XSLT

En este capítulo se introduce la transformación de documentos XML mediante el lenguaje XSLT. Nos centraremos en la conversión en documentos HTML.

Tabla de contenido

XSLT

1 Introducción

XSL (e*Xtensible Stylesheet Language*) es una familia de lenguajes desarrollados por el W3C que permiten describir cómo debe ser presentada la información contenida en un documento XML. Este lenguaje consta de tres recomendaciones del W3C:

- XSL *Transformations* (XSLT) ➔ Permite transformar documentos en sintaxis XML a otra diferente.
- XML *Path Language* (Xpath) ➔ Permite buscar y acceder a los nodos del documento XML, así como seleccionar partes de este.
- XSL *Formatting Objects* (XSL-FO) ➔ Especifica el formato visual con el cual se quiere presentar el documento (abandonado en 2013).
- XQuery ➔ Un lenguaje para hacer consultas en documentos XML.

Este documento está dedicado al primero, XSLT o Transformaciones XSL, que consiste en un estándar de la organización W3C como una forma de transformar documentos XML en otros formatos, como XHTML, o incluso a formatos que no son XML.

Actualmente, XSLT es muy usado en la edición web, generando páginas HTML o XHTML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad.

XSLT está basado en XML, de forma que cada hoja de transformaciones es un documento XML bien formado.

Se trata de un lenguaje de programación declarativo que permite generar documentos en diferentes formatos de salida (XML, HTML, texto, PDF, RTF, etc), a partir de un documento XML. Se describe como declarativo porque consiste en la declaración de una serie de reglas o plantillas que hay que aplicar a un documento XML para transformarlo en la salida

deseada. Una regla asocia un patrón (expresión) con elementos del documento XML de partida y les aplica una serie de acciones.

Estas reglas se almacenarán en un documento de texto, que habitualmente tiene la extensión `.xsl` y se denomina hoja de transformaciones que, junto con el documento `.xml` origen serán pasados como parámetros a un procesador XSLT, el cual generará como salida el nuevo documento transformado.

1.1 Diferencias entre CSS y XSLT

El W3C ha definido dos familias de normas para hojas de estilos. La más antigua y simple es CSS, que permite definir propiedades de formato para elementos de marcado. Habitualmente, este lenguaje se emplea para dar formato a documentos web, pero también se puede utilizar para dar formato a documentos XML.

Sin embargo, a la hora de dar formato a un documento XML empleando CSS, hay ciertas tareas que no se pueden acometer:

- No se puede cambiar el orden en el que los elementos de un documento XML se visualizan (no puede ordenar elementos o filtrarlos según algún criterio).
- No se pueden realizar operaciones, como sumar los valores de todos los elementos `<precio>` de un documento.
- No se pueden combinar múltiples elementos, como todas las nóminas anuales de un empleado con el fin de obtener un total de ingresos y retenciones.

Estas limitaciones que aparecen en el uso de CSS se superan con XSLT. De hecho, en muchas ocasiones se usará CSS y XSLT de manera complementaria. Con XSLT se determinará qué contenido de un documento XML se quiere mostrar y en qué orden, realizándose si es necesario algún cálculo sencillo; con CSS se dará un formato visual a la información generada por XSLT.

1.2 Procesador XLST

Un procesador XLST es una aplicación capaz de procesar documentos XML mediante hojas de transformaciones XSLT. Cuando el procesador XSLT lee un documento XML genera una representación de dicho documento como un árbol de nodos. Los tipos de nodo son:

- ✔ Elemento
- ✔ Atributo
- ✔ Texto
- ✔ Comentario
- ✔ Instrucción de procesamiento
- ✔ Espacio de nombres.

Como ya se ha comentado previamente, las relaciones entre los nodos son las mismas

que en un árbol genealógico: padres, hijos, ascendientes, descendientes...

Los comentarios y las instrucciones de procesamiento son parte del árbol de nodos y deben ser tenidos en cuenta a la hora de contar nodos o iterar sobre ellos.

