4.2. Transformación de contenidos

Índice

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer los conceptos de transformación de documentos XML.
- Conocer cuáles son los lenguajes de transformación que hay en el mercado como estándares.
- Conocer el lenguaje de transformación XSL.
- Conocer el lenguaje de transformación XPath.
- Aprender a realizar transformaciones con los estilos XSLT.

Introducción

Concepto

Los documentos XML proporcionan información estructurada pero que no tiene formato de salida. Eso quiere decir que, si queremos presentar esa información con un formato determinado, deberíamos transformar el documento en otro al que se le aplique el formato deseado.

Las tecnologías que podemos usar **para realizar esta transformación** son las siguientes:

- **XSLT:** (eXtensible Stylesheet Language Transformation) sirve para definir cuál será el modo de transformar un documento XML en otro.
- **XSL-FO:** (eXtensible Stylesheet Language - Formatting Object) se utiliza para transformar XML en un formato legible e imprimible por una persona, por ejemplo en un documento PDF.
- **Xpath:** permite el acceso a los componentes de un documento XML.

Las transformaciones posibles son:

- **Desde XML hacia XML:** usado para transformar documentos XML en otros con información añadida, borrada u organizada de otra forma.
- **Desde XML hacia otros formatos:**
    - **XML hacia XHTML:** a través de estilos y los lenguajes XPath y XSLT.
    - **XML hacia formatos estándares, como PDF, SVG, RTF u otros:** a través del lenguaje de transformación XSL-FO.

Lenguajes de transformación

Los documentos XML definen datos cuando se usan como contenedores de estructuras complejas de datos.

En su sintaxis no hay manera de definir el formato de salida de datos, por lo que se deben usar otros estándares para que dichos datos puedan ser transformados de una manera formateada.

Alguna vez puede ser necesario cambiar el árbol de datos de un XML para transformarlo en otro que tenga los mismos datos, pero con una estructura distinta.

Los **lenguajes de transformación de XML** son muchos. Veamos algunos ejemplos:

XSLT

Es un lenguaje para transformar documentos XML en otros documentos XML o en otros formatos, como HTML para páginas web, texto plano, o XSL Formatting Objects.

XQuery

Es un lenguaje de consulta y programación funcional que transforma las consultas y colecciones de datos estructurados y no estructurados, por lo general en forma de XML, en texto y con extensiones específicas del proveedor para otros formatos de datos (JSON, binario, etc.).

XProc

Es una recomendación W3C para definir un lenguaje de transformación XML que define tuberías XML.

XML document transform

Un lenguaje de transformación XML de Microsoft. Es un lenguaje de programación diseñado específicamente para transformar una entrada de un documento XML en una salida de documento que satisface alguna meta específica.

STX

Es un lenguaje de transformación XML que pretende tener alta velocidad, bajo consumo de memoria y ser una alternativa a la versión 1.0 y 2.0. de XSLT.

XML script

Permite la creación, el almacenamiento y la manipulación de variables y datos durante el procesamiento. XML es un lenguaje de marcado para documentos que contienen tanto el contenido (es decir, texto, imágenes, etc.) y alguna indicación de lo que desempeña el papel de contenido (por ejemplo, si está en un encabezado de sección o un pie de página, etc.).

FXT

Es una herramienta de transformación XML funcional, implementado en el estándar ML.

XDuce

Es un lenguaje con una sintaxis sencilla. Está escrito en ML.

CDuce

Extiende XDuce a usos de propósito general como lenguaje de programación funcional.

XACT

Es un sistema basado en Java para las transformaciones XML de programación. Como características notables incluye plantillas XML como valores inmutables y un análisis estático para garantizar la seguridad de tipos utilizando los tipos de esquema XML.

XFun

Es un lenguaje funcional para definir transformaciones entre los árboles de datos XML.

XStream

Es un lenguaje de transformación funcional para documentos XML basados en CAML. Las transformaciones se pueden producir de manera masiva.

HaXml

Es una biblioteca y una colección de herramientas para escribir transformaciones XML en Haskell. Su enfoque es muy consistente y potente.

