UNIDAD DIDÁCTICA 5

# CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS PARA EL INTERCAMBIO DE INFORMACIÓN

## MÓDULO PROFESIONAL: LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

CESUR
Tu Centro Oficial de FP

# Índice

# RESUMEN INTRODUCTORIO

A lo largo de esta unidad estudiaremos qué son las tecnologías de transformación de documentos y los estándares y ámbitos de aplicación conociendo también la descripción de la estructura y de la sintaxis.

A continuación, veremos la creación y utilización de plantillas conociendo las herramientas y depuración.

Por último, trataremos la conversión entre diferentes formatos de documentos.

# INTRODUCCIÓN

Una ventaja de los lenguajes de marcas o markup language, es la posibilidad de añadir una serie de etiquetas al contenido de texto original que poseen una función o significado concreto y permiten que los textos sean procesados o interpretados correctamente.

Dentro de este tipo de lenguajes destacamos XML, considerado como un metalenguaje, al ir un paso más allá de los lenguajes de marcas, pues ofrece al usuario la opción de poder definir elementos propios, a través de esas etiquetas, creando incluso una estructura propia, lo que nos lleva a poder utilizar XML con diferentes lenguajes y aplicaciones, es uno de los factores que lo convierten en un lenguaje muy interesante de conocer.

Además, XML es un buen formato para estructurar e intercambiar información. No obstante, un sistema no tiene por qué necesitar siempre todos los datos contenidos en un XML, o puede requerirlos en un formato diferente. Para dar solución a este problema, surge XSL, una especificación que permitirá transformar un XML en otro de forma sencilla. De esta forma, se podrá obtener un documento con los datos que se requieran, al gusto del usuario, sin tener que reescribirlo.

Por otro lado, también es posible asociar una hoja de estilos a un archivo XML, permitiendo que su contenido se pueda mostrar con el aspecto de una página web.

# CASO INTRODUCTORIO

En la empresa de desarrollo de software que trabajas, tenéis uno de los clientes con actividad de venta online de moda y complementos, que necesita simplificar la consulta a archivos XML que se exportaron en su día para la obtención de datos en formato tabla a través de un navegador web.

Va a ser el programador encargado de crear un archivo de transformación de los archivos XML existentes con el objetivo de poderlos mostrar en un navegador web, tal y como solicita el cliente.

Al final de la unidad serás capaz de reconocer la sintaxis del lenguaje XSLT, conocerás las aplicaciones de los lenguajes de marcas XML, XSL y Xpath y sabrás utilizar el lenguaje Xpath para separar la información de los documentos XML de las etiquetas de estos, donde aplicarás el lenguaje XSLT para dar formato a los datos.

# 1. TECNOLOGÍAS DE TRANSFORMACIÓN DE DOCUMENTOS. ESTÁNDARES. ÁMBITOS DE APLICACIÓN

*Como programador especializado en lenguaje de marcas, formas parte del equipo desarrollador del departamento de informática de tu empresa, que está centrado en la utilización de XML, por lo que debes establecer qué tecnologías de transformación de documentos ofrece XML y concretar sus ámbitos de aplicación.*

A partir de un documento XML, es posible desarrollar un sistema de intercambio de información entre dispositivos o aplicaciones, pero a su vez, también existe la posibilidad de definir transformaciones a aplicar con el fin de convertir ese documento a otro formato distinto a XML.

Estas transformaciones convertirán el documento XML en un documento de otro tipo, en el que se deberá tener en cuenta la compatibilidad, en el sentido de compartir algunas partes de la estructura, y que sea legible para un usuario final.

La aplicación de una transformación de documento XML suele ser muy útil en determinados casos, en los que se espera disponer de un mismo documento con distintos formatos, partiendo del documento original XML se puede obtener una versión de texto ASCII, en otro vocabulario de XML como RSS, SVG, etc., u otros formatos como RTF, ODT, DOCX, HTML, TeX o PDF.

Como ya sabemos, XML nos permite diseñar documentos que podemos archivar conteniendo información estructurada mediante la utilización de etiquetas, que permite su posterior presentación en un formato de visualización de acuerdo con nuestras necesidades.

Se ha convertido en la tecnología base de gran cantidad de aplicaciones en diferentes ámbitos como son:

- Desarrollo y diseño web, con posibilidad de usar CSS.
- Transformación de documentos XML, en otro, e incluso en otros formatos como HTML, para utilización en navegadores web.
- Herramientas de productividad y procesadores de textos, como Excel, Word, etc.
- Software de publicaciones, como pueden ser libros.
- En aplicaciones para móviles, tanto en Android como iOS.
- Intercambio de información compartida con formato JSON, por ejemplo.


Ámbitos aplicación XML.
Fuente: https://uapas2.bunam.unam.mx/matematicas/ventajas_xml/

Uno de sus usos principales es en el intercambio de datos entre distintas plataformas, aplicaciones o programas, gracias a sus posibilidades de configuración en la estructura de datos, independientemente de la fuente de donde se extraen. Permite la utilización de ficheros de aplicaciones para el almacenamiento y recuperación de datos, por parte de los usuarios.

En esa manipulación de documentos XML, pueden utilizarse determinadas tecnologías de acuerdo a sus correspondientes estándares de presentación de información, entre las que podemos citar:

- **XSL:** Lenguaje Extensible de Hojas de Estilo, que tiene por objetivo mostrar la estructura que debería tener el contenido para una correcta lectura, de modo que establece cómo debería ser diseñado el contenido de origen y cómo debería ser paginado en un medio de presentación, por ejemplo, mediante un navegador web o un dispositivo móvil.

- **XPath:** Lenguaje de Rutas XML, permite el acceso a las partes de un documento XML. Con este lenguaje se recorren y procesan documentos XML.

- **XSLT:** Desarrolla el lenguaje de transformación.Hace uso de la especificación XPath, cuyo diseño permite la utilización de forma independiente, aunque es utilizada desde la especificación XSL.