El procesador XSLT va recorriendo y procesando nodo a nodo. El nodo que se está tratando en un momento dado se denomina nodo de contexto.

Existen procesadores XLST en línea disponibles en Internet que permiten realizar la transformación de un documento XLST. También, hay otros que son aplicaciones de escritorio.

1.3 Proceso de transformación.

En este tema se realizarán transformaciones desde XML a texto, XML o HTML, siendo estas últimas las más comunes. Para realizar la transformación se precisan dos documentos:

- ✔ El documento XML de partida.
- ✔ La hoja de estilos XSLT

Una transformación XSL se puede llevar a cabo en tres ubicaciones distintas:

- ✔ En el servidor web ➔ Mediante algún lenguaje de servidor (PHP, asp.NET, JSP, etc) que aplique la XSLT al XML y envíe de vuelta el HTML producido al cliente. Esta técnica está fuera de los contenidos de este documento.
- ✔ En el cliente web ➔ El cliente web (Mozilla Firefox o Internet Explorer) realiza las transformaciones. Se puede hacer con JavaScript, aunque existen librerías que incluye funcionalidades como tratamiento de documentos XML, transformaciones XSLT, procesamiento de expresiones XPath, etc.
- ✔ Mediante una aplicación de escritorio independiente.

Los pasos para realizar una transformación son los siguiente:

1. La hoja de transformaciones XSLT es analizada y se convierte en una estructura de árbol.
2. El documento XML es procesado y convertido en una estructura de árbol.
3. El procesador XSLT se posiciona en la raíz del documento XML. Este es el contexto original.
4. Los elementos (como las etiquetas HTML) que no formen parte del espacio de nombres de prefijo `xsl`, son pasados a la cadena de salida sin modificarse. Se denominan elementos de resultado literal.
5. El procesador XSLT sólo aplica una regla o plantilla a cada nodo. Si tenemos dos reglas para el mismo nodo, el procesador sólo aplica la última que aparece.

El orden de las reglas en la hoja de transformaciones no es relevante. El orden del

recorrido del árbol por parte del procesador XSLT lo establece el documento XML.

Las hojas de transformación XSLT realizan la transformación del documento XML utilizando una o varias reglas, plantillas o *templates*. Estas plantillas unidas al documento fuente a transformar son la entrada de un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de salida, o, como en el caso de una página web, directamente en la pantalla del navegador.