Scala

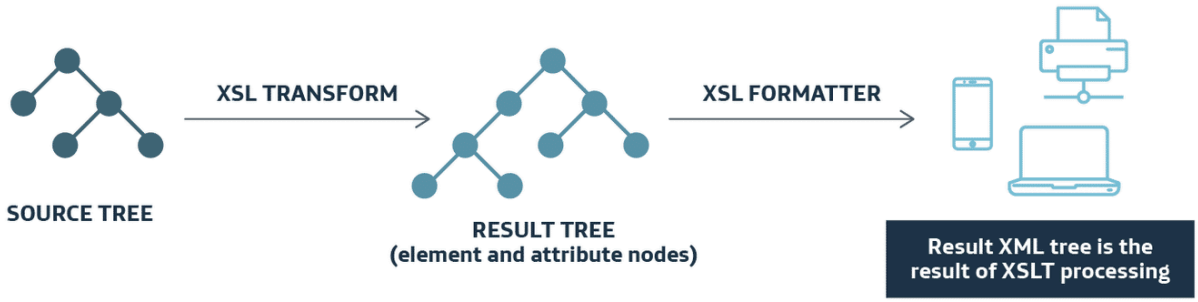
Un lenguaje funcional y orientado a objetos, de uso general, con un apoyo específico para la transformación de XML en forma de XML, coincidencia de patrones, literales y expresiones, junto con las bibliotecas XML estándar.

XLS

Lenguajes de transformación: XSL

Una hoja de estilo XSL especifica la presentación de una clase de documentos XML mediante la descripción de cómo una instancia de la clase se transforma en un documento XML que utiliza el vocabulario de formato.

XSL es un Lenguaje Extensible de Hojas de Estilo, cuyo objetivo principal es mostrar cómo debería estar estructurado el contenido, cómo debería ser diseñado el contenido de origen y cómo debería ser paginado en un medio de presentación, como puede ser una ventana de un navegador web, un dispositivo móvil, un conjunto de páginas de un catálogo, un informe o un libro.



En la construcción del árbol el proceso de transformación también añade la información necesaria para el formato de salida de dicho árbol.

Un **procesador XSL** de hoja de estilo acepta un documento XML y una hoja de estilo XSL y **produce la presentación de dicho contenido** de código XML que se pretendía por el diseñador de ese estilo.

Hay dos aspectos de este proceso de presentación:

- En primer lugar, la construcción de un árbol de resultados desde el árbol de código fuente XML.
- En segundo lugar, interpretar el árbol de resultados para producir resultados con formato adecuado para su presentación en una pantalla, en papel, o en otros medios.

El primer aspecto se denomina "transformación árbol" y el segundo se llama "formateo". El proceso de formateo es realizado por el formateador, que puede ser simplemente un motor de representación dentro de un navegador.

Beneficios XSL

A diferencia del caso de HTML, los nombres de los elementos de XML no tienen presentación semántica intrínseca.

A falta de una hoja de estilo, **un procesador no podría saber cómo representar el contenido de un documento XML** que no sea como una cadena de caracteres no diferenciado.
XSL proporciona un modelo integral y un vocabulario para escribir hojas de estilo usando la sintaxis XML.

Otras ventajas de XSL son:

- Mantener un formato de salida de un documento independientemente de su estructura y de cualquier otra especificación.
- Reutilizar el mismo estereotipo de formato para crear diversos documentos.
- Favorecer la automatización de tareas, ya que es compatible con lenguajes de programación, especialmente con ECMAScript.

Hojas de estilo

Del mismo modo que aplicamos estilos a los documentos HTML, con los documentos de datos XML podemos hacer lo mismo si queremos mejorar su visualización en un explorador. Siendo fácil de aprender y utilizar, su desventaja es que solo sirve para visualizar documentos en un navegador.

Para usar estilos CSS en un documento XML añadimos la siguiente línea:

```
<?xml-stylesheet type="text/css" href="fichero.css" ?>
```

Fichero.css es el nombre del archivo CSS que contiene las reglas de formato.

El fichero con los estilos CSS seguirá las mismas normas que aprendimos en la visualización de documentos, aplicando las diferentes reglas a las etiquetas.

Ejemplo

Vamos a aplicar estilos a un documento XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/css" href="estilos.css" ?>
<alumnos>
      <curso>
            <alumno>
                  <nombre>Juan</nombre>
                  <nota>9</nota>
                  <faltas>1</faltas>
            </alumno>
            <alumno>
                  <nombre>Luisa</nombre>
                  <nota>10</nota>
                  <faltas>0</faltas>
            </alumno>
            <alumno>
                  <nombre>Sergio</nombre>
                  <nota>5</nota>
                  <faltas>7</faltas>
            </alumno>
            <alumno>
                  <nombre>María</nombre>
                  <nota>8</nota>
                  <faltas>3</faltas>
            </alumno>
            </curso>
</alumnos>
```

Documento XML.

En el ejemplo, además de ver la estructura de datos típica en los documentos XML, podemos observar cómo se realiza la llamada a un archivo CSS:

<?xml-stylesheet type="text/css" href="estilos.css" ?>

En este archivo aplicamos las reglas para definir su visualización en el navegador.

