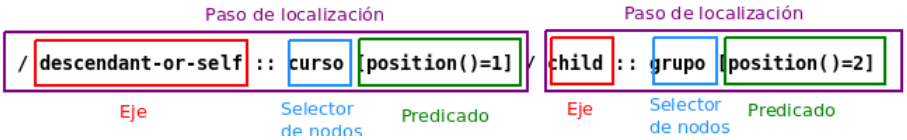


Figura 5.2.4.: Partes de un camino de localización

En algunos **pasos de localización** hay instrucciones entre corchetes ([]), después del selector de nodos. Son los conocidos como **predicados**.

Además, en los caminos de localización se pueden usar **llamadas a funciones**, **operaciones matemáticas** y **operaciones lógicas**, como por ejemplo:

```
sum(//nota_media) div count(//nota_media)

substring(//child::colegio/child::comment(), 12, 4)
```

Los caminos de localización pueden expresarse de forma **absoluta o relativa**:

- Un **camino de localización** es **absoluto** si comienza por el nodo raíz. Esto se puede reconocer fácilmente ya que el camino de localización comenzará con una barra (/). Por ejemplo:

```
/child::colegio/child::curso/child::grupo/attribute::*
```

- Si no comienza por el nodo raíz, estamos hablando de una **camino de localización relativo**, es decir, relativo al nodo de contexto que éste posicionado en un determinado lugar. En el siguiente ejemplo el nodo contexto estaría posicionado en el elemento grupo:

```
child::grupo/attribute::*
```

Además, podemos usar el **operador unión** (|) para unir el resultado de dos caminos de localización que devuelvan conjuntos de nodos, como por ejemplo:

```
descendant-or-self::apellidos | descendant-or-self::nota_media
```

5.2.3.2 Pasos de Localización

Como hemos visto en el punto anterior los caminos de localización pueden estar compuestos de **pasos de localización**. Cada paso irá refinando la búsqueda de la información que deseemos localizar. Los pasos de localización se separan unos de otros con una barra (/).

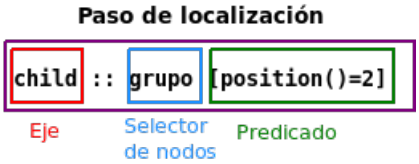


Figura 5.2.5.: Elementos de un paso de localización

Como vemos en la imagen, una paso de localización tiene **3 partes diferentes**:

- Un **eje**, que especifica la relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual.
- Un **selector de nodos**, que especifica el tipo de nodo de los nodos seleccionados por el paso de localización.
- Cero o más **predicados**, que usan expresiones lógicas para refinar aún más el conjunto de nodos seleccionados por el paso de localización.

La **sintaxis** de un paso de localización es, por tanto, el **nombre del eje** y el **selector de nodos**, separados por dos puntos dobles (**::**), seguidos de cero o más **predicados**, cada uno de los cuales va entre corchetes (**[]**). Por ejemplo:

```
child::grupo[position()=2]
```

Donde **child** es el nombre del eje, **grupo** es el selector de nodo y **[position()=2]** es un predicado.

El **conjunto de nodos seleccionados** por un paso de localización es el resultante de generar un conjunto de nodos inicial a partir del eje y el selector de nodos, para a continuación filtrar dicho conjunto por cada uno de los predicados definidos. El conjunto final de nodos es el conjunto seleccionado por el paso de localización.

5.2.3.3 Ejes

Los **ejes** se corresponden con las relaciones entre los nodos del árbol de nodos que hemos visto en puntos anteriores. Estos **trece** ejes son:

- **self**: nodo contexto.
- **child**: hijos del nodo contexto.
- **descendant**: descendientes del nodo contexto.
- **descendant-or-self**: nodo contexto y sus descendientes.
- **parent**: padre del nodo contexto.
- **ancestor**: ancestros del nodo contexto.
- **ancestor-of-self**: nodo contexto y ancestros.
- **preceding**: nodos anteriores.
- **preceding-sibling**: hermanos anteriores.
- **following**: nodos posteriores.
- **followinf-sibling**: nodos hermanos posteriores.
- **attribute**: nodos atributos.
- **namespace**: nodos de espacio de nombres. En XPath 2.0 se dejó de usar.

Hay que recordad que para seleccionar los atributos o espacios de nombres hay que utilizar la **forma explícita** de sus ejes correspondientes. Por ejemplo, en el eje **child** de un nodo elemento no contendrá sus atributos ni sus espacios de nombres.

5.2.3.4 Selectores de Nodos

Los **selectores de nodos** usan los diferentes tipos de nodos que vimos al estudiar el árbol de nodos. Cada uno tiene un tipo principal de nodos. Si un eje puede contener elementos, entonces el tipo principal del no es elemento, en otro caso, será el tipo de nodo que contenga el eje. Pudiendo ser:

- Para el eje **attribute**, el tipo de nodo principal son los **atributos**.
- Para el eje **namepspace**, el tipo de nodo principal son los **espacios de nombres**.
- Para los **demás ejes**, el tipo de nodo principal será el **elemento**.

El siguiente elemento en el paso de localización es el **selector de nodos**. XPath nos proporcionar los siguientes:

- **Nombre cualificado**: nombre del objeto que estamos buscando.
- **Todos** (*)
- **text()**
- **comment()**
- **processing-intruction()**
- **node()**

Estos son los selectores de nodos orinales, pero en versiones posteriores de XPath se agregaron algunos selectores más, que son los siguientes:

- **element()**
- **attribute()**
- **document-node()**

Un selector de nombres que sea un **nombre cualificado** (QName) es cierto si los tipos de nodo se corresponden y tiene un nombre igual al especificado por el nombre cualificado. Por ejemplo, **child::alumno** selecciona los elementos **alumno** hijos del nodo contexto. Si el nodo contexto no tiene ningún hijo alumno, seleccionará un conjunto de nodos vacíos. Otro ejemplo es, **attribute::nivel**, que seleccionaría el atributo **nivel** del nodo contexto. Si este no tuviera ningún atributo **nivel**, se seleccionará un conjunto de nodos vacíos igualmente.

Un selector de nodos * es cierto para cualquier nodo del tipo principal de nodo. Por ejemplo, **child::*** seleccionará todo elemento hijo del nodo contexto, y **attribute::*** seleccionaría todos los nodos atributos de éste.

El selector de nodos **text()** es cierto para cualquier nodo de texto. Por ejemplo, **child::text()** seleccionaría todos los nodos hijos del nodo contexto que son de tipo texto.

Análogamente, el selector de nodos **comment()** es cierto para cualquier nodo comentario, y el selector de nodos **processgin-instruction()** es cierto para cualquier instrucción de procesamiento. El selector **processing-instruction()** puede tener un argumento que sea literal; en este caso, será verdadera para cualquier instrucción de procesamiento que tenga un nombres igual al literal.

El selector **node()** es cierto para cualquier nodo de cualquier tipo.

A continuación, vamos a ver varios **ejemplos de uso** de los selectores de nodos.

- Selector nombre cualificado:

  /descendant-or-self::alumno/child::nombre

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
```

- Selector todos:

  /descendant-or-self::grupo/attribute::*

```
orden="A"
orden="B"
orden="A"
orden="B"
```

- Selector text():

  /descendant-or-self::apellidos/text()

```
Abad Álvarez
Benitez Bermúdez
Carmona Casado
Amate Antunez
Bellido Bravo
Cuesta Camacho
```

- Selector comment():

  /descendant-or-self::colegio/child::comment()

```
<!-- Datos del colegio -->
```

- Selector node():

  /descendant-or-self::grupo/child::node()

```
<alumno codigo="342">
    <nombre>Ana</nombre>
    <apellidos>Abad Álvarez</apellidos>
    <anno_nac>2018</anno_nac>
    <nota_media>6.5</nota_media>
</alumno>
```

5.2.3.5 Sintaxis Abreviada

La abreviatura más importante es que el selector **child::** puede ser omitido en los pasos de localización. A efectos prácticos, **child** es el **eje por defecto**. Por ejemplo, un camino de localización **/colegio/curso**es la abreviatura de **child::colegio/child::curso**.

Hay también una abreviatura para atributos. **attribute::** puede abreviarse usando el carácter @. Por ejemplo, un camino **grupo::[@orden="A"]** es la abreviatura de **child::grupo[attribute::orden="A']** y por tanto selecciona hijos **grupo** con un atributo **orden** con el valor igual a **A**.

La doble barra (**//**) es una abreviatura de **descendant-or-self::node()**. Por ejemplo, **//alumno**, sería la correspondiente abreviatura de **/descendant-or-self::node()/child::alumno** y por tanto seleccionará cualquier elemento **alumno** en el documento. Otro ejemplo sería **curso//alumno** que es la abreviatura de **child::curso/descendant-or-self::node()/child::alumno** y por tanto seleccionará todos los descendientes **alumno** de hijos **curso**.

Otra abreviatura es el uso del punto (**.**) en lugar de **self::node()**. Esto es particularmente útil en conjunción con **//**. Por ejemplo, el camino de localización **.//grupo** es abreviatura de **self::node()/descendant-or-self::node()/child::grupo** y por tanto seleccionará todos los descendientes elementos **grupo** del nodo contexto.

Análogamente, un paso de localización con dos puntos (**..**) es la abreviatura de **parent::node()**. Por ejemplo, **../apellidos** es las abreviatura de **parend::node/child::apellidos** y por tanto seleccionará los hijos **apellidos** del nodo padre del nodo contexto.

En resumen, las principales abreviaturas son:

- Si se omite el eje, estamos usando el eje **child** por defecto.
- Si usamos @ estamos usando el eje **attribute**.
- (**//**): paso de localización **descendant-or-self::node()**
- (**..**): paso de localización **parent::node()**
- (**.**): paso de localización **self::node()**.

Algunos ejemplos del uso de abreviaturas son las siguientes:

- **descendant-or-self::node()/child::nota_media** se abrevia **//nota_media**