- **XLink:** Lenguaje de Enlace XML, permite insertar elementos en documentos XML para crear enlaces entre recursos XML.

- **XPointer:** Lenguaje de Direccionamiento XML, ofrece la posibilidad de acceso a la estructura interna de un documento XML: Elementos, atributos y contenido, pudiendo identificar de forma única fragmentos del documento.

- **XQL:** Lenguaje de Consulta XML, facilita la extracción de datos desde documentos XML. Con opciones como la de realizar consultas flexibles que permiten la extracción de datos de documentos XML en la web.

Vamos a centrarnos en:

**XSL.**

Por ejemplo, mediante transformaciones XSL sería posible hacer que un documento que almacena información sobre un libro, con sus correspondientes capítulos y secciones, se transformara en un documento HTML en el que apareciera el título del libro con un encabezado de tipo h1, los capítulos con h2 y sus correspondientes secciones con h3.
Ejemplo de documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<libro>
<titulo></titulo>
      <capitulo>
            <titulo></titulo>
            <seccion>
                  <titulo></titulo>
            </seccion>
      </capitulo>
</libro>
```

Este documento XML, se puede transformar en otro HTML mediante el siguiente XSL (su funcionamiento se verá más adelante):

```
<!-- Transforma el documento XML anterior en un documento XHTML
-->
<xsl:stylesheet version="1.0"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:strip-space elements="libro capitulo titulo"/>
<xsl:output method="xml" indent="yes" encoding="iso-8859-1"
doctype-public="-//W3C//DTD XHTML 1.1//EN"
doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
<!-- Utiliza el título del libro como título del documento XHTML
-->
<xsl:template match="libro">
      <html>
            <head>
                  <title>
                  <xsl:value-of select="titulo"/>
                  </title>
            </head>
            <body>
                  <xsl:apply-templates/>
```

```
            </body>
      </html>
</xsl:template>

<!-- Y también como título de nivel h1 -->
<xsl:template match="libro/titulo">
      <h1>
            <xsl:apply-templates/>
      </h1>
</xsl:template>

<!-- Los títulos de los capítulos aparecerán como h2 -->
<xsl:template match="capitulo/titulo">
      <h2>
            <xsl:apply-templates/>
      </h2>
</xsl:template>

<!-- Los títulos de las secciones aparecerán como h3 -->
<xsl:template match="seccion/titulo">
      <h3>
            <xsl:apply-templates/>
      </h3>
</xsl:template>