```
alumno{
      padding: 10px;
      display: block;
      overflow: hidden;
}

alumno:nth-child(2n){
      background-color: #ccc;
}
```

```
nombre, nota, faltas{
	font-family: arial;
	font-size: 18px;
	width: 80px;
	border-right: 2px solid #aaa;
	display: block;
	float: left;
	text-align: center;
}

nota{
	font-size: 22px;
	color:green;
}

faltas{
	font-size: 22px;
	color:red;
}
```

Archivo *estilos.css*.

El resultado al abrir el documento en un navegador, después aplicar las reglas CSS, será:

XPATH

Concepto

XPath debe su nombre al uso de una notación de caminos o *path*, similar al de las URL para **navegar a través de la estructura** jerárquica de un documento XML.

XPath es un lenguaje de direccionamiento de partes o piezas de un documento XML, diseñado para ser **utilizado tanto por XSLT como por XPointer**. También tiene la capacidad de manipulación de cadenas de caracteres, números y booleanos.

XPath utiliza una sintaxis compacta, **no XML**, para facilitar el uso de XPath de los valores de los atributos en las URI y en XML.

XPath opera sobre la estructura lógica de un documento XML, en lugar de operar en su sintaxis. XPath es un elemento importante en el estándar XSLT.

XPath navega por los nodos que un XML genera. Estos nodos suelen ser de elementos, de atributos y de texto.

La forma de acceder a los nodos del árbol generado es a partir de expresiones. Una expresión XPath es un predicado que se aplica sobre una estructura de árbol correspondiente a la jerarquía de los datos de un documento XML y devuelve todo lo que encaja con ese predicado.