```
<nota_media>6.5</nota_media>
<nota_media>8.25</nota_media>
<nota_media>4.75</nota_media>
<nota_media>7.5</nota_media>
<nota_media>2.25</nota_media>
```

- **child::colegio/child::curso/attribute::etapa** se abrevia **colegio/curso/@etapa)**

```
etapa="ESO"
etapa="ESO"
```

- **self::node()/descendant::telefono** se abrevia **./descendant::telefono**

```
<telefono>900102030</telefono>
```

5.2.3.6 Predicados

Un **predicado** filtra un conjunto de nodos respecto a un eje y un selector de nodos para introducir un nuevo conjunto de nodos. El resultado de la expresión contenida en el predicado sera de tipo booleano. Los predicados dentro de corchetes (**[]**). Un paso de localización puede tener **cero o más** predicados en cascada. Si existe más de un predicado, estos se evaluarán de **izquierda a derecha**.

Si estamos evaluando un valor numérico, este será cierto si el número es igual a la posición contextual y se convertirá en falso de otro modo. Así, un camino de localización **grupo[2]** es equivalente a **grupo[position()=2]**.

Las **operaciones** que podemos usar en los predicados son las siguientes:

- **Operadores de comparación**: =, !=, <, >, >=, <=
- **Operadores lógicos**: and, or
- **Operadores matemáticos**: +, -, *, div, mod

A continuación vemos **algunos ejemplos** del uso de predicados:

- Primer Alumno de 2 ESO A:

  //curso[@nivel="2"]/grupo[@orden=.^"]/alumno[1]

```
<alumno codigo="534">
    <nombre>Alejando</nombre>
    <apellidos>Álvarez Amate</apellidos>
    <anno_nac>2018</anno_nac>
    <nota_media>5.25</nota_media>
</alumno>
```

- Apellidos de alumnos con nota superior a 8:

  //alumno[nota_media>8]/apellidos

```
<apellidos>Benitez Bermúdez</apellidos>
<apellidos>Cuesta Camacho</apellidos>
<apellidos>Camacho Camacho</apellidos>
```

- Nombres de alumnos que no nacieron en el 2017 y tienen media superior a 7:
  - **//alumno[anno_nac!=2017][nota_media>7]/nombre**
  - **//alumno[anno_nac!=2017 and nota_media>7]/nombre**

```
<nombre>Beatriz</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carmen</nombre>
```

5.2.3.7 Funciones

XPath nos ofrece un conjunto de funciones para poder realizar diferentes operaciones y evaluar expresiones. Todas las implementaciones de XPath deben incluir las **funciones** que vamos a ver en esta sección y que podremos usar para **evaluar expresiones**.

Vamos a especificar cada función con un prototipo de función, que indica el tipo devuelto, el nombre de la función y el tipo de argumentos, que pueden ser número, string, booleano o node-set. Si un tipo de argumento es seguido de un símbolo de interrogación indica que el argumento es opcional, en otro caso, el argumento es obligatorio.

Esta **funciones** se pueden clasificar en cuatro grupos y son las siguientes:

- **Funciones de Conjuntos de Nodos**: son funciones que nos permiten trabaja con nodos y realizar diferentes operaciones con nodos. Estas funciones son las siguientes:
  - *number* **last()**: devuelve el número total de nodos del contexto seleccionado.
  - *number* **position()**: devuelve la posición del nodo actual dentro de los nodos del contexto seleccionado. Esta posición de enumera desde el 1, y no desde el 0 como en otros lenguajes de programación.
  - *número* **count(*nodo-set*)**: devuelve el número de nodos del conjunto de nodos pasado como argumento.
  - *string* **name(*nodo-set?*)**: devuelve una cadena de caracteres con el nombre cualificado de un nodo del conjunto de nodos que se le pasa como argumento, formado por una URI del espacio de nombres y el nombres local. Si no se le pasa ningún argumento, tomará el nodo contexto como argumento. Si no se declaran espacios de nombres devolverá el mismo resultado que la función local-name().
  - *string* **loca-name(*node-set?*)**: devuelve el nombres local, sin URI, de un nodo del conjunto de nodos pasado como argumento. Si no se le pasa ningún argumento tomará los nodos contexto como argumento. Si no se declaran espacios de nombres dará el mismo resultado que la función name().
  - *string* **namespace-uri(*nodo-set?*)**: devuelve el URI del espacio de nombres, sin el nombres local, de un nodo del conjunto de nodos que se le pasa como argumento. Si no se le pasa ningún parámetro, tomará los nodos contexto como argumentos.
  - *node-ser* **id(*object*)**: selecciona elementos mediante su identificador único. Los nodos deben ser declarados de tipo ID en DTD o el XML Schema correspondiente.

- **Funciones de Cadenas de Caracteres**: son funciones que nos permiten trabajar con cadenas de caracteres, permitiéndonos buscar subcadenas, hacer concatenaciones, etc... Esta funciones son:
  - *string* **string(*object?*)**: convierte un objeto en cadena de caracteres. Si no se especifica el objeto toma un conjunto de nodos con el nodo contexto como único miembro.
  - *string* **concat(*string*, *string*, *string**)**: devuelve la concatenación de las cadenas pasadas como argumento.
  - *boolean* **starts-with(*string*, string)**: devuelve verdadero si la primera cadena del argumento comienza con la segunda cadena del argumento. Devuelve falso en caso contrario.
  - *boolean* **contains(*string, string*)**: devuelve verdadero si la primera cadena de los argumentos contiene a la segunda y falso en caso contrario.
  - *string* **substring-before(*string*, *string*)**: devuelve las subcadena, de la primera cadena del argumento, que precede a la aparición de la cadena pasada como segundo argumento, o la cadena vacía si la cadena no aparece en ésta. Si el argumento es la cadena vacía, entonces devuelve una cadena vacía.
  - *string* **substring-after(*string*, *string*)**: devuelve la subcadena, de la primera cadena del argumento, que sigue a la aparición de la segunda cadena pasada como argumento o la cadena vacía en caso de que esta no aparezca en la primera cadena. Si el argumento es la cadena vacía, devolverá una cadena vacía.
  - *string* **substring(*string*, *number*, *number?*)**: devuelve la subcadena que comienza en la posición indicada por el segundo argumento y con la longitud indicada en el tercer argumento. Si no es especifica la longitud, devuelve la cadena que comienza en la posición especificada y continúa hasta el final de la cadena. Se considera que cada carácter tiene una posición numérica dentro de la cadena, comenzado a contar desde 1.
  - *number* **string-length(*string?*)**: devuelve el número de caracteres de la cadena indicada como argumento. Si se omite el argumento, toma por defecto el nodo contexto convertido en cadena de caracteres.
  - *string* **normalize-space(*string?*)**: devuelve la cadena argumento con el espacio en blanco normalizado, es decir, elimina los espacios que se encuentra al principio y final de la cadena, además de sustituir las secuencias de espacios en blanco por uno solo. Si se omite el argumento, toma por defecto el nodo contexto convertido en cadena de caracteres.
  - *string* **translate(*string*, *string*, *string*)**: devuelve la cadena del primer argumento con las apariciones indicadas en el segundo argumento sustituidas por las indicadas en el tercer argumento.
- **Funciones Lógicas**: nos permiten realizar diferentes operaciones booleanas sobre diferentes datos.
  - *boolean* **boolean(*object*)**: convierte el objecto argumento al tipo booleano.
  - ***boolean*** **not(*boolean*)**: devuelve verdadero si su argumento es falso y falso en caso contrario.
  - *boolean* **true()**: devuelve el valor true.
  - *boolean* **false()**: devuelve el valor false.

- *boolean* **lang(*string*)**: devuelve verdadero o falso dependiendo si el lenguaje del nodo contextual, tal como se especifica por el atributo *xml:lang*, es el mismo o un sublenguaje del especificado como argumento.

- **Funciones Matemáticas**: son funciones que nos permiten realizar diferentes operaciones matemáticas. Estas son:
  - *number* **number(*object?*)**: convierte el argumento al tipo numérico. Si se omite el argumento, toma por defecto un conjunto de nodos con el nodo contexto como único elemento.
  - *number* **sum(*node-set*)**: devuelve la suma, a lo largo de todos los nodos del conjunto de nodos del argumento, resultando de convertir los valores de cadenas de caracteres de los distintos nodos en números.
  - *number* **floor(*number*)**: devuelve el mayor número entero que sea igual o menor que el número pasado como argumento.
  - *number* **ceiling(*number*)**: devuelve el menor número entero que sea igual o mayor que el número pasado como argumento.
  - *number* **round(*number*)**: devuelve el número entero más cercano al pasado como argumento.

A continuación vamos a ver **algunos ejemplos** de las funciones que hemos visto:

- Número total de alumnos nacidos en 2018:

  count(//alumno[anno_nac=2018])

```
8
```

- Nombre de los primeros alumnos de cada grupo: **//alumno[1]/nombre**

```
<nombre>Ana</nombre>
<nombre>Ana</nombre>
<nombre>Alejando</nombre>
<nombre>Ana</nombre>
```

- Alumnos que su nombre comience por 'Al':

  //alumno[starts-with(nombre,."Al")]

```
<alumno codigo="534">
    <nombre>Alejando</nombre>
    <apellidos>Álvarez Amate</apellidos>
    <anno_nac>2018</anno_nac>
    <nota_media>5.25</nota_media>
</alumno>
```

- Normalización de una cadena:

  normalize-space(."Esto es una prueba ")

