T5.1. XSLT: Modificar documentos XML

A menudo se necesita modificar un fichero XML para adaptar o modificar su información y que sea mostrada en otro documento XML.

Si deseamos transformar un XML en otro XML necesitaremos usar XSLT. Un archivo XSLT tiene la extensión XSL e indica las reglas para convertir entre formatos XML.

El documento XSL básico sería así:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
</xsl:stylesheet>
```

En este caso al estar vacío devuelve error.

Pero si completamos el documento del siguiente modo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
                Resultado que se va a mostrar
        </xsl:template>
</xsl:stylesheet>
```

Nos mostraría una hoja básica sin tratar ningún tipo de información la información.

Para ver una transformación práctica vamos a realizar el siguiente ejemplo:

Crear un fichero llamado **biblioteca.xml**, será el documento original XML y donde haremos una referencia al documento de la transformación, cuando abramos el documento con un navegador, directamente se aplicará la transformación que tenemos referenciada.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="transformacion.xsl" type="text/xsl"?>
<biblioteca>
         <libro>
                        <title>Don Quijote</title>
                        <autor>Cervantes</autor>
                </libro>
                <libro>
                        <title>Poeta en Nueva York</title>
                        <autor>Lorca</autor>
                </libro>
</biblioteca>
```

Crear el fichero XSLT llamado **transformacion.xsl**, donde mantendremos el mismo esquema del documento, pero eliminando el campo de autor y dejando únicamente el título:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
                <bibliotecareducida>
                        <xsl:for-each select="/biblioteca/libro">
                                <libro>
                                        <titulo>
                                                <xsl:value-of
select="title"/>
                                        </titulo>
                                </libro>
                        </xsl:for-each>
                </bibliotecareducida>
        </xsl:template>
</xsl:stylesheet>
```

Para ellos remarcamos dos etiquetas:

- **xsl:for-each:** que recorre todos los elementos como si de un for se tratase, en este caso recorre los elementos que encuentre dentro de biblioteca (elemento raíz) y una vez dentro, recorre cada uno de los libros. La forma de indicarlo es con la sentencia "/biblioteca/libro" como si de una consulta se tratase.
- **xsl:value-of:** nos devuelve un valor en concreto, en este caso extrae el "title" del documento original y lo enmarca entre etiquetas "titulo".

Desarrollar este ejemplo con dos ficheros en la misma carpeta y tratar de abrirlo con un navegador, no todos los navegadores permiten las transformaciones XSLT, prueba con Internet Explorer o Firefox. Si no, existen herramientas online como la publicada en EVA para desarrollar los ejercicios.

T5.2. XSLT: Salida del ejemplo

El ejemplo explicado anteriormente devuelve una salida similar a esta:

```
<bibliotecareducida>
  <libro>
    <titulo>Don Quijote</titulo>
  </libro>
  <libro>
    <titulo>Poeta en Nueva York</titulo>
  </libro>
</bibliotecareducida>
```

Vamos a explicar a partir del código XSLT de dónde viene cada elemento de la salida

`<?xml version="1.0" encoding="UTF-8"?>` **Cabecera de XML**
`<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">` **Cabecera para declarar que es una transformación**
`<xsl:template match="/">` **Toma como elemento inicial la etiqueta raíz y empezamos a crear la nueva salida**
`<bibliotecareducida>` **Nueva etiqueta raíz del documento**
`<xsl:for-each select="/biblioteca/libro">` **Por cada elemento libro que se encuentre dentro de biblioteca dará una vuelta al bucle**
`<libro>` **Creamos una etiqueta de apertura por cada libro que recorremos**
`<titulo>` **Creamos una etiqueta titulo dentro de cada libro**
`<xsl:value-of select="title"/>` **Escribimos el contenido del "title" de cada libro recorrido (recordar que en cada vuelta del bucle for podemos acceder al contenido de cada elemento)**
`</titulo>` **Cerramos la etiqueta de titulo**
`</libro>` **Cerramos la etiqueta de libro**
`</xsl:for-each>` **Cierre del bucle, si hay otro libro vuelve a la línea del for**
`</bibliotecareducida>` **Cerramos la nueva etiqueta raíz**
`</xsl:template>` **Cerramos el template o salida del documento**
`</xsl:stylesheet>` **Cerramos la transformación**