```
/tienda/lista[1]
```

Selecciona el primer elemento *lista* que es hijo del elemento *tienda*.

Características

XPath utiliza expresiones de ruta para seleccionar nodos o conjuntos de nodos en un documento XML. Estas expresiones de ruta se parecen mucho a las expresiones que se utilizan en el sistema de archivos de un ordenador.

Una ruta de ubicación se puede utilizar como una expresión. La expresión devuelve el conjunto de nodos seleccionados por la ruta.

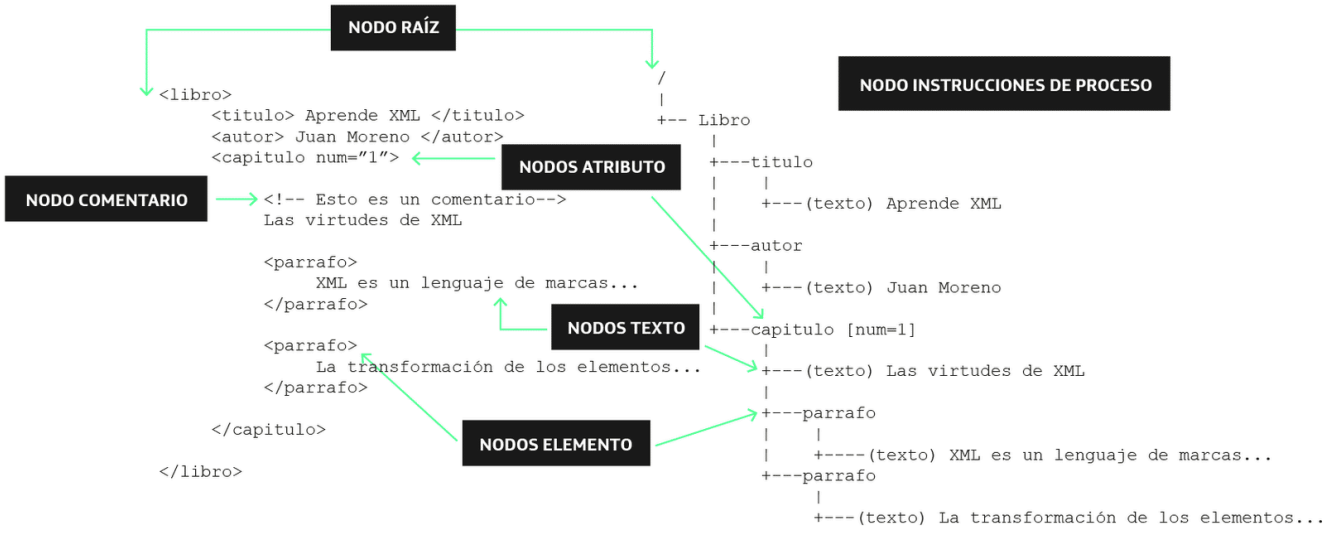
El operador **"|"** calcula la unión de los operandos, que deben ser conjuntos de nodos. Los predicados se utilizan para filtrar expresiones, de la misma manera que se utilizan en las rutas de ubicación.

- XPath es una sintaxis para definir partes de un documento XML.
- XPath utiliza expresiones de ruta para navegar en documentos XML.
- XPath contiene una biblioteca de funciones estándar.
- XPath es un elemento importante en el estándar XSLT.
- XPath es una recomendación del W3C.

Elementos de XPath

Vamos a estudiar los elementos con los que trabaja XPath a nivel de objetos. En el ejemplo vemos la creación de árbol y definición de los elementos.

Puedes ampliar la información sobre XML Path Language (XPath).



Nodo comentario

Se pueden integrar comentarios dentro del documento de la siguiente manera:
<!--esto es un comentario -->

En el árbol también se generan nodos para cada nodo con comentarios.

Nodos atributo

Cada nodo elemento puede tener o tiene asociado un conjunto de nodos atributo; el elemento es el padre de cada uno de estos nodos atributo.

Sin embargo, un **nodo atributo no es un hijo de su elemento padre**, por lo que no se consideran hijos del elemento al que están asociados, sino etiquetas añadidas al nodo

elemento. Aquellos atributos que tengan un valor asignado en el esquema asociado, se tratarán como si ese valor se le hubiese dado al escribir el documento XML.

Nodos texto

Son aquellos caracteres del documento que no están marcados con ninguna etiqueta y están agrupados en un nodo. Este tipo de nodos no tiene hijos. Cada carácter dentro de una sección CDATA se trata como dato de carácter. Los caracteres dentro de comentarios, instrucciones de procesamiento y valores de atributos, no producen nodos de texto.

Nodos elemento

Son cada uno de los elementos del documento XML. Todos ellos tienen un elemento padre (excepto el nodo raíz). Pueden tener identificadores únicos. Para ello es necesario que un atributo esté definido en una DTD o en un fichero *schema* XML asociado. Esto nos permite referenciar dicho elemento de forma mucho más directa.

Un documento no puede tener dos elementos con el mismo identificador único.

Nodo Raíz

Es el nodo que contiene toda la información XML. No debemos confundirlo con el elemento raíz del XML, pues en XPath se coloca justo antes de este, por lo que lo contiene. XPath lo identifica con "/".

En el ejemplo el elemento raíz está justo antes del nodo *<libro>*.

Nodo instrucciones proceso

Hay un nodo instrucción de procesamiento para cada una de las instrucciones, con la excepción de aquellas que se produzcan dentro de la declaración de tipo de documento.

En el árbol se generan nodos para cada nodo con comentarios y con instrucciones de proceso. Al contenido de estos nodos se puede acceder con la propiedad *string-value.*

La declaración XML no es una instrucción de procesamiento, por lo tanto, no hay ningún nodo instrucción de procesamiento correspondiente a la declaración XML.

Los predicados

Los predicados usados en XPath filtran un conjunto de nodos con respecto a un eje para producir un nuevo conjunto de nodos.

Un predicado se utiliza para filtrar una secuencia de nodos aplicando una prueba específica. La expresión de predicado se incluye entre corchetes y se enlaza con el último nodo de una expresión de ruta de acceso.

Existen dos tipos de elementos que podemos incluir en un predicado:

- Test.
- Ejes.

Test

Permiten restringir lo que devuelve una expresión Xpath. Podemos agruparlos en función de los ejes a los que se puede aplicar.

Aplicable a cualquier eje:

- **"*"**, solo devuelve elementos, atributos o espacios de nombres, pero **no permite obtener nodos de texto o comentarios de cualquier tipo**.
- ***node()***, devuelve los nodos de todos los tipos.

```
//parrafo/*
```

Selecciona todos los nodos (principales) descendientes de *parrafo*.

```
//parrafo/node()
```

Selecciona todos los nodos descendientes de *parrafo* (de cualquier tipo: texto, comentarios...).

Solo aplicables a ejes de contenido:

- ***text()***, devuelve cualquier nodo de tipo texto.
- ***comment()***, devuelve cualquier nodo de tipo comentario.
- ***processing-instruction()***, devuelve cualquier tipo de instrucción de proceso.

```
//parrafo/text()
```

Selecciona el texto de todos los nodos *parrafo*.

```
//parrafo//text()
```

Selecciona todos los textos dependientes de todos los nodos *parrafo*.

Una doble barra: // (en su forma larga descendant::) selecciona TODOS los nodos que descienden del conjunto.

Ejes

Son los que permiten seleccionar el conjunto de nodos dentro del nodo contexto que cumple un patrón, que es el que se define en la expresión. Pueden ser o no de contenido.

ancestor: devuelve todos los nodos de los que el nodo contexto es descendiente.

```
//ancestor::parrafo/ancestor::*
```

Devuelve todos los hijos que tiene el nodo *parrafo*.