```
Esto es una prueba
```

- Alumnos que no se llaman 'Carmen':

  //alumno[not(nombre=Çarmen")]/nombre

```
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Alejando</nombre>
```

- Suma de las notas de los alumnos de 1 ESO A:

  sum(//curso[@nivel="1"]/grupo[@orden=.ᴬ"]/alumno/nota_media)

```
19.5
```

5.2.4 Estrategias de Uso

A la hora de crear sentencias de XPath hay ciertas estrategias que podemos seguir para obtener los resultados deseados de una forma más precisa. Ademas hay situaciones concretas que se resuelven de mejor manera usando un determinado patrón.

A continuación veremos las principales estrategias que nos ayudarán a crear consultas eficientes.

5.2.4.1 Posicionamiento de Predicados

Hay ciertas **estrategias** que podemos tener en cuenta a la hora de **posicionar los predicados** dentro de un paso de localización. Vamos a ver un ejemplo para que quede más claro como posicionar éstos.

Imaginemos que queremos saber el **grupo del alumno que se apellida 'Bellido Bravo'**.

Primero, buscamos en la estructura de nuestro documento XML la posición del atributo y del elemento que se indica en el enunciado. Para ello, podremos usar las siguientes sentencias:

```
colegio/curso/grupo/alumno/apellidos

colegio/curso/grupo/@orden
```

A continuación, sabemos que el resultado debe ser el número de orden, es decir, nuestro camino de localización debe acabar en ese atributo. Este será nuestro punto de partida. Probamos por lo tanto la consulta **colegio/curso/grupo/@orden)** antes de filtrar el apellido.

```
orden="A"
orden="B"
orden="A"
```

Para posicionar el predicado podemos utilizar varias estrategias. Una de ellas puede ser **poner el predicado** en el **primer elemento padre común**, que en nuestro ejemplo sería grupo. Es decir:

```
colegio/curso/grupo[alumno/apellidos="Bellido Bravo"]
```

Si probamos esta consulta, obtendremos el grupo que buscamos, pero nos devuelve todo el grupo, no solamente su orden.

Podemos **mezclar ambos caminos de localización** en el elemento común que habíamos determinado antes, obtendremos el resultado que queremos. Es decir, usaremos el camino de localización del atributo que queremos mostrar con el predicado que situaremos en el elemento común.

```
colegio/curso/grupo[alumno/apellidos="Bellido Bravo"]/@orden
```

Simplificando con doble barra en la parte inicial quedaría:

```
 //grupo[alumno/apellidos="Bellido Bravo"]/@orden
```

Cuando trabajamos con información en distintos niveles de nuestra estructura **es importante no utilizar** la **doble barra** (//) en mitad de los caminos de localización, ya que la **búsqueda se reiniciará** desde el **elemento raíz** y no con el predicado establecido. Por ejemplo, si sustituimos la expresión anterior por **//grupo[//apellidos="Bellido Bravo"]/@orden**, el resultado obtenido no será el deseado, como vemos a continuación.

```
orden="A"
orden="B"
orden="A"
orden="B
```

Vemos también la utilidad de utilizar el operador nodo actual (**.**) cuando necesitamos **bajar niveles** y del operador nodo padre (**..**) cuando necesitamos **subir niveles**.

5.2.4.2 Consultas Anidadas

En ocasiones necesitamos realizar **varias consultas anidadas**, es decir, que necesitemos el resultado de una consulta para realizar la segunda consulta.

Por ejemplo, si queremos saber la nota media de los alumnos/as que tienen una nota media menor que la del alumno que se apellida 'Abad Álvarez', tendremos que hacer **dos consultas**.

La **primera consulta** para saber la nota media de 'Abad Álvarez' y **otra** para saber los alumnos/as que tienen una nota media inferior a un número cualquiera, por ejemplo 5. Estas consultas podrían ser las siguientes:

```
//alumno[apellidos="Abad Álvarez"]/nota_media

Resultado:

<nota_media>6.5</nota_media>


//alumno[nota_media < 5]/nota_media

Resultado:

<nota_media>4.75</nota_media>
<nota_media>2.25</nota_media>
<nota_media>4</nota_media>
<nota_media>2.5</nota_media
```

Ahora tendremos que mezclar las dos consultas y en lugar del número 5 de antes poner la subconsulta que nos muestra la nota de 'Abad Ávarez', dando como resultado la siguiente consulta:

```
//alumno[number(nota_media) < //alumno[apellidos="Abad Álvarez"]/nota_media ]/nota_media

Resultado:

<nota_media>4.75</nota_media>
<nota_media>2.25</nota_media>
<nota_media>5.25</nota_media>
<nota_media>4</nota_media>
<nota_media>2.5</nota_media>
```

5.2.4.3 Evitar Elementos Repetidos

A veces puede que queramos mostrar todos los resultados de un determinado tipo pero sin que salgan resultados repetidos. Gracias al eje **preceding** podemos hacerlo de forma sencilla. Recordad que este eje no tiene abreviación por lo que su notación es la completa.

En primer lugar, mostramos **todos los nombres de los alumnos**:

```
//alumno/nombre

Resultado:

<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Ana</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Alejando</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carmen</nombre>
```

Si ahora quisiéramos que se nos mostrara el resultado sin los nombres duplicados, podemos añadir el eje **preceding** junto con la función **not()** obteniendo la consulta y resultados siguientes:

```
//alumno[not(nombre=preceding::nombre)]/nombre

Resultado:

<nombre>Ana</nombre>
<nombre>Beatriz</nombre>
<nombre>Carlos</nombre>
<nombre>Benito</nombre>
<nombre>Carmen</nombre>
<nombre>Alejando</nombre>
```

5.3 XSL Transformations (XSLT)

El **lenguaje de transformaciones XSL** (Extensible Stylesheet Language Transformations ó XSLT) es una recomendación aprobada por el consorcio W3C, cuya versión 1.0 fue aprobada el 16 de Noviembre de 1999. El mismo día que la recomendación XPath 1.0.

Gracias a este lenguaje, los procesadores XSLT pueden transformar un documento XML en otros documentos XML, HTML o de texto plano con estructuras y contenido diferentes al original. Esta transformación se realiza gracias al uso de **hojas de estilo** (**xsl:stylesheet**). El lenguaje XSLT define la sintaxis y semántica de estas hojas de estilo.

Una **transformación** expresada en XSLT describe reglas para transformar un árbol de nodos origen en un árbol de nodos resultado. La transformación se logra asociando patrones definidos en la plantilla (**xsl:template**). Un **patrón**, expresado mediante XPath, se compara con los elementos de origen. Si cumple con alguna de las reglas de la plantilla, éstos pasan a formar parte del árbol de nodo resultado.

Es importante resaltar que el **árbol de nodos resultado** es distinto del árbol de nodos origen. Además la estructura del árbol de nodos resultado puede ser completamente diferentes al de origen. Al construir el árbol resultado los elementos del árbol origen se pueden reordenar, filtrar y agrupar en una estructura arbitraria.

5.3.1 Hojas de Estilo XSLT

Para crear el árbol de nodos resultado debemos crear una **hoja de estilo**, en la que podremos usar **diferentes tipos de elementos**, siendo estos:

- **Elementos del espacio de nombres** asociados a la URI **http://www.w3.org/1999/XSL/Transform**. Se suele emplear el prefijo **xsl** y son las instrucciones que usa el el lenguaje XSLT. Algunos ejemplos son: **xsl:stylesheet**, **xsl:template**, **xsl:value-of**, **xsl:for-each**, etc...
- **Elementos de extensión** que se utilizan como mecanismos implementados por diferentes desarrolladores para aportar funcionalidades extra. Cada desarrollador determinará su uso. No los estudiaremos en esta unidad.
- **Elementos de resultado literal** (LRE), que son elementos que se añaden al árbol de nodos resultado y que no pertenecen al espacio de nombres xsl. Por ejemplo, cuando incluimos elementos HTML o texto entre la información obtenida del documento XML.

El **lenguaje XSLT no especifica** como se **asocia** una hoja de estilo con un documento XML. Para esto, se usa un mecanismo propio de XML, las **instrucciones de procesamiento**. Debe tenerse en cuenta que algunos navegadores, por seguridad, bloquean estas instrucciones de procesamiento. Un ejemplo de documento XML asociado con una hoja de estilo lo podemos ver en la siguiente figura.

Figura 5.3.1.: Documento XML asociado con una hoja de estilo

En este ejemplo, se utiliza la instrucción de procesamiento **xml-stylesheet**, indicando en el atributo **type** el tipo de archivo con la nomenclatura MIME (**text/xml** o **text/xsl**) y en el atributo **ref**, la ubicación de la hoja de estilo.

```
<?xml-stylesheet type="text/xsl" href="colegio.xsl"?>
```

Esta instrucción de procesamiento no es obligatoria, ya que al ejecutar el procesador XSLT se le puede indicar como parámetros los documentos XML y XSL sobre los que queremos realizar la transformación.

Para **crear una hoja de estilos** usaremos la instrucción **xsl:stylesheet**, necesitando declarar el espacio de nombres de XSLT que se corresponde con la URI **http://www.w3.org/1999/XSL/Transform**. Se suele asociar este espacio de nombres con el **prefijo xsl:**. Además, esta instrucción tiene una atributo obligatorio, **version**, en el que indicaremos la versión del lenguaje XSLT en la que vamos a codificar nuestra hoja de estilos.

Normalmente, cada hoja de estilos se creará en un **archivo de texto plano** con las extensión **.xsl**. Como dijimos anteriormente, el lenguaje XSLT es derivado de XML, por lo que será necesario incluir en la primera línea el prólogo XML. Posteriormente declararemos nuestra hoja de estilos como vemos en el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
...
</xsl:stylesheet>
```

5.3.1.1 Simplificación de Hojas de Estilo

XSLT permite una **sintaxis simplificada** para las hojas de estilo que consten con una plantilla para el nodo raíz. La sintaxis simplificada permite omitir el elemento **xsl:stylesheet** cuando se trate de un**elemento de contenido literal**. Además, en vez de declarar una plantilla **xsl:template**, esta se sustituirá directamente por dicho elemento de contenido literal. Desde este contenido literal se podrá acceder a las instrucciones del espacio de nombres **xsl:** como si se hubiera seleccionado como patrón el nodo raíz del árbol de nodos origen.