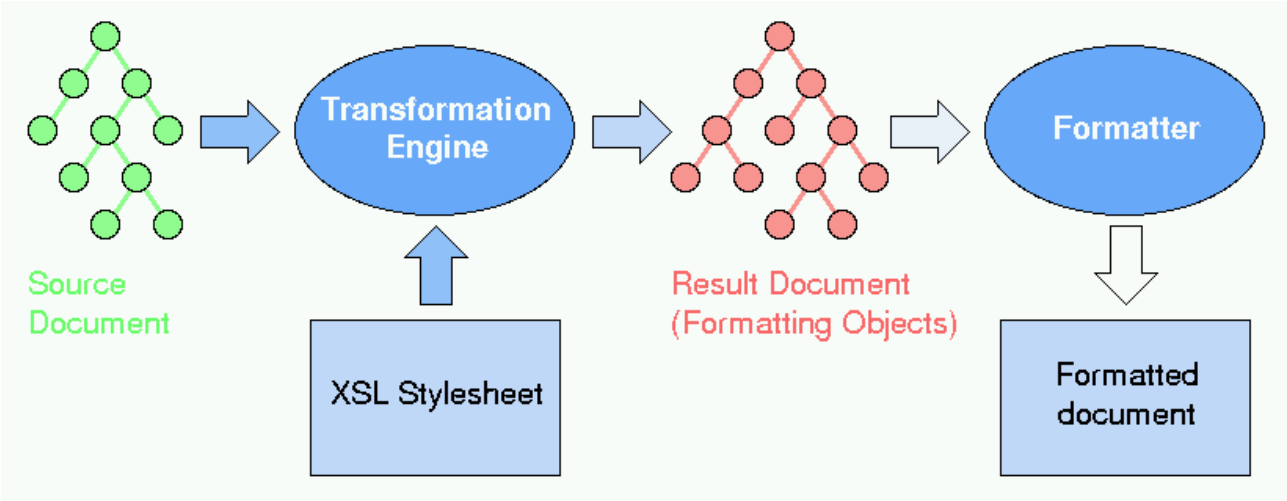

Figura 1: Transformación XSLT

1.4 Vincular un documento XML a una hoja de transformación XSLT

Para que un documento XML pueda transformarse empleando una hoja de transformación XSLT hay que establecer un vínculo del primero con la segunda. Esta transformación consiste en la aplicación de formato a los nodos del documento XML y puede hacerse mediante CSS o XSLT.

Si hemos decido aplicar formato mediante una hoja de estilos CSS tendremos que añadir en el documento XML, justo debajo de la declaración de documento XML, la siguiente instrucción.

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

Por ejemplo, si a un documento XML le vamos a aplicar un formato mediante un archivo CSS de nombre `componentes.css` que se encuentra en el mismo directorio que el documento XML original añadiríamos esta instrucción.

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

Obviamente, al indicar la URL del archivo CSS a vincular hace que podamos utilizar cualquier archivo CSS disponible en Internet y cuyo formato se ajuste a los elementos de nuestro documento XML.

Por otro lado, si lo que vamos a vincular es una hoja de transformación XSLT la instrucción anterior cambia de la siguiente manera:

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

Vemos que el atributo type indica ahora un tipo MIME text/xsl mientras que href apunta a una hoja de transformación XSLT. Siguiendo con nuestro ejemplo sería así:

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

Una vez hemos vinculado nuestro documento XML con nuestra hoja de estilos u hoja de transformación, solamente tenemos que invocar el formateador para que aplique el formato y genere la salida.

2 Formato mediante CSS

Se puede asociar de forma permanente una hoja de estilo, sea CSS o XSLT a un documento XML mediante la instrucción de procesamiento `<?xml-stylesheet ?>`. Esta instrucción se ubica al principio del documento XML, después de la declaración `<?xml ... ?>`.

Cuando se visualiza en un navegador web un documento XML enlazado con una hoja de estilo CSS, se muestra el resultado de la aplicación de los estilos a los elementos existentes.

El siguiente ejemplo consiste en un documento XML con los componentes de un ordenador al que se le va a aplicar una hoja de estilos CSS, generando una salida en un navegador. El código del documento `componentes.xml`, en cuya segunda línea se enlaza con la hoja de estilos `componentes.css`, es:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="componentes.css"?>
<componentes>
    <titulo>Componentes de un ordenador</titulo>
    <componente>
         <item>Placa base</item>
         <fabricante>ASUS</fabricante>
         <modelo>H81-M</modelo>
         <precio>215</precio>
    </componente>
    <componente>
         <item>Tarjeta gráfica</item>
         <fabricante>Nvidia</fabricante>
          <modelo>RTX2650</modelo>
          <precio>175</precio>
    </componente>
    <componente>
         <item>Disco duro</item>
         <fabricante>Samsung</fabricante>
         <modelo>EVO850</modelo>
         <precio>90</precio>
    </componente>
    <componente>
```

```
        <item>Monitor</item>
        <fabricante>ASUS</fabricante>
        <modelo>180K</modelo>
        <precio>290</precio>
    </componente>
</componentes>
```

El código de la hoja de estilos es:

```
componentes {
    display: block;
  font-family: sans-serif;
}

titulo {
    display: block;
    color: darkblue;
    font-weight: 600;
    font-size: 22pt;
    margin: 0.5em;
    text-decoration: underline;
}

componente {
    display: block;
    float: left;
    border: 2px solid black;
    border-radius: 10px;
    width: 15%;
    margin: 1em;
}

item {
    display: block;
    color: #000080;
    font-weight: 400;
    font-size: 18pt;
    text-align:center;
}

fabricante {
    display: block;
    color: #600060;
    font-weight: 400;
    font-size: 18pt;
    margin: 5px;
}

modelo {
    display: block;
    color: #006000;
    font-weight: 400;
    font-size: 18pt;
    margin: 5px;
```