```
//*[@href]/ancestor::capitulo
```

Devuelve los nodos *capitulo* que tienen hijos con el atributo *href*.

child: es el eje utilizado por defecto para devolver los hijos de un nodo. Su forma habitual es la barra, **"/"**, también puede ponerse */child::*

```
/libro/titulo
```

Devuelve los nodos *titulo* de *libro* (versión abreviada con una /).

```
/libro/child::titulo
```

Devuelve los nodos *titulo* de *libro* (sin abreviar).

attribute: permite seleccionar los atributos que deseemos. Es el único eje que no es de contenido.

```
/libro/capitulo/@ref
```

Selecciona el atributo *ref* que tengan las etiquetas *capitulo*.

```
/libro/capitulo[@ref]/*
```

Selecciona todos los nodos hijos que tengan como atributo en la etiqueta *capitulo* el atributo *ref*.

```
/libro/capitulo[@autor='Juan']/*
```

Selecciona todos los nodos hijos que tengan como atributo en la etiqueta *capitulo* el atributo *ref* con el valor "Juan".

descendant: permite seleccionar todos los nodos que descienden del conjunto de nodos contexto.

Se corresponde con la doble barra, "**//**", aunque se puede usar *descendant::*.

```
/libro//parrafo
```

Selecciona todos los nodos descendientes de *parrafo*, no solo los hijos directos.

self: se refiere al nodo contexto y se corresponde con el punto "**.**".

```
./parrafo
```

Devuelve los nodos *parrafo* descendientes del nodo contexto.

parent: selecciona los nodos padre. Para referirnos a él usamos los dos puntos, "**..**".

```
../capitulo
```

Devuelve los nodos *capitulo* descendientes del nodo padre.

Varios predicados pueden unirse mediante los operadores lógicos ***and***, ***or*** o ***not***.

Funciones

Las funciones predeterminadas para XPath pueden incluirse en las expresiones.

Cada función en la biblioteca de funciones se especifica utilizando un prototipo **que proporciona el tipo de retorno, el nombre de la función y el tipo de argumentos**.

Si un tipo de argumento es seguido por un signo de interrogación el argumento es opcional, de lo contrario el argumento es requerido.

Las funciones que podemos utilizar son:

- ***boolean()***, al aplicarla sobre un conjunto de nodos devuelve *true* si no esta vacío.
- ***not()***, al aplicarla sobre un predicado devuelve *true* si el predicado es falso, y *false* si el predicado es *true*.
- ***true()***, devuelve el valor *true*.
- ***false()***, devuelve el valor *false*.
- ***count()***, devuelve el número de nodos que forman un conjunto de nodos.
- ***name()***, devuelve el nombre de un nodo.
- ***local-name()***, devuelve el nombre del nodo actual o del primer nodo de un conjunto de nodos.
- ***namespace-uri()***, devuelve la URI del nodo actual o del primer nodo de un conjunto dado.
- ***position()***, devuelve la posición de un nodo en su contexto comenzando en 1.
- ***last()***, devuelve el último elemento del conjunto dado.
- ***normalize-space()***, permite normalizar los espacios de una cadena de texto, es decir, si se introduce una cadena donde hay varios espacios consecutivos, esta función lo sustituye por uno solo.
- ***string()***, es una función que convierte un objeto en una cadena. Los valores numéricos se convierten en la cadena que los representa, teniendo en cuenta que los positivos pierden el signo. Los valores booleanos se convierten en la cadena que representa su valor, esto es *true* o *false*.

- ***concat()***, devuelve dos cadenas de texto concatenadas. El ejemplo siguiente devuelve "Xpath permite obtener datos de un fichero XML":
  concat('XPath', 'permite obtener datos de un fichero XML')
- ***string-length()***, devuelve el número de caracteres que tiene una cadena de caracteres.
- ***sum()***, devuelve la suma de los valores numéricos de cada nodo en un conjunto de nodos determinado.

Funciones XPath
Si deseas saber más sobre las funciones de XPath puedes visitar este enlace.
https://www.w3.org/TR/1999/REC-xpath-19991116/#corelib

XSLT

XSLT (XSL Transformation) es el lenguaje de hojas de estilo recomendado para manejar las transformaciones de documentos XML en cualquiera de sus versiones.

XSLT es mucho más sofisticado que CSS, ya que con XSLT se pueden agregar y/o eliminar elementos en la representación y también atributos desde o hacia el archivo de salida.

También a través de XSLT **se pueden reorganizar y ordenar elementos**, realizar pruebas y tomar decisiones sobre qué elementos ocultar y mostrar, ya que cuenta con una parte procedimental en la que se pueden tomar decisiones.

XSLT utiliza el lenguaje XPath para encontrar información en un documento XML.

De XML a XML

Para poder transformar ficheros XML que tienen un formato especificado en otro fichero XML diferente, debemos usar una hoja de estilo XSLT que se puede aplicar para traducir los datos de una gramática XML a otra.

Aunque más adelante profundizaremos en los diferentes métodos que tenemos a nuestra disposición gracias a XSLT, veamos un ejemplo de transformación de un XML a otro. En el siguiente ejemplo podemos ver que el XML está estructurado con la información almacenada en atributos.

El objetivo de este ejemplo es crear un nuevo XML con la información en etiquetas, a excepción del nombre de la tienda, que se dejará como atributo.

La estructura original de cada objeto es:

<item type="tienda" nombre="amazon" codigo="1102" producto="tablet 7 pulgadas" precio="28.875"/>

Y se quiere conseguir esta otra estructura para cada elemento tienda:

```
<tienda marca="amazon">
  <identificacion>tablet 7 pulgadas</identificacion>
  <codigo>1102</codigo>
  <precio>28.875</precio>
</tienda>
```

```
<?xml version="1.0"?>
<productos>
  <item type="tienda" nombre="amazon" codigo="1102" producto="tablet 7 pulgadas"
precio="28.875"/>
  <item type="tienda" nombre="tiendaPC" codigo="1104" producto="tablet 10
pulgadas" precio="92.250"/>
  <item type="tienda" nombre="tiendaPC" codigo="1105" producto="tablet 14
pulgadas" precio="20.313"/>
</productos>
```

XML origen.

Para realizar la transformación creamos un XML con las etiquetas de procesado, que localizan los nodos en el documento original mediante XPath y lo transforman.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
      <portfolio>
        <xsl:for-each select="productos/item[@type='tienda']">
          <tienda>
            <xsl:attribute identificacion="marca">
                <xsl:value-of select="@nombre"/>
            </xsl:attribute>
            <identificacion><xsl:value-of select="@producto"/></identificacion>
            <codigo><xsl:value-of select="@codigo"/></codigo>
            <precio><xsl:value-of select="@precio"/></precio>
          </tienda>
        </xsl:for-each>
      </portfolio>
    </xsl:template>
</xsl:stylesheet>
```

XML traductor.

Es pronto aún para entender toda la sintaxis del ejemplo de traducción, pero vamos a comentar algunas de las líneas más importantes para comprender su funcionamiento.

- Dado que se trata de un documento XML, lo declaramos como tal:
  <?xml version="1.0"?>

- Tenemos que definir que el documento XML se trata de una hoja de estilos XSL:
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

- Con la siguiente línea indicamos el formato del documento de salida. Este es un elemento de nivel superior y siempre debe ser un hijo de ***<xsl:stylesheet>*** o de ***<xsl:transform>***.

  Los atributos que usamos en este caso son: *method*, donde ponemos que será otro XML; e *indent*, donde indicamos que será "indentada" según jerarquía.
  <xsl:output method="xml" indent="yes"/>

- Ahora se representa que se trata de una plantilla con una serie de acciones que se realizarán si el patrón de la plantilla encaja (*match*) con el árbol de nodos del XML: ***<xsl:template match="/">***

- A partir de aquí es donde empezamos a definir la transformación que sufrirá nuestro documento. Insertando en la plantilla las etiquetas de la nueva estructura e indicando cuál será su contenido, realizando un rastreo en el XML original.
  ... <portfolio> <xsl:for-each select="productos/item[@type='tienda']">
  <tienda> <xsl:attribute identificacion="marca"> ...

  Podemos comprobar que la sintaxis de la etiqueta ***xsl:for-each*** realiza una iteración aplicando las transformaciones sobre los elementos nodo encontrados por medio de *select*, donde hemos colocado el prefijo de ***XPath.***

```
<?xml version="1.0"?>
<portfolio>
  <tienda marca="amazon">
    <identificacion>tablet 7 pulgadas</identificacion>
    <codigo>1102</codigo>
    <precio>28.875</precio>
  </tienda>
  <tienda marca="tiendaPC">
    <identificacion>tablet 10  pulgadas</identificacion>
    <codigo>1104</codigo>
    <precio>92.250</precio>
  </tienda>
  <tienda marca="tiendaPC">
    <identificacion>tablet 14 pulgadas</identificacion>
    <codigo>1105</codigo>
    <precio>20.313</precio>
  </tienda>
</portfolio>
```

Resultado final de la transformación.

De XML a xHTML

Gracias a XSLT no solo podemos realizar transformaciones entre documentos XML. Como veremos, podemos crear muchos tipos de documentos, entre ellos documentos xHTML.