En este caso de simplificación, el elemento literal debe tener el atributo **xsl:version**. Para poder usar este atributo, así como las demás instrucciones del espacio de nombres **xsl:** se deberá añadir dicho espacio de nombres al citado elemento literal. Este elemento que actúa como plantilla no podrá contener instrucciones de nivel superior.

En el siguiente ejemplo, tenemos el documento **colegio.xml**, al que vamos a realizar una transformación.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
               <title>Colegio Cervantes</title>
            </head>
            <body>
               <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
               <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
               <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Si simplificamos este documento, el resultado sería el siguiente:

```
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <head>
        <title>Colegio Cervantes</title>
    </head>
    <body>
        <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
        <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
        <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
    </body>
</html>
```

En esta simplificación el **elemento de contenido literal** sería el elemento **html**, que contiene la declaración del espacio de nombres **xslnm** del lenguaje XSLT y el atributo obligatorio **version**. Posteriormente, haciendo uso del prefijo **xsl:**, podemos utilizar instrucciones que no sean de nivel superior como **xls:value-of**.

5.3.2 Elementos XSLT de Nivel Superior

Si consideramos un documento XSLT como un documento XML, el elemento raíz será **xsl:stylesheet** ó **xsl:transform**. Los **elementos de nivel superior** son aquellos del espacio de nombres **xsl:** que son hijos directos de alguno de estos dos elementos. Tiene una tratamiento especial ya que su ámbito de aplicación es toda la hoja de estilos que estamos declarando.

Los elementos de nivel superior **más utilizados** son:

- **xsl:template**
- **xsl:variable** y **xsl:param**
- **xsl:output**

También tenemos otros elementos de nivel superior que aunque, son menos utilizados, es conveniente que los conozcamos ya que cada uno tiene una utilizada muy específica. Estos son:

- **xsl:import** y **xsl:include**: sirven para añadir contenido a las hojas de estilo que estamos desarrollando. Los contenidos añadidos mediante **xsl:import** tiene una menor preferencia que los desarrollados en nuestra hoja de estilo. La preferencia de los contenidos añadidos con **xsl:include** es la misma que los desarrollados en nuestra hoja de estilos. Ambos elementos tiene una atributo **href** que indica donde se encuentran los contenidos a añadir.
- **xsl:strip-space** y **xsl:preserve-space**: sirve para indicar al procesador XSL como deber actuar al normalizar los espacios en blanco.
- **xsl:key**: sirve para declarar un elemento como clave de forma que pueda emplearse como los tipos de dato ID, IDREF y IDREFS. Declara una clave con nombres que se puede usar en cualquier lado de la hoja de estilo con la función **key()**.
- **xls:decimal-format**: define el formato que se empleará para convertir los números en cadenas de caracteres.
- **xsl:namespace-alias**: sirve para definir un prefijo alternativo para un espacio de nombres.
- **xsl:attribute-set**: se utiliza para crear grupos de atributos que se pueden reutilizar en los elementos **xsl:element** y **xsl:param**.

Estos elementos de nivel superior pueden aparecer varias veces o no aparecer en el documento. El orden de aparición no es relevante salvo para **xsl:import** que debe aparecer en primer lugar. Además, los elementos de nivel superior no pueden usarse dentro de otros elementos excepto **xsl:variable** y **xsl:param**.

5.3.2.1 xsl:template

Las plantillas (**xsl:template**) constan de dos partes: **un patrón**, expresado mediante XPath en el atributo **match**, que se compara con los nodos del árbol origen, y **unas instrucciones** que se aplicarán a los nodos seleccionados.

Con la aplicación de estas instrucciones y la incorporación de los elementos de contenido literal, podremos ir construyendo los nodos que necesitemos para el árbol de nodos resultado. Esto permite que la hoja de estilo sea aplicable a distintos documentos XML siempre que éstos tengan una misma estructura.

El **atributo mathc** es obligatorio a no ser que existe el atributo **name**. El atributo **name** sirve para hacer referencia a una plantilla mediante la instrucción **xsl:call-template**. El valor del atributo **match** no puede incluir el uso de variables.

En una **hoja de estilo** podemos tener **más de una plantilla**. La ejecución de las plantillas tiene una serie de abreviaciones que si no tenemos en cuenta puede llegar a confundir. Cada plantilla se puede hacer coincidir con un determinado nodo de nuestra estructura. Algunos ejemplos de **constricciones plantilla** son los siguientes:

- **Plantilla vacía**: si tenemos una plantilla que se corresponde con un nodo de nuestra estructura y es plantilla no tiene contenido entonces al recorrer los nodos no mostrará nada.
- **Nodos sin plantillas asociadas**: si en un determinado nodo de nuestra construcción no creamos plantilla, por defecto, ese nodo mostrará el contenido textual de ese nodo determinado, pero no mostrará el valor de sus atributos.

En el siguiente ejemplo tenemos una plantilla asociado a los nodos de tipo 'alumno', pero es plantilla esta vacía, por lo que al recorres estos nodos mostrará lo que contiene la plantilla, es decir, nada. Además, en nuestra estructura, tenemos otros nodos que no tienen correspondencia con el atributo **match** de ninguna plantilla, por lo que mostrará el contenido textual de esos nodos sin plantilla.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml"/>
    <xsl:template match="//alumno">
        <!-- No hacer nada -->
    </xsl:template>
    </xsl:stylesheet>
```

El resultado de esta transformación será:

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

Cervantes
900102030
```

En ocasiones cuando tenemos **más de una plantilla** vemos que **no funciona como esperábamos**. Por ejemplo, si tenemos una plantilla para 'alumno' que nos muestra el 'nombre' y otra plantilla para 'apellidos' que nos muestra el propio elemento '.', vemos que no funciona correctamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml"/>

    <xsl:template match="//alumno">
        <xsl:value-of select="nombre"/>
    </xsl:template>

    <xsl:template match="//apellidos">
        <xsl:value-of select="."/>
    </xsl:template>
</xsl:stylesheet>
```

El resultado, no es el que esperamos, como vemos a continuación:

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

Cervantes
900102030

Ana
Beatriz
Carlos

Ana
Benito
Carmen
```

EL **motivo** de este comportamiento es el **recorrido del árbol origen**. Cuando recorremos en la primera plantilla los elementos 'alumno' podemos mostrar perfectamente su nombre. Pasamos al siguiente

'alumno' y volvemos a mostrar el 'nombre', y así hasta el último elemento 'alumno'. Cuando vamos a aplicar la segunda plantilla ya hemos **terminado de recorrer** todo el árbol de nodos origen y no podemos volver atrás. Para que esto funcione, debemos usar la instrucción **xsl:apply-templates**.

Sabiendo esto, entenderemos porque el siguiente ejemplo, si mostraría correctamente ambos valores, nombre y apellidos, ya que los recorre cuando nos encontramos en el nodo alumno.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml"/>

    <xsl:template match="//alumno">
        <xsl:value-of select="nombre"/>
        <xsl:value-of select="apellido"/>
    </xsl:template>

</xsl:stylesheet>
```

Una **forma habitual de simplificar** el **uso de plantillas** es usar una única plantilla que accede al elemento raíz. Dentro de esta única plantilla podemos utilizar XPath para seleccionar las partes del documento XML que queramos e instrucciones de recorrido, como **xsl:for-each** y selección **xsl:if** para construir la estructura que deseemos. Por ejemplo, la siguiente estructura con llamadas anidadas mediante el uso de plantillas:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="text" />
    <xsl:strip-space elements="*"/>
    <xsl:template match="/">
        <xsl:apply-templates select="//alumno"/>
    </xsl:template>
    <xsl:template match="//alumno">
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="//anno_nac">
        <xsl:text>privado</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

Se puede simplificar a una sola plantilla con expresiones XPath e instrucciones **xsl:for-each** y **xsl:if**, como vemos a continuación.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="text" />
        <xsl:template match="/">
            <xsl:for-each select="//alumno">
                <xsl:value-of select="nombre"/>
                <xsl:value-of select="apellidos"/>
                <xsl:text>privado</xsl:text>
                <xsl:value-of select="nota_media"/>
            </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```

El resultado de ambas hojas de estilo es el mismo:

```
AnaAbad Álvarezprivado6.5BeatrizBenitez Bermúdezprivado8.25 ...
```

5.3.2.2 xsl:variable y xsl:param

Una **variable** es un nombre que se vincula con un valor. El valor de la variable puede ser de distintos tipos que devuelven las expresiones y funciones XPath. Hay dos elementos que utilizan esta vinculación de valores, **xsl:variable** y **xsl:param**.

Ambos elementos tienen un atributo obligatorio **name** en el cual hay que especificar el nombre de la variable, en el momento de utilizar su contenido, con un símbolo **$** delante. El otro atributo de estos elementos es **select**, que si se especifica en una expresión XPath almacenará su valor en una variables.

La **diferencia entre ambos** elementos es que el valor de **xsl:param** es un valor predeterminado que se usa normalmente pero que puede ser sobreescrito cuandoa se invoca una plantilla o una hoja de estilo mediante el elemento **xsl:with-param**, que tenga el mismo nombre (**name**) pero un valor distinto al predeterminado.

A continuación vemos unos ejemplos del uso de estos elementos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="text"/>
    <xsl:template match="/">

        <xsl:variable name="mensaje" >Hola mundo</xsl:variable>
        <xsl:value-of select="$mensaje" />
        <xsl:text>&#10;</xsl:text>
        <xsl:variable name="nombre_cole" select="colegio/nombre" />
        <xsl:value-of select="$nombre_cole"/>

    </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="text"/>
    <xsl:template match="/">

        <xsl:param name="mensaje" >Hola mundo</xsl:param>
        <xsl:value-of select="$mensaje" />
        <xsl:text>&#10;</xsl:text>
        <xsl:param name="nombre_cole" select="colegio/nombre" />
        <xsl:value-of select="$nombre_cole"/>

    </xsl:template>
</xsl:stylesheet>
```

5.3.2.3 xsl:output

Con esta instrucción especificamos el **formato de salida deseado**. Esta instrucción se puede omitir ya que el procesador XSLT devolverá el resultado como secuencias de caracteres. El uso de este elemento nos facilita poder especificar características de salida. Los **atributos más usados** son:

- **method**: especifica el formato de salida. Los valores que puede tomar este atributo son: **xml**, **html** y **text**,

- **version**: especifica la versión del método de salida.
- **indent**: especifica si la salida tiene que ser indentada o no. Los valores que puede tomar son: **yes** y **no**.
- **encoding**: para especificar la codificación.
- **standalone**: especifica si el documento tendrá relación con otros documentos. Los posibles valores que puede tomar son: **yes** y **no**.
- Otros atributos menos usados son: **media-type**, **doctype-system**, **doctype-public**, **omit-xml-declaration** y **cdata-section-elements**

```
xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html" version="5.0" encoding="UTF-8" indent="yes" />
....
</xsl:stylesheet>
```

5.3.3 Otros Elementos de XSLT

Además de los elementos que hemos visto en la sección anterior, que son los más usados, XSLT tiene otros elementos que nos permiten realizar diferentes acciones sobre el documento y que podemos clasificar en tres grupos principales:

- Intrucciones de manipulación de plantillas:
  - **xsl:apply-templates**
  - **xsl:call-template**
  - **xsl:with-param**
- Intrucciones de control de flujo:
  - **xsl:for-each**
  - **xsl:if**
  - **xsl:choose**
  - **xsl:when**
  - **xsl:otherwise**
  - **xsl:sort**
- Intrucciones de de salida:
  - **xsl:value-of**
  - **xsl:number**
  - **xsl:element**
  - **xsl:attribute**
  - **xsl:text**
  - **xsl:comment**
  - **xsl:processing-instruction**

Todas estas instrucciones las veremos con más detalle en los siguientes puntos, pero hay otras, que aunque no las veremos en profundidad, si conviene que sepamos de su existencia, que son las siguientes:

- **xsl:copy**

  Con esta instrucción podemos añadir el nodo actual al árbol de nodos destino. Al copiar un nodo se copian sus nodos espacios de nombres pero no sus atributos ni sus elementos hijos. Este elemento puede usar el atributo **use-attribute-sets** para añadir conjuntos de atributos declarados anteriormente. Solo se puede usar esta instrucción con nodos que puedan tener atributos o elementos hijos, es decir, el nodo raíz y nodos elementos.

- **xsl:copy-of**

  Con esta instrucción podemos añadir un fragmento completo del árbol de nodos al árbol de destino. No es necesario convertir ese fragmento a cadenas de caracteres. Tiene una **atributo obligatorio** llamado **select** donde debe indicarse, mediante una expresión XPath, el fragmento de árbol a copiar. Cuando se copia un elemento se copia el propio elemento y también sus nodos atributo, sus nodos espacio de nombres y sus elementos hijo.

- **xsl:apply-imports**

  Sirve para indicar cuando realizar una importación para sobreescribir una regla de plantillas.

- **xsl:message**

  Se utiliza para enviar mensajes al procesador XSLT. Tiene un atributo, **terminate**, que puede tomar los valores **yes** y **no**, y que se utiliza para indicar al procesador XSLT que interrumpa el procesamiento de la hoja de estilos.

- **xsl:fallback**

  Su función es establecer una secuencia ordenada de ejecución.

5.3.3.1 Instrucción de Manipulación de Plantillas

Esta funciones se emplean para realizar diferentes operaciones con las plantillas. En concreto, en este grupo las instrucciones más destacada es la siguiente:

- l**xsl:apply-templates**

  Como vimos con la instrucción **xsl:template**, cuando comparamos el árbol de nodos origen con las expresiones **match**, por defecto, no se continúa buscando más plantillas recorriendo los elementos hijo. Es decir, se pasa al siguiente elemento hermano que coincide con la expresión **match**.

  Con la instrucción **xsl:apply-template** podemos indicar si queremos que se siga con la búsqueda de plantillas en los elementos hijo. Esta instrucción tiene una atributo **select** en el que podemos indicar, mediante una expresión XPath, el conjunto de nodos hijos que queremos que se siga recorriendo y buscando plantillas que se correspondan con ellos. Si no estamos usando el atributo **select** estaremos indicando que se recorran todos los elementos hijo en busca de plantilla.

  Como contenido de la instrucción **xsl:apply-template** puede aparecer la función **sort()**, que nos ordenará los resultados antes de aplicar la plantilla que corresponda. También podemos encontrar la función **xsl:with-param** que nos permite pasar valores a la plantilla. Estos valores sobreescribirán los valores pode defecto indicados por la instrucción **xsl:param**:

En el **siguiente ejemplo**, vemos el uso de **xsl:apply-template**. Podemos ver como se van invocando los elementos desde el elemento raíz hasta el elemento hijo. En unos casos usando el atributo **select** y en otros no. En la segunda plantilla, por ejemplo, no se usa **select**, por lo que se aplica a todos los elemento hijo (**nombre**, **apellidos**,...). Estos elementos, al no tener plantilla por defecto, mostrarán su contenido textual, excepto el elemento **anno_nac** que si tiene una plantilla que especifica que se muestre el texto **privado** en vez de su contenido textual.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="text" />
    <xsl:strip-space elements="*"/>

    <xsl:template match="/">
        <xsl:apply-templates select="//alumno"/>
    </xsl:template>

    <xsl:template match="//alumno">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="//anno_nac">
        <xsl:text>privado</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

- **xsl:call-template**

  Se usa para invocar una plantilla que esta declarada con un atributo **name**.

- **xsl:with-param**

  Permite pasar valores a una plantilla que sobreescribirán los indicados por el elemento **xsl:param** que tenga el mismo atributo **name**.

5.3.3.2 Instrucciones de Control

En este grupo nos encontramos diferentes instrucciones que nos ayudarán a controlar el flujo de ejecución de las diferentes instrucciones que usemos, permitiéndonos recorres nodos o seleccionar unos o otros en función de una condición.

Las **principales instrucciones de control** son las siguientes:

- **xsl:for-each**

  Con esta función podemos crear un **bloque repetitivo** que **recorra de uno en uno todos los nodos** indicados mediante una expresión XPath en el atributo **select**. Dicho atributo es obligatorio. El valor indicado deben ser un conjunto de nodos. El recorrido de estos nodos se hace en el orden en el que son procesados, salvo en caso de que se utilice la función **sort()**.

- **xsl:if**

  Esta instrucción se utiliza para crear **bloques condicionales**, de modo que **solo ejecutemos una parte** de la plantilla si se **cumple una determinada condición**. Esta condición debe especificarse en el atributo **test**. La expresión de la condición se evaluará y se convertirá al tipo **boolean**. Si el resultado es **true**, entonces se ejecutar el contenido del elemento **xsl:if**. Si el resultado es **false** se pasará a la siguiente instrucción después del elemento **xsl:if**.

En el siguiente ejemplo, vamos a realizar una transformación en la que vamos a ordenar los alumnos por orden de apellido descendente y de cada uno de ellos mostraremos su nombres y su fecha de nacimiento o su nota media dependiendo de si su posición es par o impar.

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:for-each select="//alumno">
            <xsl:sort select="apellidos" order="descending"/>
            <li>
              <xsl:if test="((position() mod 2)=1)">
                <xsl:value-of select="nombre"/>
                <xsl:text>. Fecha de nacimiento: </xsl:text>
                <xsl:value-of select="anno_nac"/>
              </xsl:if>
              <xsl:if test="((position() mod 2)=0)">
                <xsl:value-of select="nombre"/>
                <xsl:text>. Nota media: </xsl:text>
                <xsl:value-of select="nota_media"/>
              </xsl:if>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Esta misma transformación también podría realizarse sustituyendo **xsl:for-each** por una llamada recursiva a otra plantilla mediante **xsl:apply-template**:

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:apply-templates select="//alumno">
            <xsl:sort select="apellidos" order="descending"/>
          </xsl:apply-templates>
        </ol>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="alumno">
    <li>
      <xsl:if test="((position() mod 2)=1)">
        <xsl:value-of select="nombre"/>
        <xsl:text>. Fecha de nacimiento: </xsl:text>
        <xsl:value-of select="anno_nac"/>
      </xsl:if>
      <xsl:if test="((position() mod 2)=0)">
        <xsl:value-of select="nombre"/>
        <xsl:text>. Nota media: </xsl:text>
        <xsl:value-of select="nota_media"/>
      </xsl:if>
    </li>
  </xsl:template>

</xsl:stylesheet>
```

El resultado en ambo casos sería el mismo, el que se muestra a continuación:

```
<html>
    <body>
        <ol>
            <li>Carmen. Fecha de nacimiento: 2017</li>
            <li>Carmen. Nota media: 4</li>
            <li>Carlos. Fecha de nacimiento: 2017</li>
            <li>Carmen. Nota media: 10</li>
            <li>Beatriz. Fecha de nacimiento: 2018</li>
            <li>Beatriz. Nota media: 8.25</li>
            <li>Benito. Fecha de nacimiento: 2018</li>
            <li>Benito. Nota media: 2.25</li>
            <li>Ana. Fecha de nacimiento: 2017</li>
            <li>Ana. Nota media: 7.5</li>
            <li>Alejando. Fecha de nacimiento: 2018</li>
            <li>Ana. Nota media: 6.5</li>
        </ol>
    </body>
</html>
```

- **xsl:choose**, **xsl:when** y **xsl:otherwise**

  Esta instrucciones nos permiten construir bloques condicionales, como la instrucción **xsl:if**, con al diferencia de que nos permite evaluar más de un caso posible.

  Consiste en una secuencia de elementos **xsl:when** seguidos de elementos **xsl:otherwise** opcionales. Cada elemento **xsl:when** tiene un único atributo **test**, que especifica una expresión. Cuando se procesa un elemento **xsl:choose**, cada uno de los elementos **xsl:when** es evaluado en orden. Se ejecutará el primer y solo el primer elemento **xsl:when** que haya devuelto **true**. Si no hay elementos **xsl:when** con resultado **true** entonces se ejecutará el elemento **xsl:otherwise**. Si este último no existe y no hay ningún elemento **xsl:when** evaluado a **true**, no se ejecutará nada.

- **xsl:sort**

  Con esta instrucción podemos ordenar los elementos seleccionados por las instrucciones **xsl:apply-template** o **xsl:for-each**. Debemos agregar **xsl:sort** como elemento hijo de alguna de estas instrucciones. Podemos agregar más de una elemento **xsl:sort**, en cuyo caso, la ordenación se realizará en el orden de aparición. Si **xsl:sort** se aplica a un bucle **xsl:for-each** deberá aparecer como primer hijo de este elemento. En el atributo **select** se indicará una expresión que devuelva una cadena de caracteres que servirá como clave de ordenación del conjunto de nodos seleccionados. El valor por defecto de este atributo es el nodo actual (**.**)

  Otros atributos importantes son:

  - **order**: especifica el sentido del orden, pudiendo tomar los valores **ascending** o **descending**.
  - **lang**: especifica el lenguaje de la clave de ordenación.
  - **data-type**: especifica el tipo de dato. Puedo tomar los valores **text**, **number** o un **nombre cualificado**.
  - **case-order**: especifica casos especiales. Puede tomar los valores **upper-first**, **lower-first** para los **data-type="text"**.

  En el siguiente ejemplo vamos a realizar una transformación que muestre una tabla con los alumnos ordenados por apellido con las filas de distintos colores según las notas medias obtenidas. Recordando un poco de HTML+CSS, en este ejemplo queremos asignar dinámicamente con un **xsl:choose** el valor del atributo **class** del elemento **tr** para dar uno de los tres estilos creados.

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <head>
        <style>
          .aprobado {color:green;}
          .suspenso {color:red;}
          .sobresaliente {color:blue}
        </style>
      </head>

      <body>
        <table border="0">
          <thead style="background-color:gray; color:white;">
            <th>NOMBRE</th>
            <th>APELLIDOS</th>
            <th>NOTA MEDIA</th>
          </thead>
          <tbody style="background-color:white;">
            <xsl:for-each select="//alumno">
              <xsl:sort select="apellidos"/>
              <xsl:element name="tr">
                <xsl:attribute name="class">
                  <xsl:choose>
                    <xsl:when test="nota_media < 5">suspenso</xsl:when>
                    <xsl:when test="nota_media < 9">aprobado</xsl:when>
                    <xsl:otherwise>sobresaliente</xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
                <td><xsl:value-of select="nombre"/></td>
                <td><xsl:value-of select="apellidos"/></td>
                <td style="text-align: center;"><xsl:value-of select="nota_media"/></td>
              </xsl:element>
            </xsl:for-each>
          </tbody>
        </table>
      </body>
    </html>
</xsl:template>

</xsl:stylesheet>
```

El **resultado** será una tabla como la que vemos en la siguiente figura:

NOMBRE	APELLIDOS	NOTA MEDIA
Ana	Abad Álvarez	6.5
Alejando	Álvarez Amate	5.25
Ana	Amate Antunez	7.5
Ana	Antunez Amaya	2.5
Benito	Bellido Bravo	2.25
Benito	Benítez Bellido	6.5
Beatriz	Benitez Bermúdez	8.25
Beatriz	Bravo Benítez	7.75
Carmen	Camacho Camacho	10
Carlos	Carmona Casado	4.75
Carmen	Casado Carmona	4
Carmen	Cuesta Camacho	8.75

Figura 5.3.2.: Tabla en HTML resultado

Que tendrá como **código HTML** el **generado** por la transformación XSLT que hemos realizado y que es el siguiente:

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <style>
      .aprobado {color:green;}
      .suspenso {color:red;}
      .sobresaliente {color:blue}
    </style>
  </head>

  <body>
    <table border="0">
      <thead style="background-color:gray; color:white;">
        <th>NOMBRE</th><th>APELLIDOS</th><th>NOTA MEDIA</th>
      </thead>
      <tbody style="background-color:white;">
        <tr class="aprobado">
          <td>Ana</td>
          <td>Abad &Aacute;lvarez</td>
          <td style="text-align: center;">6.5</td>
        </tr>
        <tr class="aprobado">
          <td>Alejando</td>
          <td>&Aacute;lvarez Amate</td>
          <td style="text-align: center;">5.25</td>
        </tr>
        <tr class="aprobado">
          <td>Ana</td>
          <td>Amate Antunez</td>
          <td style="text-align: center;">7.5</td>
        </tr>
        <tr class="suspenso">
          <td>Ana</td>
          <td>Antunez Amaya</td>
          <td style="text-align: center;">2.5</td>
        </tr>
        <tr class="suspenso">
          <td>Benito</td>
          <td>Bellido Bravo</td>
          <td style="text-align: center;">2.25</td>
        </tr>
        <tr class="aprobado">
            <td>Benito</td>
            <td>Ben&iacute;tez Bellido</td>
            <td style="text-align: center;">6.5</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

NOTA: no se han incluído todos los elementos td generados en el código anterior, para que sea más legible y se quepa en una sola página, pero nos podemos hacer una idea del código generado por la transformación XSLT.

5.3.3.3 Instrucciones de Salida

Estas instrucciones nos permiten manipular la salida de la transformación, por ejemplo, añadiendo atributos o elementos al árbol resultado, realizando conversiones de datos o añadiendo comentarios o instrucciones de procesamiento.

Las **principales instrucciones** en este grupo son las siguientes:

- **xsl:value-of**

  Este elemento se utiliza para generar texto, ya sea extraído del árbol de nodos origen, generado por una expresión XPath o insertado de valores de una variable. El elemento **xsl:value-of** creará un nodo de tipo text en el árbol de nodos resultado. Tiene un atributo obligatorio, **select**, en el que se indicará la expresión que será evaluada y su resultado se convertirá a una cadena de caracteres. Si queremos añadir un conjunto de nodos, es decir, elementos con sus correspondientes hijos, al árbol de nodos destino, deberíamos utilizar **xsl:copy-of** en su lugar.

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes" />
  <xsl:template match="/">
    <xsl:text>
    </xsl:text>
    <xsl:comment>Esto es un saludo</xsl:comment>
      <xsl:text>
      </xsl:text>
    <xsl:element name="saludo">
      <xsl:attribute name="idioma">español</xsl:attribute>
      <xsl:text>Hola, bienvenido al colegio </xsl:text>
      <xsl:value-of select="colegio/nombre" />
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