T5.3. XSLT: Salida de atributos

Partiendo del fichero original:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="transformacion.xsl" type="text/xsl"?>
<biblioteca>
         <libro>
                        <title>Don Quijote</title>
                        <autor>Cervantes</autor>
                </libro>
                <libro>
                        <title>Poeta en Nueva York</title>
                        <autor>Lorca</autor>
                </libro>
</biblioteca>
```

Dado el archivo XML del catálogo generar un XML en el que el autor vaya como un atributo del título, es decir, que quede algo así::

```
<catalogo>
      <libro>
        <titulo escritor="Cervantes">Don Quijote</titulo>
      </libro>
      <libro>
        <titulo escritor="Lorca">
        Poeta en Nueva York
        </titulo>
      </libro>
</catalogo>
```

Vamos a explicar a partir del código XSLT de dónde viene cada elemento de la salida

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

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```
Cabecera para declarar que es una transformación

```
        <xsl:template match="/">
```
Toma como elemento inicial la etiqueta raíz y empezamos a crear la nueva salida

```
                <catalogo>
```
Nueva etiqueta raíz del documento

```
                        <xsl:for-each select="/biblioteca/libro">
```
Por cada elemento libro que se encuentre dentro de biblioteca dará una vuelta al bucle

```
                                <libro>
```
Por cada libro se crea una etiqueta

```
                                   <titulo>
```
Por cada libro creamos una etiqueta, ésta será la que contenga el atributo

```
                                        <xsl:attribute name="escritor">
```
Definimos que la etiqueta contenedora "titulo" va a tener un atributo llamado escritor

```
                                            <xsl:value-of select="autor"/>
```
El valor del atributo será el valor que hay en el fichero original en la etiqueta llamada "autor"

```
                                        </xsl:attribute>
```
Cerramos la declaración del atributo

```
                                        <xsl:value-of select="title"/>
```
La línea de value-of entre las etiquetas, declara el contenido de etiqueta

```
                                   </titulo>
```
Cerramos la etiqueta titulo contenedora del atributo

```
                                </libro>
```
Cerramos la etiqueta libro

```
                        </xsl:for-each>
```
Cierre del bucle, si hay otro libro vuelve a la línea del for

```
                </catalogo>
```
Cerramos la nueva etiqueta raíz

```
        </xsl:template>
```
Cerramos el template o salida del documento

```
</xsl:stylesheet>
```
Cerramos la transformación

T5.4. XSLT: Condicionales

Dado el archivo XML del catálogo generar un XML en el que el autor vaya como un atributo del título, es decir, que quede algo así::

```
<coches>
    <coche>
      <marca>Mercedes</marca>
      <color>rojo</color>
    </coche>
    <coche>
      <marca>BMW</marca>
      <color>verde</color>
    </coche>
    <coche>
      <marca>Audi</marca>
      <color>verde</color>
    </coche>
</coches>
```

Vamos a explicar a partir del código XSLT de dónde viene cada elemento de la salida

```
<?xml version="1.0" encoding="UTF-8"?> Cabecera de XML
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> Cabecera para declarar que es una transformación
      <xsl:template match="/"> Toma como elemento inicial la etiqueta raíz y empezamos a crear la nueva salida
                        <cochesrojos> Nueva etiqueta raíz del documento
                            <xsl:for-each select="/coches/coche"> Por cada elemento coche que se encuentre dentro de biblioteca dará una vuelta al bucle
                                <xsl:if test="color='rojo'"> Filtramos por los coches que tengan el color rojo
                                                    <coche> Por cada coche se crea una etiqueta
                                                      <xsl:value-of select="marca"/> Mostramos el contenido de la etiqueta marca de cada coche
                                                    </coche> Cerramos la etiqueta coche
                                </xsl:if> Cerramos el condicional
                                        </xsl:for-each> Cierre del bucle, si hay otro libro vuelve a la línea del for
                        </cochesrojos> Cerramos la nueva etiqueta raíz
      </xsl:template> Cerramos el template o salida del documento
</xsl:stylesheet> Cerramos la transformación
```

T5.5. XSLT: Obtención de atributos

Dado el siguiente archivo XML del catálogo generar un XML en el que el aparezca la marca y la matrícula en la misma etiqueta, es decir, que quede algo así:

```
<coches>
    <coche matricula="4545FFF">
      <marca>Mercedes</marca>
      <color>rojo</color>
    </coche>
    <coche matricula="2222BCC">
      <marca>BMW</marca>
      <color>verde</color>
    </coche>
    <coche matricula="1234DDD">
      <marca>Audi</marca>
      <color>verde</color>
    </coche>
</coches>
```

Generar un XML en el que el aparezca la marca y la matrícula en la misma etiqueta, es decir, que quede algo así:

```
<coches>
    <coche>Mercedes 4545FFF</coche>
    <coche>BMW 2222BCC</coche>
    <coche>Audi 1234DDD</coche>
</coches>
```

Vamos a explicar a partir del código XSLT de dónde viene cada elemento de la salida

```
<?xml version="1.0" encoding="UTF-8"?> Cabecera de XML
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> Cabecera para declarar que es una transformación
    <xsl:template match="/"> Toma como elemento inicial la etiqueta raíz y empezamos a crear la nueva salida
                <coches> Etiqueta raíz del documento
                    <xsl:for-each select="/coches/coche"> Por cada elemento coche que se encuentre dentro de biblioteca dará una vuelta al bucle
                                <coche> Por cada coche se crea una etiqueta
                        <xsl:value-of select="@matricula"/> Recuperamos el atributo matricula de coche
                        <xsl:value-of select="marca"/> Mostramos el contenido de la etiqueta marca de cada coche
                                </coche> Cerramos la etiqueta coche
                            </xsl:for-each> Cierre del bucle, si hay otro libro vuelve a la línea del for
                </coches> Cerramos la nueva etiqueta raíz
    </xsl:template> Cerramos el template o salida del documento
</xsl:stylesheet> Cerramos la transformación
```

T5.6. Operadores condicionales

En general, las condiciones se escriben así:

- > o mayor que o **>**
- < o menor que o **<**
- >= o mayor o igual o **>=**
- <= o menor o igual o **<=**
- != o distinto

T5.7. XSLT: Insertar texto

Definición y Uso

El <xsl:text> elemento se utiliza para escribir texto literal de la salida.

Tip: Este elemento puede contener texto literal, referencias de entidad, y #PCDATA.

Sintaxis

```
<xsl:text
disable-output-escaping="yes|no">

  <!-- Content:#PCDATA -->

</xsl:text>
```

atributos

Atributo	Valor	Descripción
disable-output-escaping	yes no	Opcional. "yes" indica que los caracteres especiales (como "<") debe ser de salida como es. "no" indica que los caracteres especiales (como "<") deben ser de salida como "& lt;". Por defecto es "no" . Este atributo no es compatible con Netscape 6

Ejemplo 1

Muestra el título de cada CD. Inserta un ", " entre cada CD-título, si no es el último CD - o la penúltima. Si se trata del último CD, se añade un "!" detrás del título. Si se trata de la penúltima, añadir una ", and " detrás del título:

```
<?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>My CD Collection</h2>
    <p>Titles:
    <xsl:for-each select="catalog/cd">
      <xsl:value-of select="title"/>
      <xsl:if test="position() < last()-1">
        <xsl:text>, </xsl:text>
      </xsl:if>
      <xsl:if test="position()=last()-1">
        <xsl:text>, and </xsl:text>
      </xsl:if>
      <xsl:if test="position()=last()">
```

```
        <xsl:text>!</xsl:text>
      </xsl:if>
    </xsl:for-each>
    </p>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

T5.8. ¿Qué es XQuery?

XQuery, también conocido como XML Query, es un lenguaje creado para buscar y extraer elementos y atributos de documentos XML. La mejor forma de entender este lenguaje es diciendo que XQuery es para XML lo que SQL es para las bases de datos. XQuery es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Abarca desde archivos XML hasta bases de datos relacionales con funciones de conversión de registros a XML.

Su principal función es extraer información de un conjunto de datos organizados como un árbol n-ario de etiquetas XML. En este sentido XQuery es independiente del origen de los datos.

Una consulta XQuery podría resolver, por ejemplo, la siguiente pregunta: "Seleccionar todos los libros con un precio menor a 20 euros de la colección de libros almacenada en un documento XML llamado catalogo.xml".

XQuery es un lenguaje funcional, lo que significa que en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL. Diversas expresiones pueden combinarse de una manera muy flexible con otras expresiones para crear nuevas expresiones más complejas y de mayor potencia semántica.