</xsl:stylesheet>
```

XSL funciona como un lenguaje avanzado para crear hojas de estilos. Es capaz de transformar, ordenar, filtrar datos XML y darles formato basándolo en sus valores.

**XSLT.**

XSL cuenta con un lenguaje que soporta la ejecución de toda esta variedad de transformaciones que se puedan necesitar. Se trata del **XSLT** (Extended Styleshet Language Transformation), definido como un lenguaje XML basado en reglas que transforman la estructura y contenido de un documento XML en otro, también basado en texto, siguiendo las indicaciones de un segundo documento llamado transformación u hoja de estilo que varía según sea el resultado final que se persiga.

La procedencia común del CSS y del XSL explica que en la terminología XSLT se hable de hoja de estilo. La evolución del XML ha demostrado que un lenguaje dirigido a transformaciones es necesario en muchas tareas, y no exclusivamente en las relacionadas con la presentación o el estilo, aunque la tarea de especificar el estilo de un documento siga siendo una de las funciones más importantes del lenguaje XSLT dentro de la familia XML.

De forma más concreta, en XML se llama transformación a un programa que a partir de un documento fuente XML y de un documento hoja de estilo, lee y cambia el documento fuente para obtener un documento resultado, siguiendo las reglas expresadas en el contenido del documento hoja de estilo. Tras ser sometido al proceso de transformación, el documento resultante puede ser muy distinto de la fuente, incluyendo la posibilidad de añadir texto o etiquetas, aunque siempre obteniendo un tipo de documento basado en texto.

Por tanto, los procesadores XSLT partiendo de dos documentos, la fuente y la hoja de estilo generan un archivo que puede, en principio tener casi cualquier formato estructurado basado en texto.

El procesado XSLT puede darse en tres contextos diferentes:

a. En un proceso “batch”.
b. Sobre un servidor Web.
c. En un navegador.

En el primer caso, el procesador parte de archivos (los archivos fuente y las correspondientes hojas de estilo) y genera otro archivo sobre una determinada carpeta que representa los resultados de la transformación.

Cuando el procesado tiene lugar en un servidor Web, también involucra archivos fuentes y hojas de estilo, aunque ahora en lugar de escribir los archivos resultantes en un disco, los resultados fluyen en el navegador del usuario. Cuando los navegadores soportan el procesado XSLT, tanto los archivos fuente como las hojas de estilo se mandan al navegador que presenta el resultado correspondiente directamente al usuario. Como puede verse, el entorno de trabajo del XSLT puede ser diverso, aunque su funcionamiento sea el mismo en los tres contextos.



**ENLACE DE INTERÉS**

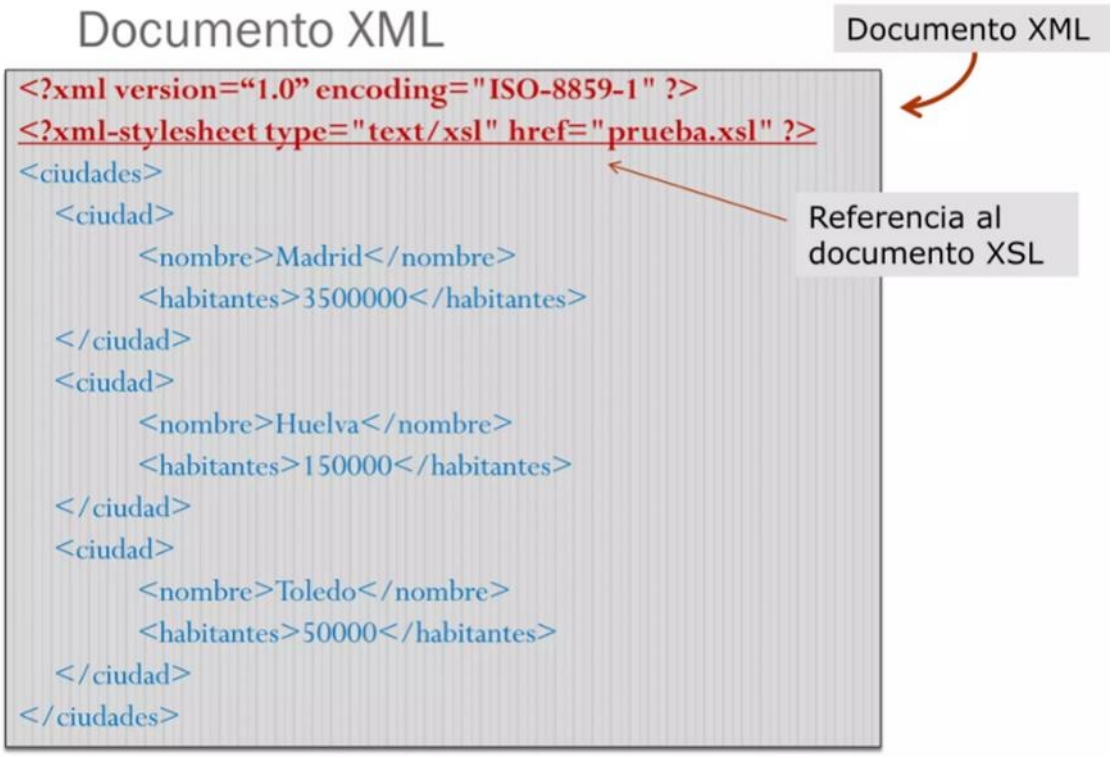
Accede a esta web para conocer el manual sobre tecnologías XML, para la creación de sitios web:

# 2. DESCRIPCIÓN DE LA ESTRUCTURA Y DE LA SINTAXIS

*Dentro del desarrollo de software en el que trabajas en el departamento de informática de tu empresa, tienes que crear un documento XSLT para transforma los documentos XML actuales y permitan la transmisión de información del sitio web con el que estás trabajando.*

Partiendo de un documento XML inicial, vamos a obtener un resultado utilizando XSLT para creación de un documento que contendrá el código fuente del programa, incorporando las reglas de transformación que se aplican al documento de inicio. Siendo necesario el procesador para aplicar esas reglas.

Conocer la estructura y sintaxis de este tipo de documentos es fundamental para su correcta aplicación, como vamos a ir viendo a continuación en su modelado y procesado, la utilización del XPath y la estructura de la hoja de estilo en XSLT.




Ejemplo de llamada a un documento XSL.

Fuente: https://es.slideshare.net/juanjosetaboadaleon/u71-xsltcurso201516

## 2.1 Modelo y procesado XSLT

La hoja de estilo XSLT incorpora la descripción de una serie de reglas que indican cómo deben procesarse cada uno de los nodos del documento fuente, usando para ello una estructura de objetos como modelo de datos.

Estas reglas, son conocidas también como patrones o plantillas (template rules) ya que durante la ejecución tiene lugar una serie sistemática de comprobaciones entre partes del documento fuente y estos patrones, de forma que cuando se produce un ajuste de alguna de ellas, se dispara el mecanismo, obteniendo el resultado especificado en la regla. Se llama constructor al programa que después de evaluar secuencialmente los nodos documento fuente obtiene a partir de ellos una parte del árbol resultado, de acuerdo con lo declarado por las reglas de las hojas de estilo correspondiente.

Toda regla en XSLT consta de dos partes: una primera en la que dado un patrón se buscan los elementos del árbol fuente que se ajustan a él, para lo que se recurre a expresiones **XPath**, y una segunda donde la plantilla construye una parte del árbol resultado con el constructor.

Un documento XSL es un documento XML con el espacio de nombres http://www.w3.org/1999/XSL/Transform (para el que suele usarse como prefijo xsl) y cuyo elemento raiz es xsl:stylesheet. El documento puede incluir tanto elementos no definidos por XSLT, como otros procedentes de otras hojas de estilo incorporados por inclusión o por importación. Su sintaxis es la de XML y su vocabulario consta de etiquetas especiales que representan operaciones concretas a realizar con el texto del documento fuente.

Por tanto, su cabecera debe ser:

```
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
```

Ahora, solo falta asociarlo con un documento XML al que se desea aplicar. Para ello, se emplea la etiqueta de declaración <?xml-stylesheet?>. Por lo que queda:

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

El atributo type define el tipo de fichero (los dos valores posibles que puede tomar son text/xsl, que denota un documento XSL y text/css, que denota un documento css) y el atributo href indica el archivo correspondiente a la hoja de estilo.

**ENLACE DE INTERÉS**

Aquí encontrarás más información sobre la referencia a los estándares XSLT:

## 2.2 XPath

XML Path Language, es un lenguaje que permite escribir expresiones regulares para recorrer y procesar un documento XML.

Por ejemplo, una expresión consistente en un nombre, producirá una identificación con los elementos que tengan dicho nombre. En el siguiente caso, si se tiene el elemento "titulo":

```
<titulo>La historia de PI</titulo>
```

Y la hoja de estilos contiene una plantilla como:

```
<xsl:template match="titulo">
    <h2>
<xsl:value-of/>
    </h2>
<xsl:template>
```

Al producirse un ajuste de la expresión con el elemento <titulo>, se procederá a procesar el contenido de esa plantilla. La expresión <xsl:value-of/>, se sustituye por el valor del elemento capturado por la plantilla que, en este caso, es <titulo>, por lo que el valor extraído será "La historia de PI". Por tanto, la transformación realizada es:

```
<h2>La historia de PI</h2>
```

Los pasos que sigue un procesador XSLT son:

1. Examinar los documentos fuente y las hojas de estilo.
2. Analizar y comparar los nodos para aplicar las reglas de la hoja de estilo.

3. Crear el árbol resultado.
4. Completar el documento resultante.

Por ejemplo, para el siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="intro.xsl"?>
<miMensaje>
      <mensaje> Bienvenidos a XSLT </mensaje>
</miMensaje>
```

Donde el contenido de la hoja de transformaciones intro.xsl, es:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version= "1.0" xmlns:xsl
"http://www.w3.org/1999/XSL/Transform">
<xsl:template match="miMensaje">
    <html>
    <body><xsl:value-of select="mensaje"/></body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

En el primer paso, el procesador XSLT lee ambos contenidos.