  El **resultado** de esta transformación sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
    <!--Esto es un saludo-->
    <saludo idioma="español">Hola, bienvenido al colegio Cervantes</saludo>
```

- **xsl:number**

  Se usa para insertar un número formateado en el árbol de nodos de destino. El número que se va a insertar se debe especificar en el atributo **value** mediante una expresión. La expresión se evaluará y el objeto resultante se convertirá en un número. El número se redondeará a un entero y después se convertirá en una cadena de caracteres. Si no se especifica el atributo **value** el número añadido estará basado en la posición del nodo contexto. Los siguientes **atributos** nos ayudan a controlar como se realiza la conversión:

  - **level**: especifica que niveles del árbol de nodos origen se utilizarán. Sus valores pueden ser **single**, **multiple** o **any**. Por defecto, su valor será **single**.
  - **count**: será un patrón que especifique que nodos serán contados en ese nivel. Por defecto, se contarán todos los nodos semejantes al nodo contexto.
  - **from**: es un patrón que especifica desde donde se empieza a contar.

  Para **especificar un formato** se utiliza el atributo opcional **format**, al que se le indicará un patrón con una combinación de los siguientes caracteres:

- Un **token que acabe en 1**: formará una secuencia numérica. Por ejemplo, 1 formará la secuencia 1, 2, 3,... y 001 formará la secuencia 001, 002, 003, ...
- **Letra A**: formará la secuencia A, B, C.., AA, AB, ...
- **Letra a**: formará la secuencia a, b, c.., aa, ab, ...
- **Letra I**: formará la secuencia I, II, III, IV, ...
- **Letra i**: formará la secuencia i, ii, iii, iv, ...
- Otras numeraciones especificadas en **unicode**: **&#x30A2**; **&#x30A4**; **&#x0E51**; **&#x05D0**; **&#x10D0**; **&#x03B1**; **&#x0430**;

El patrón por defecto es 1.

- **xsl:element**

  Esta instrucción permite crear un elemento con el nombre indicado por el atributo obligatorio **name**. También se le puede indicar un espacio de nombres de forma opcional con el atributo **namespace**. El contenido de este elemento podrá ser, por ejemplo, elementos **xsl:attribute**, otros elementos **xsl:element** hijos, comentarios **xsl:comment**, etc..

- **xsl:attribute**

  Este elemento puede ser usado para añadir atributos a los elementos creados en un plantilla mediante el elemento **xsl:element** o directamente escribiendo las etiquetas de apertura y cierre (LRE). El nombre del atributo será indicado mediante el atributo obligatorio **name**. También se le puede indicar un espacio de nombres opcional con el atributo **namspace**. El contenido de este elemento será el valor del atributo.

  Al igual que una elemento se puede crear mediante **LRE**, los atributos también pueden construirse mediante el uso de llaves . A esto se le llama **attribute template values**. Consiste en escribir el atributo dentro de su correspondiente etiqueta y cuando se quiere introducir el valor indicarlo mediante una expresión entre llaves .

  Si un elemento tiene varios atributos o un atributo que se quiere reutilizar se puede usar la instrucción **xsl:attribute-set**.

- **xsl:comment**

  Este elemento creará un nodo comentario en al árbol resultado. Su contenido será una cadena de texto.

- **xsl:processing-instruction**

  Este elemento creará un nodo de instrucción de procesamiento en el árbol de nodos destino. En el atributo obligatorio **name** se deberá especificar el nombre o clase de instrucción de procesamiento.

  A partir de nuestro documento **colegio.xml**, vamos a crear otro documento XML que contenga un listado de alumnos ordenador por notas cuyo nombre comience por 'A':

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" version="1.0" indent="yes" />
  <xsl:template match="/">

  <xsl:text>
  </xsl:text>

  <xsl:comment>Documento de estilos CSS vinculado al archivo XML resultante</xsl:comment>

  <xsl:text>
  </xsl:text>

  <xsl:processing-instruction
    name="xml-stylesheet">href="style.css" type="text/css"</xsl:processing-instruction>

  <xsl:text>
  </xsl:text>

  <xsl:element name="listado">
    <xsl:comment>
      Listado de alumnos ordenados por notas cuyo nombre empieza por A
    </xsl:comment>
    <xsl:for-each select="//alumno[starts-with(nombre,'A')]">
      <xsl:sort select="nota_media" order="descending" data-type="number"/>
      <xsl:element name="persona">
        <xsl:attribute name="calificacion">
          <xsl:value-of select="nota_media"/>
        </xsl:attribute>
        <xsl:attribute name="num_orden">
          <xsl:number value="position()" format="1"/>
        </xsl:attribute>
        <xsl:value-of select="nombre"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="apellidos"/>
      </xsl:element>
    </xsl:for-each>
  </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

El resultado sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Documento de estilos CSS vinculado al archivo XML resultante-->
<?xml-stylesheet href="style.css" type="text/css"?>
<listado>
    <!--Listado de alumnos ordenados por notas cuyo nombre empieza por A-->
    <persona calificacion="7.5" num_orden="1">Ana Amate Antunez</persona>
    <persona calificacion="6.5" num_orden="2">Ana Abad Álvarez</persona>
    <persona calificacion="5.25" num_orden="3">Alejando Álvarez Amate</persona>
    <persona calificacion="2.5" num_orden="4">Ana Antunez Amaya</persona>
</listado>
```

5.3.4 Funciones Propias de XSLT

Además de las instrucciones que hemos visto en la sección anterior, XSLT proporciona un número de funciones propias que nos permiten realizar diferentes operaciones.

Estas funciones son:

- *node-set* **document(*object*, *node-set?*)**

  Función que permite el acceso a otros documentos XML distintos del original. Tiene dos parámetros. El **primero** que indica **la URI** donde está el documento a añadir. Si no se indica nada, se tomará el documento actual como valor por defecto. El **segundo** parámetro opcional nos permite seleccionar una parte de ese nuevo documento mediante una expresión XPath. Es función **devuelve** un conjunto de nodos.

- *node-set* **key(*string*, *object*)**

  Esta función es similar a **id()** en XPath. El primer parámetro es el nombre de la clave, que debe ser un nombre cualificado indicado en un elemento **xsl:key**. El segundo debe ser el valor para dicha clave. Devuelve el conjunto de nodos del documento que cumplen con estas condiciones.

- *string* **format-number(*number*, *string*, *string?*)**

  Convierte el primer parámetro a una cadena de caracteres siguiendo el patrón indicado en el segundo parámetro. Si existe un tercer parámetro indicará el nombres de **xsl:decimal-format** que se utilizará, si no se indica, se cogerá el valor por defecto.

- *node-set* **current()**

  Devuelve el valor del nodo contexto. Normalmente se sustituye por su versión abreviada, usando el carácter para designar al nodo contexto (**.**).

- *string* **generate-id(*node-set?*)**

  Genera un identificador único para el conjunto de nodos pasado como parámetro. Si no se especifica el conjunto de nodos, se tomará el nodo contextual por defecto.

Otras funciones menos usadas son: **unparsed-entity-uri()**, **system-property()**, **element-avaible()** y **function-avaible()**.

En el siguiente **ejemplo** vamos a realizar una transformación para calcular la media de notas de los alumnos agrupados por nombre.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <xsl:key name="nombre_clave" match="//alumno/nombre" use="."/>
      <html>
        <body>
          <ol>
            <xsl:for-each
              select="//alumno/nombre[generate-id() = generate-id(key('nombre_clave',.)[1])]">

              <xsl:sort select="." order="ascending" data-type="text"/>
              <xsl:variable name="nombre_actual" select="."/>

              <xsl:variable name="media"
              select="sum(//alumno[nombre=$nombre_actual]/nota_media) div
              count(//alumno[nombre=$nombre_actual]/nota_media)"/>
              <li>
                <xsl:value-of select="$nombre_actual"/>
                <xsl:text>. Media de notas: </xsl:text>
                <xsl:value-of select="$media"/>
              </li>
            </xsl:for-each>
          </ol>
        </body>
      </html>
    </xsl:template>

</xsl:stylesheet>
```

En este caso, generamos una clave por cada nombre de alumno, recorremos los nombres de alumnos

que al generar de nuevo la clave es igual a la primera ya creada. Realizamos la media agrupando por el nombre que tenemos seleccionado. Otra forma de hacerlo hubiera sido usando el eje **preceding** para evitar elementos repetidos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" version="5.0" indent="yes" />

  <xsl:template match="/">
    <html>
      <body>
        <ol>
          <xsl:for-each select="//alumno[not(nombre=preceding::nombre)]">
            <xsl:sort select="nombre" order="ascending" data-type="text"/>
            <xsl:variable name="nombre_actual" select="nombre"/>
            <xsl:variable name="media"
            select="sum(//alumno[nombre=$nombre_actual]/nota_media) div
            count(//alumno[nombre=$nombre_actual]/nota_media)"/>
            <li>
              <xsl:value-of select="$nombre_actual"/>
              <xsl:text>. Media de notas: </xsl:text>
              <xsl:value-of select="$media"/>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
</xsl:template>

</xsl:stylesheet>
```

5.3.5 Procesadores XSLT

Un **procesador XSLT** es una aplicación que realiza una transformación en un documento XML siguiendo las reglas indicadas en otro documento XSLT. El resultado, será otro documento en alguno de los formatos permitidos: XML, HTML y texto plano.

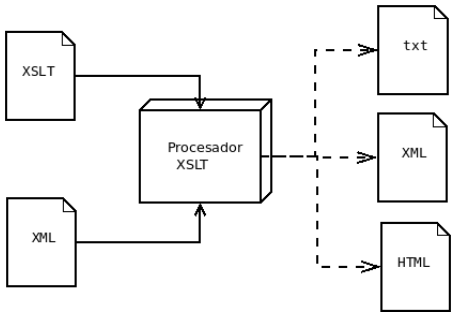


Figura 5.3.3.: Esquema procesador XSLT

Como hemos comentado a lo largo de esta unidad, el procesador XSLT **lee el archivo XML inicial** y crea el árbol de nodos origen. A continuación, va **procesando** las **instrucciones de la hoja de estilos** recogida en el documento XSLT. Esta hoja de estilos contendrá una o varias plantillas, las cuales tendrán una expresión de emparejamiento. Estas expresiones serán buscadas en archivo XML original y si encuentra coincidencias se ejecutarán las instrucciones indicadas. Estas instrucciones irán **generando el árbol de nodos destino**. Una vez recorridas todas las plantillasy generado el árbol de nodos destino, éste se escribirá en el documento de salida.

También hay que tener en cuenta que el consorcio W3C publica sus recomendaciones sobre el lenguaje XSLT pero no se encarga de desarrollar los procesadores. Existen varias empresas que se dedican a ello.

Algunos de las empresas que ha **desarrollado procesadores** de XSLT son las siguientes:

- **Saxon**

  Desarrolla su procesador XSLT llamado saxon con las recomendaciones de la W3C mas recientes: XSLT 3.0 y XPath 3.1.

  Se distribuye como librerías de programación para varios lenguajes como Java, .NET, C/C++, PHP, Python y Javascript, aunque también se puede usar en la línea de comandos gracias al empaquetado de aplicaciones de jar, de Java.

  El fundador es esta empresa es Michael Kay, uno de los participantes en las recomendaciones del W3C.