Veamos ahora un ejemplo similar al anterior, pero en este caso transformado un documento con una estructura nativa de XML a un formato web como el xHTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <libro>
      <titulo>La conjura de los necios</titulo>
      <descripcion>Cuerpo del documento</descripcion>
    </libro>
```

XML con los datos a transformar.

Como vemos en el ejemplo anterior, tenemos la información de un libro almacenado en un documento XML. Aunque, como ya aprendimos, podemos darle estilos CSS al documento para facilitar su visualización en los exploradores, esto no es aconsejable por temas relacionados con SEO o por la compatibilidad con otros procesos web.

Para solucionarlo creamos el siguiente XSLT, indicando la estructura típica de un HTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="libro">
    <html>
      <head>
        <title><xsl:apply-templates select="titulo" mode="head"/></title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="titulo" mode="head">
    <xsl:value-of select="text()"/>
  </xsl:template>
  <xsl:template match="titulo">
    <h1><xsl:value-of select="text()"/></h1>
  </xsl:template>
  <xsl:template match="descripcion">
    <h3><xsl:value-of select="text()"/></h3>
  </xsl:template>
</xsl:stylesheet>
```

Documento XML traductor.

Al igual que el ejemplo donde convertimos un XML a otro XML, vamos transformando el documento.

```
<html>
    <head>
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
    <title> La conjura de los necios </title>
    </head>
    <body>
      <h1> La conjura de los necios </h1>
      <h3>Cuerpo del documento</h3>
    </body>
</html>
```

Documento xHTML resultante.

Como resultado obtenemos un documento xHTML completamente normalizado.

Vista del documento xHTML en un explorador.

Procesador XSLT

Los procesadores XSLT son aplicaciones capaces de procesar documentos XML y realizar transformaciones mediante hojas XSLT.

Cuando se procesa el documento se genera una representación del mismo como un árbol de nodos y lo va recorriendo nodo a nodo realizando las transformaciones.

El nodo que se está procesando en cada momento se denomina **nodo contexto**.

Este tipo de aplicaciones las podemos encontrar en lenguajes de servidor (PHP, ASP, .NET, JSP, etc.) y en entornos de cliente, como los exploradores web, o lenguajes como JavaScript.

XSL Transformation Tool (XSLT)
Procesador on-line de XSLT. http://online-toolz.com/tools/xslt-transformation.php

Pasos para la transformación

- Se analiza la hoja de transformación y se convierte en un árbol.
- El documento XML es procesado y convertido en una estructura de árbol.

- El procesador XSLT se posiciona en la raíz del documento XML; en este punto el procesador se encuentra en el contexto original.
- Los elementos que no formen parte del espacio de nombres de prefijo *xsl* son procesados sin transformaciones.
- El procesador aplica cada regla nodo a nodo.

Funciones XSLT

Los elementos *<xsl:stylesheet>* y *<xsl:transform>* son elementos similares que sirven para definir el elemento raíz de la hoja de transformaciones.

<xsl: stylesheet>

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

<xsl: transform>

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

Como atributos obligatorios tiene ***version***, que indica la versión del documento XSLT y ***xmlns***, que marca el espacio de nombres del documento.

Para obtener acceso a las características, a los elementos y a los atributos de XSLT debemos declarar el espacio de nombres XSLT en la parte superior del documento.

El xmlns *xsl = "http://www.w3.org/1999/XSL/Transform"* apunta al espacio de nombres oficial del W3C XSLT. Si usamos este espacio de nombres se debe incluir el atributo *version = "1.0"*.

Esta etiqueta no reemplaza a la que define el tipo de documento XML:
<?xml version="1.0" encoding="UTF-8"?>

Elemento *<xsl:template>*

El elemento ***<xsl: template>*** se utiliza para **crear plantillas para transformaciones**.

El atributo ***match*** se utiliza para asociar una plantilla con un elemento XML. También sirve para definir una plantilla para todo el documento XML.

El valor del atributo *match* se construye a través de una expresión XPath, es decir, match = "/" define el documento completo.

```
<xsl:template match="/">
```

Elemento *<xsl:value-of>*

El elemento *<xsl: value-of>* puede utilizarse para extraer el valor de un elemento XML y añadirlo al flujo de salida de la transformación.

Usando el mismo ejemplo que hemos usado anteriormente podríamos incluir esta instrucción.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Titulo</th>
      <th>Autor</th>
    </tr>
    <xsl:for-each select=”libreria">
    <tr>
      <td><xsl:value-of select="/libro/titulo"/></td>
      <td><xsl:value-of select="/libro/autor"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento *<xsl:for-each>*

El elemento XSL *<xsl: for-each>* se puede utilizar para seleccionar cada elemento XML de un conjunto de nodos especificado recorriendo todos los nodos del tipo que indiquemos en el elemento.