En el siguiente paso, en la hoja de estilo, la instrucción <xsl:template match=”miMensaje”> define el elemento XSLT template, cuyo atributo match especifica los nodos del documento fuente a los que hace referencia. Para ello, se usa una expresión XPath, que en este caso es el elemento miMensaje. Por tanto, el template encajará con:

```
<miMensaje>
      <mensaje> Bienvenidos a XSLT </mensaje>
</miMensaje>
```

A la hora de construir el árbol resultado, tras comenzar por <html> y <body>, se procesa xsl:value-of select en el contexto del elemento que ha encajado con el template. Por tanto, obtendrá el contenido de la etiqueta <mensaje> que se encuentra dentro del anterior fragmento de código. Es decir: “ Bienvenidos a XSLT ”.

El árbol resultado, indica la sucesión de elementos HTML y de texto que deben copiarse en la salida.

De esta forma, finalmente, el documento resultante tras el despliegue del árbol con los elementos sustituidos es:

```
<html>
<body> Bienvenidos a XSLT </body>
</html>
```

## 2.3 Estructura de un documento hoja de estilo XSLT

En un documento XSLT existen básicamente dos categorías de elementos: los llamados elementos de alto nivel encargados de ejecutar una tarea específica y las instrucciones de plantilla que definen las condiciones para que se efectúe cada transformación.

- xsl:include que permite referenciar plantillas procedentes de una fuente externa.
- xsl:output que proporciona métodos para la salida (xml, html, text).
- xsl:strip-space que elimina antes del procesamiento todos los nodos consistentes en espacios en blanco.

Estas instrucciones son las encargadas de describir las reglas que especifican la transformación a aplicar a los elementos del documento fuente.

- <xsl:template match=" ... "> utilizada para definir las reglas de transformación para los nodos que coinciden con la expresión XPath (que se expresa entre comillas) que viene dada en el atributo match.

Por ejemplo, si una hoja de estilo quiere declarar que cada vez que se encuentre un elemento 'NOMBRE', tenga lugar alguna acción, se usa la sintaxis:

```
<xsl:template match="NOMBRE">
...
<xsl:template>
```

- <xsl:apply-templates select=" ... "> encargada de aplicar todos los patrones posibles a los elementos coincidentes con la expresión XPath que acompaña al atributo select con la que selecciona los elementos objetivo. En consecuencia, apply-templates indica al procesador que debe buscar cualquier patrón que se pueda activar, situado por debajo del nivel actual del nodo contexto. Si no se indica el atributo select, se aplicarán todas las plantillas que encajen en ese nodo.

**NOTA DE INTERÉS**

Es importante prestar atención a la diferente sintaxis de las estructuras xsl:template y xsl:apply-templates. Mientras que xsl:template usa el atributo match para indicar qué elementos encajan con la plantilla, xsl:apply-templates usa el atributo select para indicar qué plantillas aplicar en ese punto.

La combinación de estas dos instrucciones es clave para el procesamiento XSLT, ya que previamente a cualquier transformación, la parte del documento fuente XML que contiene la información que se va a pasar al documento resultado debe ser copiada y por ello ésta debe seleccionarse con una expresión XPath.

Por ello, a la sección del documento seleccionada por match se la llama genéricamente un nodo XSLT y, en el caso de que lo que vaya a seleccionarse sea el documento completo, el nodo es su raíz, para lo que se usa de acuerdo con Xpath:match="/", donde la barra /, indica al procesador que este nodo se aplica al nivel de la raíz del documento:

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

**EJEMPLO PRÁCTICO**

Alberto trabaja como programador en el departamento de informática de una empresa y le han pedido que muestre el funcionamiento de xsl:apply-templates.

¿Cómo será el proceso?

**Solución**.

Alberto muestra el proceso de la siguiente manera:

La plantilla raíz transforma cada nodo libro en la correspondiente plantilla libro. Esta última, simplemente obtiene el título. Por tanto, el resultado será una lista de todos los títulos de los libros contenidos en el archivo fuente será:

```
<!-- Plantilla raíz -->
  <xsl:template match="/">
    <xsl:apply-templates select="libreria/libro" />
  </xsl:template>

  <!-- Plantilla libro -->
  <xsl:template match="/librería/libro">
    <xsl:value-of select="titulo" />
</xsl:template>
```

**VÍDEO DE INTERÉS**

Aquí podrás ver como José Luis explica cómo crear un XSLT con XML:

# 3. CREACIÓN Y UTILIZACIÓN DE PLANTILLAS. HERRAMIENTAS Y DEPURACIÓN

*En la creación del software que estás desarrollando con XML, deberás incorporar el uso de plantillas para conseguir optimizar el tratamiento de datos y convertirlo en un modo más dinámico, además de comprobar la corrección del lenguaje mediante algún tipo de herramienta que depure el código.*

Las plantillas XML ofrecen una forma más dinámica a la hora de tratamiento de información, gracias a la posibilidad de adaptar sus campos a la información que se va introduciendo, permitiendo la creación de bloques, con instrucciones condicionales que permiten, por ejemplo, mostrar o no determinados bloques.