```
}

precio {
    display: block;
    font-family: arial;
    color: #800000;
    font-weight: 400;
    font-size: 18pt;
    margin: 5px;
}
```

La salida en el navegador sería la siguiente:

Figura 2: Formato con CSS

3 Estructura de un documento XSL

Una hoja de transformaciones tiene la siguiente estructura:

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

    ...
</xsl:stylesheet>
```

Una hoja de transformaciones está compuesta de:

1. Una declaración de documento XML. Recomendable establecer el juego de caracteres del documento.
2. Un elemento raíz, denominado `<xsl:stylesheet>` o `<xsl:transform>` con un espacio de nombres `http://www.w3.org/1999/XSL/Transform`, siendo `xsl` el prefijo.
3. Los elementos de nivel superior que, si aparecen, son siempre hijos del elemento raíz. Estos son: `<xsl:import>` (si aparece, debe ser el primer hijo del elemento raíz), `<xsl:include>`, `<xsl:namespace-alias>`, `<xsl:output>`, `<xsl:strip-space>`, `<xsl:preserve-space>`, `<xsl:attribute-set>`, `<xsl:key>`, `<xsl:param>` (puede ser parámetro global o local), `<xsl:variable>` (puede ser variable global o local) y `<xsl:template>` con el que se definen las reglas o plantillas.

Generalmente, para obtener un resultado, es suficiente utilizar un número reducido de instrucciones de transformación.

Los elementos `<xsl:stylesheet>` y `<xsl:transform>` son equivalentes y se usan para definir el elemento raíz de la hoja de transformaciones. Debido a ello, a partir de ahora cuando se haga referencia al elemento `<xsl:stylesheet>`, lo que se diga será equivalente para `<xsl:transform>`.

Atributos obligatorios

`version`

Especifica la versión del XSLT del documento.

`xmlns`

Espacio de nombres del documento XML. Por defecto se usará xsl.

Atributos optativos principales

`exclude-result-prefixes`

Lista de espacios de nombres (prefijos), separados por espacios, que no deben ser enviados a la salida.

Dentro del elemento `<xsl:stylesheet>` se incluyen las plantillas que realizan la transformación de los elementos del documento XML. Si el documento XSLT no tiene ninguna plantilla definida, el procesador envía a la salida el texto del documento XML de entrada sin atributos.

Vamos a ver a continuación los elementos de nivel superior, descendientes directos de la raíz de documentos `<xsl:stylesheet>` para realizar la transformación.

3.1 Formato de salida

El elemento `<xsl:output>` define el formato del documento de salida. Es un elemento de nivel superior, y debe aparecer como hijo de `<xsl:stylesheet>`.

Atributos optativos principales:

`method`

Define el formato de salida. Por defecto es XML, pero si el primer hijo del nodo raíz es <html> y no hay nodos de texto previos, entonces el formato de salida es HTML. Posibles valores de este atributo son *xml*, *html*, *text*.

`version`

Define la versión del formato de salida (sólo para formatos de salida XML y HTML).

`encoding`

Indica el juego de caracteres de la salida. Por defecto es UTF-8.

`indent`

Indica si la salida debe ser sangrada de acuerdo a su estructura jerárquica.

`omit-xml-declaration`

Indica si la instrucción de procesamiento que declara el documento como XML se omitirá en la salida. Por defecto vale no, lo que significa que sí se enviará a la salida la declaración de documento XML <?xml version="1.0"?>. Posibles valores de este atributo son *no* y *yes*.

`standalone`

Indica si a la instrucción de procesamiento de declaración de tipo de documento como XML, <?xml version="1.0"?>, se le añade el atributo *standalone*. Por defecto vale no. Posibles valores de este atributo son *no* y *yes*.

Veamos un ejemplo:

En la siguiente hoja de transformación indicamos que el formato de salida es HTML, con un juego de caracteres iso-8859-1 y se omite en la salida la declaración de documento XML.