Usando el mismo ejemplo que hemos usado anteriormente podríamos incluir este elemento.

También podemos filtrar la salida del archivo XML añadiendo un criterio al atributo *select* en el elemento *<xsl: for-each>*.

```
<xsl: for-each select = ”libreria / libro [autor= 'Miguel de Cervantes']">
```

Los operadores de filtros legales son:

- = Igual.
- ! = Distinto.
- ≪ Menor que.
- ≫ Mayor que.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Titulo</th>
      <th>Autor</th>
    </tr>
    <xsl:for-each select=”libreria">
    <tr>
      <td><xsl:value-of select="/libro/titulo"/></td>
      <td><xsl:value-of select="/libro/autor"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Ejemplo de documento de transformación a xHTML.

Elemento *<xsl:sort>*

Podemos usar el elemento *<xsl:sort>* para organizar la salida por un filtrado específico. Por ejemplo, por el autor del libro:

```
<xsl:sort select="autor"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Titulo</th>
      <th>Autor</th>
    </tr>
   <xsl: for-each select = ”libreria/libro”>
    <xsl:sort select="autor"/>
    <tr>
      <td><xsl:value-of select="/libro/titulo"/></td>
      <td><xsl:value-of select="/libro/autor"/></td>
    </tr>
    </xsl:for-each>
```

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

Elemento *<xsl:if>*

Podemos usar una estructura condicional para poder filtrar qué elementos se renderizan o transforman y cuáles no. La sintaxis de este elemento es:

```
<xsl:if test="expression">
...alguna salida si la expresión es cierta...
</xsl:if>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Titulo</th>
      <th>Autor</th>
    </tr>
   <xsl: for-each select = ”libreria/libro”>
     <xsl:if test="precio > 10">
        <tr>
          <td><xsl:value-of select="titulo"/></td>
          <td><xsl:value-of select=”autor"/></td>
          <td><xsl:value-of select="precio"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento *<xsl:choose>*

El elemento *<xsl: choose>* se utiliza junto con *<xsl: when>* y *<xsl: otherwise>* para expresar múltiples pruebas condicionales. La sintaxis de este elemento es:

```
<xsl:choose>
  <xsl:when test="expression">
```

```
    ... alguna salida ...
  </xsl:when>
  <xsl:otherwise>
    ... alguna salida ...
  </xsl:otherwise>
</xsl:choose>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Titulo</th>
      <th>Autor</th>
    </tr>
<xsl:for-each select="libreria/libro">
    <tr>
      <td><xsl:value-of select="titulo"/></td>
      <xsl:choose>
        <xsl:when test="precio > 10">
          <td bgcolor="#ff0fff">
          <xsl:value-of select="autor"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="autor"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento *<xsl:apply-templates>*

El elemento *<xsl: apply-templates>* aplica una plantilla al elemento actual o a los nodos secundarios del elemento actual.

Si añadimos un atributo *select* al elemento *<xsl: apply-templates>* procesará solamente el elemento secundario que coincida con el valor del atributo. Podemos usar el atributo *select* para especificar el orden en el que se procesan los nodos secundarios.

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

```
<xsl:template match="/">
  <html>
  <body>
  <h2>Colección de libros</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
<xsl:template match=”libro">
  <p>
  <xsl:apply-templates select=”titulo"/>
  <xsl:apply-templates select=”autor"/>
  </p>
</xsl:template>
<xsl:template match=”titulo">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
<xsl:template match=”autor">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
</xsl:stylesheet>
```

Elementos XSLT

Si quieres aprender más elementos, funciones o atributos referentes a XSLT, puedes visitar este enlace. https://developer.mozilla.org/es/docs/Web/XSLT/Elementos

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta lección hemos aprendido cómo los documentos XML definen datos cuando se usan como contenedores de estructuras complejas de datos. En su sintaxis no hay manera de definir el formato de salida de estos datos, por lo que se deben usar otros estándares para presentarlos formateados.

En ocasiones también es necesario cambiar el árbol de datos de un XML para transformarlo en otro que tenga los mismos datos pero con una estructura distinta.

XML permite crear documentos con datos pero sin un formato determinado. Mediante las transformaciones podemos obtener otros documentos que contienen partes del documento original. También podemos construir documentos con formato XHTML y cambiar los *schemas* de un XML a otro distinto. Estos nuevos documentos generados son el producto de las transformaciones.

Uno de los estándares de transformación de documentos XML en otros documentos XML o en formatos diferentes es XSL, que a través del lenguaje de transformación XSLT permite manejar la estructura de datos de los documentos entrantes para generar documentos con una estructura específica.