Plantillas XML.
Fuente: https://blog.gladtolink.com/2023/07/27/plantillas-xml-diseno-de-pagina-2/

En la creación y utilización de plantillas, los pasos a seguir serían:

1. Creación de documento XML. Puede utilizarse cualquier editor de texto, como por ejemplo Notepad++.

2. Comprimir el archivo XML en un ZIP.

3. Enlazar el archivo XML, de modo que los campos de la plantilla coincidan con las etiquetas del documento.

4. Configurar la visibilidad o no de los bloques mediante condicionales.

En este proceso existen cuestiones que deben ser tenidas en cuenta y que hacen referencia a la manipulación de datos en el documento XSLT, la utilización de instrucciones iterativa y el uso de estructuras condicionales como vamos a ir detallando a continuación.

## 3.1 Manipulación de datos

El objetivo de estas instrucciones consiste en extraer datos de los nodos fuente para poder procesarlos antes de incluirlos en el documento resultado. Si con las instrucciones anteriores, se generaba un archivo con otro formato, como HTML, con estas, se puede generar otro XML pero con una estructura distinta a la del original.

Como ejemplo de la funcionalidad de este grupo de instrucciones, digamos que XSLT, en vez de proporcionar un patrón para cada elemento de un documento, actúa proporcionando un elemento que duplica los nodos desde el árbol fuente al árbol resultado. Por ello, hay que prestar especial atención al elemento <xsl:copy> que, además de producir una copia del nodo contexto, lo ubica en el árbol resultado.

Así, tomando como fuente el archivo intro.xml ya visto y la hoja de estilo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:apply-templates match = "miMensaje">
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy>
</xsl:apply-templates>
<xsl:apply-templates match = "mensaje">
<xsl:copy>
&apos;Hola Mundo&apos; para variar
</xsl:copy>
</xsl:apply-templates>
</xsl:stylesheet>
```

Se observa que, gracias a las instrucciones:

```
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy>
```

Se produce un duplicado del nodo contexto en el árbol resultado. Y al aparecer, además:

```
<xsl:copy>
&apos;Hola Mundo&apos; para variar
</xsl:copy>
```

El elemento copy, reemplaza el contenido del elemento con texto.

El resultado de la transformación de intro.xml es:

```
<?xml version = "1.0"?>
<miMensaje>
<mensaje>'Hola Mundo' para variar</mensaje>
</miMensaje>
```

Obviamente, lo más habitual es la copia de un subárbol, a partir de un nodo determinado específicamente para ello.

```
<xsl:copy-of select="...">
```

Que devuelve el conjunto de nodos completo correspondiente a la expresión XPath indicada es el atributo select, que duplica en el resultado los nodos seleccionados.

Al contrario del elemento copy, el elemento copy-of duplica tanto todos los hijos (texto, instrucciones de procesamiento, comentarios, etc.) como los atributos del nodo.

Otras instrucciones de manipulación de uso frecuente son:

- <xsl:value-of select="..."> que devuelve bien el valor del atributo indicado, bien el texto asociado con el nodo proporcionado (nótese que los atributos deben ir precedidos de @).

- <xsl:sort . . . > indica los criterios de ordenación para el conjunto de nodos que se está procesando mediante otras instrucciones.

**¿SABÍAS QUE...?**

El elemento <xsl:copy> genera una copia nodo actual al documento de salida, a nivel superficial, del nodo y cualquier nodo del nombre de espacio asociado. No lo hará ni de los elementos hijo ni de los atributos del nodo actual.

## 3.2 Instrucciones iterativas

Una de las principales ventajas de XSL, frente a los CSS consiste en la posibilidad de efectuar operaciones de control sobre el documento. Para ello, consta de instrucciones XSLT que permiten realizar tanto iteraciones como estructuras condicionales.

Una estructura iterativa se representa en XSLT de la siguiente forma:

```
<xsl:for-each select="...">
```

Cuyo significado es aplicar las reglas que constituyen el cuerpo de la instrucción, a cada uno de los elementos que coincidan con la expresión XPath que acompaña a select. En otras palabras, el elemento for-each es un bucle que procesa todos los nodos que indica select.

**ENLACE DE INTERÉS**

Aquí encontrarás más información acerca de la presentación y transformación de documentos XML, mediante CSS y transformaciones XSLT:

Por ejemplo, se tiene un archivo fuente con un elemento 'GENTE' que contiene una lista de subelementos 'PERSONA', que representan personas, si xsl:for-each se usa con la expresión XPath 'GENTE/PERSONA', se procesarían todos los elementos 'PERSONA' presentes en el elemento 'GENTE'. Para obtener un listado de personas, después de este for-each, se podría poner la siguiente expresión:

```
<xsl:value-of select="NOMBRE"/>
```

El elemento xsl:for-each iría iterando por todas las personas y el elemento xsl:value-of obtendría el valor almacenado en el elemento 'NOMBRE' de cada una de ellas, con lo que se obtendría un listado de los nombres de todas las personas almacenadas en el archivo.

XSLT permite iterar a través de un conjunto de nodos de forma que los nodos también pueden ordenarse, si ello fuera necesario. Para ello, se usa la expresión <xsl:sort select="…" order="…"/>. Se trata del elemento XSLT sort, que ordena los nodos seleccionados por un elemento for-each por el contenido del campo indicado en select, de forma ascendente o descendente, indicando order="ascending" o order="descending", respectivamente.

**EJEMPLO PRÁCTICO**

Luisa trabaja como programadora en el departamento de informática de una empresa y le han pedido que a partir del siguiente documento uso.xml, utilice una hoja de estilo uso.xsl que, por medio de una estructura iterativa, obtenga un listado ordenado de los capítulos del libro, incluyendo entre paréntesis el número de páginas de las que consta cada uno de ellos.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml:stylesheet type="text/xsl" href="uso.xml"?>
<libro isbn = "999-99999-9-X">
<titulo>El primer libro de XML</titulo>
<autor>
<nombre>José</nombre>
<apellido>Pérez</apellido>
</autor>
<capitulos>
<preface num="1" paginas="2">Bienvenidos</preface>
<capitulo num="1" paginas="4">Introducción a XML</capitulo>
<capitulo num="2" paginas="2">Elementos XML</capitulo>
<apendice num="1" paginas="9">Entidades</apendice>
</capitulos>
</libro>
```

¿Cómo será el código?