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

    <xsl:output method="html" encoding="iso-8859-1" omit-xml-
declaration="yes"/>

</xsl:stylesheet>
```

3.2 Plantillas

Ya hemos mencionado anteriormente que la transformación del nodo XML a su salida formateada se realiza mediante reglas o plantillas que se aplican a los nodos del documento XML. Definimos una plantilla mediante el elemento `<xsl:template>`. Es, junto con el elemento `<xsl:apply-templates>`, la instrucción principal de las hojas de transformaciones. Representa una plantilla con una serie de acciones que se realizarán si el patrón de la plantilla, definido en el atributo `match`, encaja con algún elemento o elementos del árbol XML.

En la definición de una plantilla tendremos en cuenta lo siguiente:

- ✔ El atributo `match` tiene como valor la ruta del elemento en el documento XML (una expresión XPath) e indica qué elementos del árbol XML se verán afectados por la plantilla.
- ✔ La transformación a realizar consiste en enviar a la salida todo el contenido dentro de la plantilla, es decir, lo que se ponga entre `<xsl:template>` y `</xsl:template>`. En el caso de no definir la transformación, es decir, no incluir contenido en la plantilla, el nodo XML al que se aplica se reemplazaría por una cadena vacía.
- ✔ Cuando se aplica una plantilla a un nodo del documento XML, se aplica únicamente al nodo seleccionado, pero se sustituye el nodo y todos sus descendientes por el

resultado de la aplicación de la plantilla, lo que nos haría perder la información de los descendientes.

- ✔ Si se quiere que antes de sustituir el nodo y todos sus descendientes por el contenido de la plantilla se apliquen también a los nodos descendientes otras plantillas que hayamos definido, tendremos que utilizar la instrucción `<xsl:apply-templates>` dentro del contenido de la plantilla.

Una plantilla es una regla, cuyas acciones se ejecutarán si el patrón que tiene asociado la plantilla encaja con algún elemento del árbol XML. Cuando una plantilla se aplica, se dice que ha sido instanciada.

Atributos optativos principales:

`name`

Indica un nombre para la plantilla. Sí se omite, entonces el atributo *match* es obligatorio. No puede haber en una hoja de transformaciones dos plantillas con el mismo nombre, sino se producirá un error.

`match`

Indica un patrón XPath o ruta del nodo XML al que se le aplicará la plantilla. Si se omite, entonces el atributo *name* es obligatorio.

`priority`

Es un valor real entre -9 y 9, con el primero como prioridad más baja y el segundo como prioridad más alta. Se utiliza para aplicar las reglas de resolución de conflictos entre plantillas.

`mode`

Permite distinguir dos plantillas cuyo atributo *match* sea el mismo, de manera que en la invocación desde el *<xsl:apply-templates>* se pueda indicar explícitamente cual usar.

Veamos un ejemplo de plantilla. En la siguiente hoja de transformación hemos definido una plantilla que se aplicará al elemento raíz del árbol y que generará un documento HTML idéntico para cualquier documento XML de entrada.

```
<?xml version="1.0"?>
    <xsl:stylesheet version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
         <xsl:output method="html" indent="yes"/>
         <xsl:template match="/">
              <html lang="es">
                   <body>
                   <h1>Documento HTML estático</h1>
                   </body>
              </html>
         </xsl:template>
    </xsl:stylesheet>
```

Centrémonos en el elemento `<xsl:template>` que define la plantilla. Podemos

observar lo siguiente:

- ✔ El contenido de la plantilla es código HTML, ya que el resultado del procesamiento del nodo está definido como tal en el elemento `<xsl:output>`.
- ✔ El contenido de la plantilla será el resultado de aplicar la plantilla al nodo XML correspondiente, el cuál está definido en el atributo `match`. En este ejemplo el valor de `match` es `/`, es decir, la raíz del árbol XML del documento al que se aplica esta plantilla.
- ✔ Dentro del contenido de la plantilla no se ha incluido un elemento `<xsl:apply-templates>` por lo que el nodo raíz del documento XML, junto con todos sus descendientes (en la práctica todo el documento XML), se reemplaza únicamente por la transformación definida en esta plantilla.
- ✔ Dentro de la plantilla se ha incluido contenido que no tiene el prefijo `xsl` por lo que se envía tal cual a la salida (elementos de resultado literal).

Según lo anterior, el resultado del procesamiento será el siguiente:

```
<html lang="es">
    <body>
    <h1>Documento HTML estático</h1>
    </body>
</html>
```

El resultado de este ejemplo es muy simple, ya que ningún dato del documento XML al que se aplique será incluido en la salida. Más adelante veremos como usar el elemento `<xsl:value-of>` para seleccionar valores de los elementos XML.

3.2.1 Procesamiento de los nodos hijo

El elemento `<xsl:apply-templates>` es, junto con el anterior `<xsl:template>`, la instrucción principal del lenguaje de transformaciones XSLT. La correcta comprensión de su funcionamiento nos permitirá sacar el máximo partido de las transformaciones que deseemos realizar.

Vimos en el apartado anterior, que al procesar un nodo XML, se reemplaza el nodo y todos sus descendientes por el contenido de su plantilla. Esto evita que se puedan aplicar más plantillas a los nodos hijo. Si queremos aplicar transformaciones a estos nodos tenemos que incluir dentro del contenido de la plantilla del nodo de contexto el elemento `<xsl:apply-templates>` para aplicar las plantillas de los nodos hijo.

Por tanto, este elemento procesa los nodos hijos del nodo de contexto que se encuentra procesando actualmente, es decir, seguir aplicando plantillas a los nodos hijos.

Atributos optativos principales

`select`

Es una expresión XPath que especifica los nodos a seleccionar para ser procesados. Si se

omite, todos los nodos descendientes del nodo actual serán seleccionados. Si se indica una expresión que no coincide con ningún conjunto de nodos, no se aplicará ninguna transformación.

`mode`

Permite distinguir entre varias plantillas con igual valor en su atributo *match*, por lo tanto sólo tiene sentido su existencia si existe el atributo *select*. Las plantillas que se aplicarían también deberán venir etiquetados con el mismo atributo *mode*.

Veamos un ejemplo. La siguiente hoja de transformaciones mostrará todo el contenido textual de los distintos elementos, descendientes de la raíz `/` del documento XML de partida, dentro de una capa `<div>`.

En la plantilla principal, que se vincula con la raíz del documento XML de partida, se envía a la salida la estructura principal de una página HTML y se invoca la aplicación de plantillas sin atributos. Esto producirá el efecto ya comentado: se enviará a la salida todo el contenido textual de los elementos descendientes de `/`, Si hubiera más plantillas, se aplicarían cuando correspondiera.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xs1="http://www.w3 .org/1999/XSL/Transform">

  <xsl:output encoding="utf-8" omit-xml-declaration="yes"
method="html" indent="yes"/>

  <xsl:template match="/">
    <html>
      <body>
        <h1>Transformación de los componentes de un PC</h1>
        <div><xsl:apply-templates /></div>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Veamos el resultado de aplicar esta hoja de transformación XSLT.

```
<html lang="es">
  <body><h1>Transformación de los componentes de un PC</h1>
  Componentes de un ordenador