- **Gnome**

  Tiene una implementación de las recomendaciones XSLT 1.0 y XPath 1.0 en una librería llamada libxslt. Esta librería tiene una utilidad de línea de comandos llamada xsltproc.

  Gnome tiene otra librería relacionada con XML llamada libxml2 que permite trabajar y validar documentos XML. También tiene una utilidad de línea de comandos llamada xmllint

- **Apache**

  Desarrolló un procesador de XSLT muy utilizado llamado Xalan. Este procesador implementaba las recomendaciones XSLT 1.0 y XPath 1.0. Si distribuía como librerías para los lenguajes Java y C/C++. Su última versión estable es la 2.7.1 lanzada en Noviembre de 2007.

  Dentro de Apache existe un proyecto hermano llamado Xerces que permite trabajar con documentos XML y validarlos.

- **Microsoft**

  Desarrollo una librería con funciones XML llamada MSXML (Microsoft XML Core Services) que implementaba XML 1.0, XML Schema 1.0 (2ª Edición), XPath 1.0 y XSLT 1.0. Su última versión es la 6 de 2016.

La forma de usar estos procesadores varía según las necesidades. Podemos encontrar procesadores XSLT que se ejecutan en la **línea de comandos** de nuestro sistema operativo (xsltproc, saxonb-xslt, xalan), como **librerías** de distintos lenguajes de programación, en **páginas web** accesibles con nuestro navegador (xslttest, xsltransform) o integrados en **editores específicos** de XML (XML Copy Editos, Oxygen XML, Stylus Studio) o **generales** (Netbeans, Visual Studio, Eclipse, ...).

5.3.6 Depuración de Documentos XSLT

Algunos editores incluyen la funcionalidad de **depuración del código**.

Esta funcionalidad permite **ejecutar paso a paso** las instrucciones de forma que facilita enormemente **encontrar y corregir errores**. Además suelen permitir mostrar los **valores** que van tomando las **variables** o estructuras de datos. También permiten establecer **puntos de ruptura** (breakpoints) o ejecución hasta el cursos de forma que podemos ejecutar el código hasta encontrar estos puntos de ruptura o la línea donde tenemos el cursos.

Entre los editores genéricos con versión open source y que tengan esta funcionalidad de depuración tenemos:

- IntelliJ IDEA + Plugin XSLT Debugger
- Eclipse + Plugin EclipseXML Editors and Tools
- NetBeans + Plugin netbeans-xslt-debugger (obsoleto)

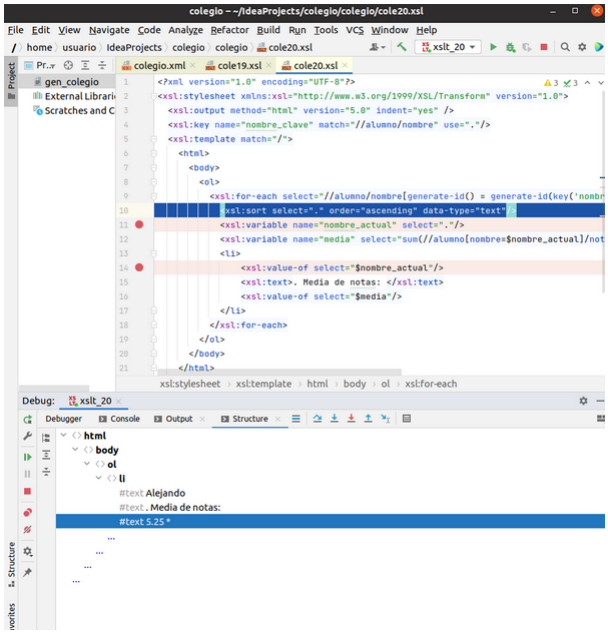


Figura 5.3.4.: Depuración XSLT en Intellij IDEA

Dentro de los **editores especializados** en XML no tenemos ninguno con versión gratuita, pero sin con períodos de evaluación de 15 o 30 días:

- Oxygen XML Editor
- Liquid XML Studio
- Editix
- Stylus Studio
- Altova XMLSpy

A. Anexos Tema 3

A.1 Servidores

Un **servidor**, como la misma palabra indica, es un ordenador o máquina informática que esta al "servicio" de otras máquinas u ordenadores, llamadas **clientes**, y que suministran a estos todo tipo de información.

Este tipo de relación entre diferentes dispositivos genera uno de los esquemas mas empleados y en el que se basa gran parte de internet, como es el esquema **cliente-servidor**, del cual podemos ver un ejemplo en la siguiente figura.

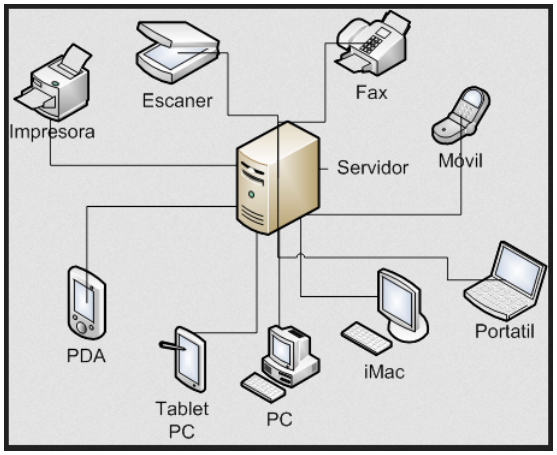


Figura A.1.1.: Arquitectura Cliente-Servidor

Como vemos en este esquema, tenemos una máquina servidora que se comunica con diferentes clientes, todos demandando algún tipo de información. Esta información puede ser desde archivos de texto, audio, vídeo, imágenes, programas, consultas a bases de datos, etc...

Los **servidores** suelen ser **más potentes** que un ordenador normal, sobre todo en lo relativo **capacidad de almacenamiento** como de **memoria principal**, ya que suelen dar servicios a muchos clientes, aunque esto también depende de las necesidades, ya que si vamos a tener pocos clientes, podemos tener un servidor con menores prestaciones.

Existen muchos **tipos de servidores**, como servidores de corre, proxy, web, de bases de datos, de aplicaciones, etc...

A.1.1 Servidores Web

Un **servidor Web** se encarga de almacenar documentos HTML, imágenes, vídeos, documentos y todo tipo de información relacionado con una web. Además, se encarga de servir esta información y hacerla llegar a los clientes.

A la hora de crear una página web será necesario contar con un **dominio** y un **servidor** donde alojar los archivos de nuestra web. Si se trata de una web muy extensa, se recomienda usar un **servidor local** mientras se desarrolla, ya que si contratamos un hosting de pago antes de que este finalizada tendremos un periodo de tiempo en el que no lo estaremos usando, ademas tenemos el riesgo de **no tener** aún el **SEO adecuado**, ya que durante el proceso de desarrollo se irá añadiendo mucha información.

Un **servidor local** es simplemente un ordenador común con una serie de aplicaciones instaladas para que podamos usarlo de servidor de pruebas y así corregir las imperfecciones y realizar los cambios necesarios en nuestra página web hasta que estemos seguro de que la podemos poner online. Suelen ser servidores que se instalan en el **mismo ordenador** u **otro diferente** en donde se realiza el desarrollo de la web pero que **solo ofrecen servicios** a nuestra **red local**.

También tenemos la posibilidad de hacer que **nuestro ordenador** funcione como un servidor online. En este caso necesitaremos un conjunto de aplicaciones entre las que podemos encontrar:

- **Sistema Operativo**: debemos tener instalado un sistema operativo, ya sea **Windows**, **Linux**, **MacOS** o algún otro de nuestra preferencia.
- **Servidor Web**: debemos tener instalada una aplicación de servidor web, que permita al ordenador ofrecer esta función. Algunos de los más usados son **Apache**, **Nginx** o **LiteSpeed**.
- **Base de Datos**: por norma general las aplicaciones de servidores web nos pedirán que tengamos instalado un sistema de bases de datos. Algunos de los más usados son **MySQL**, **PostgreSQL** o **MongoDB**.
- **Soporte para Lenguaje del Servidor**: también debemos tener instalado el soporte para el lenguaje que vayamos a usar en la parte del servidor o **backend**, ya sea este **PHP**, **Java**, **Javascript**, **Ruby**, etc..

La mayoría de las aplicaciones que podemos emplear son de **código abierto** y todas pueden ser instaladas por separado, aunque podemos encontrar **paquetes** que incluyen todas las aplicaciones necesarias para poder empezar a funcionar como un servidor.

Uno de los paquetes más empleados es **LAMP** (Linux, Apache, MySQL y PHP), que nos permitirá instalar todo lo necesario de una sola vez así como gestionar los diferentes servicios desde una misma interfaz de forma cómoda y sencilla.

Glosario

ASCII American Standard Code for Information Interchange. 7

CRM Customer Relationship Management. 18

DTD Document Type Definition. 5

ERP Enterprise Resourcer Planning. 18

GML Generalized Markup Language. 7

HTML Hypertext Markup Language. 7

metalenguaje Lenguaje que permite la definición de otros lenguajes. 8

OBCL Openbravo Commercial Licence. 21

OBPL Openbravo Public Licence. 20

SaaS Software as a Service. 22

SGML Standard Generalized Markup Language. 7

texto plano Es aquel texto formado solo por datos sin formato, es decir, solo por caractéres.. 10

unicode Es un código que permite el tratamiento informático de textos en cualquier lenguaje y disciplina técnica, ya que incluye todos los carácteres conocidos para cualquier lengua. Es compatible con ASCII. 12

URI Uniform Resource Identifier. Son hypervínculos que dan acceso a un recurso remoto. 17

UTF-8 8-bit Unicode Transformation Format. 12

W3C Word Wide Web Consortium. 8

XML eXtensible Markup Language. 8

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.