**Solución:**

Luisa deberá crear el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:for-each select="capitulos/capitulo">
<xsl:sort select="@num" order="ascending"/>
<tr>
<td align="right">
capítulo <xsl:value-of select="@num"/>
</td>
<td>
<xsl:value-of select="."/> ( <xsl:value-of select="@paginas"/> paginas )
</td>
</tr>
</xsl:for-each>
```

## 3.3 Estructuras condicionales

XSLT también proporciona elementos para realizar procesamiento condicional, tal como hace una sentencia if en programación tradicional. Se trata del elemento:

```
<xsl:if test="…">
```

**NOTA DE INTERÉS**

Es importante tener en cuenta que si las expresiones condicionales contienen comparaciones de tipo > o <, éstas deben ser expresadas mediante las entidades > y < respectivamente, ya que, si no, se interpretarían como elementos de apertura o cierre de etiquetas.

Al igual que en programación, esta estructura, permite escribir una regla que aplica un determinado patrón sólo si la expresión indicada en el atributo test se evalúa como cierta.

Otra estructura condicional es xsl:choose, que representa un condicional múltiple, similar a la estructura switch de programación. Hace uso de los elementos xsl:when y xsl:otherwise. El primero de ellos para especificar cada uno de los casos, y el segundo como caso por defecto. Por tanto, la estructura de xsl:when, será similar a la de xsl:if, expresando la condición mediante el atributo test. En conjunto, la estructura choose, sería:

```
<xsl:choose>
<xsl:when test="…">
…
</xsl:when>
<xsl:when test="…">
…
</xsl:when>
…
<xsl:otherwise>
</xsl:otherwise>
</xsl:choose>
```

**EJEMPLO PRÁCTICO**

Juan Carlos trabaja como programador en el departamento de informática de una empresa y le han pedido que a partir de un documento fuente agenda.xml, se organicen citas y tareas conociendo fecha, hora y tipo de tarea, utilizando una estructura choose, indicará si la cita es en horario de mañana, tarde o noche, en función del rango de horas.

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
<anyo value="2018">
<fecha mes="11" dia="15">
<nota hora="1430">Visita al médico</nota>
<nota hora="1620">Clase de inglés</nota>
</fecha>
<fecha mes="12" dia="4">
…
</fecha>
<fecha mes="12" dia="20">
…
</fecha>
</anyo>
</agenda>
```

¿Cómo será el código?

**Solución**.

Juan Carlos deberá crear el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:choose>
<xsl:when test="@hora &gt;= '0500' and @hora &lt; '1200' ">
Ma&#241;ana (xsl:value-of select="@hora"/>);
</xsl:when>
<xsl:when test="@hora &gt;= '1200' and @time &lt; '1700' ">
Tarde (xsl:value-of select="@hora"/>);
</xsl:when>
<xsl:when test="@hora &gt;= '1700' and @hora &lt; '2359' ">
Tarde-Noche (xsl:value-of select="@hora"/>);
</xsl:when>
<xsl:when test="@hora &gt;= '0100' and @hora &lt; '0500' ">
Noche(xsl:value-of select="@hora"/>);
```

```
</xsl:when>
 <xsl:otherwise>
Todo el d&#237;a.
</xsl:otherwise>
</xsl:choose>
<xsl:value-of select="."/>
<xsl:if test=".= ' ' ">n/a</xsl:if>
</xsl:stylesheet>
```

## 3.4 Herramientas y depuración

Cuando lo que queremos es crear y validar documentos XML o esquemas, podemos hacerlo mediante un editor de texto plano y un navegador web, pero, también disponemos de una serie de herramientas que nos facilitarán la tarea, como son los siguientes productos de escritorio:

- Notepad ++ (Gratuito)
- Editix XML Editor (Gratuito)
- XML Copy Editor (Gratuito)
- NetBeans (Gratuito)

También tenemos herramientas de validación online incluso, algunas páginas web que nos validan un xml, dtd y schema:

XML Validation

XSD Validation

Los depuradores van a proporcionar información sobre los datos contenidos en los documentos y plantillas XML, facilitando la localización y corrección de los posibles errores existentes, con acciones que se pueden ejecutar paso a paso, pudiendo intervenir en la ejecución para pausar y reanudar, e incluso detener.

El análisis de la salida que se genera se obtiene mediante herramientas como las que hemos citado anteriormente, como es el caso de Editix XML, que incorporan un depurador.

Permiten la creación de un documento XML, mediante la incorporación de elementos de software, desde otro documento XML gracias a la aplicación de la hoja de estilos XSLT.

# 4. CONVERSIÓN ENTRE DIFERENTES FORMATOS DE DOCUMENTOS

*Dentro de tus cometidos como programador, a la hora de utilizar XML para la creación de los documentos de la aplicación a desarrollar, tienes que conocer la utilidad que existe para la conversión entre diferentes formatos desde o hasta XML, por lo que pondrás en práctica algún tipo de conversión para conocer cómo realizarlas.*

Cuando hablamos de conversión de documentos XML, tenemos que hacer referencia a XSL (eXtensible Stylesheet Language) o lenguajes de hojas de estilo extensible.

La conversión de un documento XML a otro formato que pueda ser representado mediante cadenas de caracteres, ofreciendo alternativas de otros lenguajes como HTML, formatos como CSV, o la utilización de incluso sentencias de SQL, dentro de otros tipos de lenguajes de programación.

Dentro XSL encontramos técnicas de transformación de documentos, como las que hemos tratado en el apartado 1 de esta unidad didáctica, entre las que hemos destacado XSLT, con el que podemos definir el modo de transformación de un documento XML en otro, o Xpath, que permite el acceso a los distintos componentes que forman los documentos XML.

En este último caso concreto de Xpath, deberemos tener en cuenta una serie de cuestiones como son:

1) Conocer lo que representa cada uno de sus términos básicos que resumimos a continuación:

| | |
|---|---|
| **Nodo raíz** | Contiene el fichero XML, identificado por "/" |
| **Nodos elemento** | Cada elemento del documento XML |
| **Nodos texto** | Caracteres del documento XML sin etiqueta |
| **Nodos atributo** | Etiquetas añadidas al nodo elemento |
| **Nodo comentario** | Generados para los elementos con comentarios |
| **Nodo actual** | Es el referido al evaluar una expresión Xpath |
| **Nodo contexto** | Conjunto de subexpresiones evaluadas sucesivamente |
| **Tamaño de contexto** | Número de nodos evaluados en una expresión Xpath |

2) Los resultados obtenidos que serán de los tipos node-set (nodos no ordenados), booleano, número y cadena.

3) La ruta de localización que nos puede devolver un conjunto de nodos, disponiendo de distintos tipos de rutas de localización como son de: Nodo raíz del documento, un elemento, atributos, espacios de nombres, comentarios, texto o instrucciones de procesamiento.

En otro de los casos como XSLT, el documento XML es el contenedor de los datos estructurados, delimitados por etiquetas con el significado que le otorga el creador del documento, el cual engloba los elementos, atributos y dependencias jerárquicas.

La transformación del documento XML, requiere hacer una selección de qué contenido se quiere transformar con Xpath, realizando la transformación ya con XSLT en un fichero con extensión .xsl.

**VÍDEO DE INTERÉS**

Aquí podrás visualizar cómo transformar un XML a HTML:

Permite generar documentos en distintos formatos desde un documento XML, para convertirlo en otro de tipo XML, HTML, texto, pdf, etc. Que se obtienen al pasar el documento con extensión .xsl junto al documento .xml, al procesador XSLT, generándose como salida el nuevo documento.

El procesador XSLT, compuesto por aplicaciones que se encargan de la transformación, en primer lugar, lee el documento XML, realiza una representación mediante un árbol de nodos (elementos, atributos, etc.), que recorre secuencialmente, realizando a continuación, el proceso de transformación.


XSLT.
Fuente: https://www.javatpoint.com/xslt-tutorial

La hoja de estilos XSLT, puede estar alojada tanto en un servidor web, en el propio cliente web o, en una aplicación de terceros encargada del proceso de transformación.

En la utilización de este tipo de tecnologías en la conversión y transformación de documentos en diferentes formatos, resulta de interés conocer cómo utilizar instrucciones de diseño, definición de tipos y combinar hojas de estilo.

## 4.1 Instrucciones de diseño

Estas instrucciones permiten la creación en el documento resultado de nuevos elementos y atributos, así como instrucciones de proceso y comentarios. Para facilitar esta tarea de diseño, la recomendación XSLT define plantillas por defecto, de forma que el fragmento:

```
<xsl:template match="/|*">
<xsl:apply-templates/>
</xsl:template>
```

Hace una correspondencia entre el elemento raíz del documento ( / ) y cualquier nodo elemento ( * ) de un documento, aplicando las plantillas a sus nodos hijos.

Resulta fácil de comprobar que:

```
<xsl:template match = "text() | @*">
<xsl:value-of select = "."/>
<xsl:template>
```

Obtiene la correspondencia para nodos texto (text()) o nodos atributos (@) dando como salida sus valores respectivos.

Asimismo, para los nodos nombres de los elementos y atributos existen las sentencias:

```
<xsl:element name="…">
```

Y:

```
<xsl:attribute name="…">
```

Mientras que:

```
<xsl:template match = "processing-instruction() | comment() "/>
```

Realiza la correspondencia entre nodos de instrucciones de proceso y nodos comentarios, aunque no efectúa ninguna acción sobre ellos.

**EJEMPLO PRÁCTICO**

Eva trabaja como programadora en el departamento de informática de una empresa y le han pedido que a partir de un documento deportes.xml, cree un nuevo documento, también XML pero con una estructura distinta.

```
<?xml version = “1.0”?>
<deportes>
<juego titulo = “baloncesto”> <id>243</id>
<para> Más popular en algunos Paises </para>
</juego>
<juego titulo = “baseball”> <id>431</id>
<para>El más conocido en América</para>
</juego>
<juego titulo = “futbol”> <id>123</id>
<para>El deporte más conocido en todo el mundo</para>
</juego>
</deportes>
```

¿Cómo será el código?

**Solución:**

Eva deberá crear el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="/">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="deportes">
<deportes>
<xsl:apply-templates/>
</deportes>
</xsl:template>
<xsl:template match="juego">
<xsl:element name="{@titulo}">
<xsl:attribute name="id">
<xsl:value-of select="id"/>
</xsl:attribute>
<comentario>
<xsl:value-of select="para"/>
</comentario>
</xsl:element>
```

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

## 4.2 Definición de tipos

El lenguaje, permite crear variables de diversos tipos, en función del contenido que se necesite almacenar. Dichas variables se podrán usar en las estructuras vistas para realizar cálculos. Los tipos son:

- Variables de vínculos de datos (xsl:variable)
- Texto normal (xsl:text)
- Números (xsl:number).

El nombre de las variables se asigna mediante el atributo name y su valor mediante el atributo select:

```
<xsl:variable name="var1" select="…"/>
```

Para acceder al valor de una variable, se usa su nombre precedido del símbolo $.

**EJEMPLO PRÁCTICO**

Rosa trabaja como programadora en el departamento de informática de una empresa y le han pedido que a partir del siguiente documento uso.xml, utilice una hoja de estilo uso.xsl que calcule el número total de páginas del libro.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml:stylesheet type="text/xsl" href="uso.xml"?>
<libro isbn = "999-99999-9-X">
<titulo>El primer libro de XML</titulo>
<autor>
<nombre>José</nombre>
<apellido>Pérez</apellido>
</autor>
<capitulos>
<preface num="1" paginas="2">Bienvenidos</preface>
<capitulo num="1" paginas="4">Introducción a XML</capitulo>
<capitulo num="2" paginas="2">Elementos XML</capitulo>
<apendice num="1" paginas="9">Entidades</apendice>
</capitulos>
</libro>
```

¿Cómo será el código?

**Solución**.

Rosa deberá crear el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="/">
<total>
Numero de páginas = <xsl:variable name="suma"
            select="sum(libro/capitulo/*/@paginas"/>
<xsl:value-of select="$suma"/>
</total>
</xsl:template>
</xsl:stylesheet>
```

En el código, la estructura:

```
<xsl:variable name="suma" select="sum(libro/capitulo/*/@paginas"/>
```

Crea un elemento variable con el atributo suma (que guarda la suma del número de páginas del libro) y el atributo select con el valor sum(libro/capitulo/*/@paginas), una expresión Xpath que suma los números de los atributos páginas de los elementos capitulo.

A continuación, en <xsl:valueof select="$suma"/>, el elemento value-of da como salida el valor de la variable suma.

## 4.3 Combinar hojas de estilo

XSLT facilita la modularidad a la hora de manejar sus hojas de estilo, lo que permite tanto importar como incluir, en un documento XSLT, otros documentos XSLT. La diferencia entre estos dos elementos con capacidad de combinación, include e import, reside en que las plantillas incorporadas por medio del elemento include tienen la misma prioridad que las plantillas locales, en cuyo caso, si una plantilla se duplicara, la que actúa es la última utilizada.

**PARA SABER MÁS**

Aquí encontrarás más información sobre que es XSLT, transformaciones, estructura, sintaxis y elementos:

**EJEMPLO PRÁCTICO**

Sebastián trabaja como programador en el departamento de informática de una empresa y le han pedido que a partir del siguiente documento uso.xml, use include para incorporar otros elementos XSLT en un documento XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml:stylesheet type="text/xsl" href="uso.xml"?>
<libro isbn = "999-99999-9-X">
<titulo>El primer libro de XML</titulo>
<autor>
<nombre>José</nombre>
<apellido>Pérez</apellido>
</autor>
<capitulos>
<preface num="1" paginas="2">Bienvenidos</preface>
<capitulo num="1" paginas="4">Introducción a XML</capitulo>
<capitulo num="2" paginas="2">Elementos XML</capitulo>
<apendice num="1" paginas="9">Entidades</apendice>
</capitulos>
</libro>
```

¿Cómo será el código?

**Solución**.

Sebastián deberá crear el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="/">
<html>
<body>
<xsl:apply-templates select="libro"/>
</body>
</html>
</xsl:template>
<xsl:template match="libro">
<h2>
<xsl:value-of select="titulo"/>
</h2>
<xsl:apply-templates/>
</xsl:template>
```

```
<xsl:include href="autor.xsl"/>
<xsl:include href="capitulos.xsl"/>
<xsl:template match="*|text()"/>
</xsl:stylesheet>
```

Donde <xsl:include href="autor.xsl"/> <xsl:include href="capitulos.xsl"/> muestra la inclusión de los archivos referenciados por el atributo href. A continuación, se muestran los documentos XSLT, autor.xsl y capitulos.xsl que tienen el objetivo de representar el nombre del autor y los nombres de los capítulos conjuntamente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="autor">
<p>Autor:
<xsl:value-of select="Nombre"/>,<xsl:value-of select="Apellido"/>
</p>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="capitulos">
Capitulos :
<ul>
<xsl:apply-templates select="capitulo"/>
</ul>
</xsl:template>
<xsl:template match="capitulo">
<li><xsl:value-of select="."/></li>
</xsl:template>
</xsl:stylesheet>
```

# RESUMEN FINAL

En esta unidad se ha tratado el tema de la conversión y adaptación de documentos para el intercambio de información, en concreto en el caso de creación de documentos XML.

Comenzando por un recorrido por las distintas tecnologías de transformación de documentos que se ajustan a sus respectivos estándares, entre las que destacan XSL, XSLT o XPath, entre otras, en unos ámbitos y aplicación que van desde el desarrollo y diseño de aplicaciones, páginas y sitios web, hasta aplicaciones para móviles en los diferentes sistemas operativos como Android e iOS, o intercambio de información compartida, con formato JSON, por ejemplo.

Aspecto importante es conocer la descripción de su estructura y de la sintaxis correcta a utilizar, en el que partiendo de un documento XML inicial, obtendremos un nuevo documento XSLT, donde la corrección en la estructura y sintaxis para el modelado de procesado XSLT con la implementación de patrones o plantillas, la escritura de expresiones regulares para recorres y procesar el documento XML, mediante XPath, , además de conocer cómo será la estructura de un documento hoja de estilo XSLT, con sus dos categorías de elementos xsl:include y xsl:output.

Dada la importancia de la creación de plantillas, se ha dedicado un apartado concreto, con los pasos a seguir, poniendo como ejemplo Notepad ++, detallando las acciones necesarias como son la manipulación de datos para la extracción de información de los nodos fuente, la utilización de instrucciones iterativas, como ventaja ante las CSS, que permiten realizar operaciones de control sobre el documento, y, por último, el empleo de estructuras condicionales mediante la sentencia “if”, para el control y selección de bloques dentro del documento XML.

Para la creación y validación de los documentos XML, disponemos de una serie de herramientas como son Notepad++, Editix XML Editor o NetBeans, entre otras, que además suelen disponer de un depurador integrado. Igualmente, destacamos la existencia de ciertos sitios web que permiten la validación online del código como es el caso de http://www.xmlvalidation.com .

Para finalizar, hemos tratado la conversión entre diferentes formatos de documentos, con referencia obligada a XSL (eXtensible Stylesheet Language) o lenguajes de hojas de estilo extensible, cuya familia incluye las ya tratadas en la unidad didáctica XSLT y XPath como técnicas de transformación de documentos XML, que pueden ser a otros formatos como HTML, CSV o incluso sentencias SQL, entre otros. Prestando especial atención a las instrucciones de diseño que permiten un nuevo documento con nuevos elementos y

atributos, la creación de los diferentes tipos de variables a utilizar y la posibilidad de combinar hojas de estilo.