    Placa base
    ASUS
    H81-M
    215 €


    Tarjeta gráfica
    Nvidia
    RTX-2650
    175 €
```

```
    Disco duro
    Samsung
    EVO 850
    90 €


    Monitor
    ASUS
    180K
    290 €

</body>
</html>
```

Vemos que el elemento `<xsl:apply-templates/>` se encuentra dentro del contenido de la plantilla que se aplica al nodo raíz del documento XML. Esto ha hecho posible aplicar transformaciones a sus nodos hijo. Sin embargo, no hemos definido más plantillas para los nodos hijo. El resultado de transformar un nodo que no tiene una plantilla es enviar a la salida el contenido del nodo. De ahí que aparezcan los contenidos de los elementos `titulo`, `componente`, `item`, `fabricante`, `modelo`, ... tal y como aparecen en el documento XML origen.

Vamos a ampliar el ejemplo anterior. Aquí vamos a definir varias plantillas:

- ✔ La principal se aplicará al elemento raíz /
- ✔ Una que se aplica al elemento `componentes` (nodo hijo de la raíz).
- ✔ Una que se aplicará al elemento `título` (nodo hijo de `componentes`).
- ✔ La última se aplica al elemento `componente` (nodo hijo de `componentes`).

La hoja de transformación XSLT sería la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output encoding="utf-8" omit-xml-declaration="yes"
method="html" indent="yes"/>
  <xsl:template match="/">
    <html lang="es">
      <body>
        <h1>Transformación de los componentes de un PC</h1>
         <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

    <xsl:template match="componentes">
    <div id="componentes">
      <h2>Composición de un PC</h2>
```

```
      <xsl:apply-templates />
      </div>
  </xsl:template>

  <xsl:template match="titulo">
      <div id="titulo">
        <h2>Lista de componentes</h2>
      </div>
  </xsl:template>

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

</xsl:stylesheet>
```

El resultado sería el siguiente:

```
<html lang="es">
  <body>
    <h1>Transformación de los componentes de un PC</h1>
    <div id="componentes"><h2>Composición de un PC</h2>
      <div id="titulo"><h2>Lista de componentes</h2></div>




    </div>
  </body>
</html>
```

En el resultado podemos observar lo siguiente:

- ✔ Vemos que la aplicación de las plantillas del nodo raíz ha provocado la aplicación de la plantilla para el elemento `componentes`.
- ✔ Este emite a la salida un título de nivel 2 en una capa y procesa la aplicación de las plantillas de los nodos hijo.
- ✔ El primer nodo hijo de `componentes` es `título`, el cual tiene como transformación una capa con un título de nivel 2.
- ✔ Finalmente, los componentes no aparecen. Hay una plantilla para el nodo `componente` (hijo de `componentes`), pero está vacía, no tiene contenido o, lo que es lo mismo, no tiene transformación. Por tanto se reemplaza cada nodo componente (y sus hijos) por una salida vacía. De ahí los saltos de línea debajo del título.

Hasta ahora hemos utilizado el elemento `<xsl:apply-templates />` para aplicar las plantillas de todos los nodos descendientes del nodo de contexto. Sin embargo podemos hacer uso del atributo `select` para especificar a qué nodos se les va a aplicar sus plantillas. Esto unido al hecho de que podemos incluir más de un elemento `<xsl:apply-templates />` nos permite aplicar plantillas de forma selectiva, seleccionando que nodos del documento original se van a enviar a la salida y cuáles no.

Por ejemplo, si en el ejemplo anterior cambiamos la plantilla del nodo /componentes e indicamos que se apliquen las plantillas de los nodos hijo correspondiente al elemento componente, tendríamos lo siguiente:

```
<xsl:template match="componentes">
    <div id="componentes">
      <h2>Composición de un PC</h2>
      <xsl:apply-templates select="componente"/>
    </div>
</xsl:template>
```

Al realizar la transformación tendremos el siguiente resultado.

```
<html lang="es">
  <body>
    <h1>Transformación de los componentes de un PC</h1>
    <div id="componentes"/>
  </body>
</html>
```

Vemos que al indicar que solamente se apliquen plantillas a los nodos hijo componente no se ha aplicado al elemento titulo. Por tanto este nodo no se transforma y no aparece su valor en la salida.

3.2.2 Path absoluto vs path relativo

En el atributo match debemos indicar una expresión XPath que puede ser relativa o absoluta. En los ejemplos previos hemos utilizado un path relativo ya que los nodos dependen del contexto actual. Un path absoluto comienza con una barra /, el cuál le dice al procesador XLST que comience en la raíz del documento sin tener en cuenta el contexto actual. En otras palabras, podemos evaluar un path absoluto desde cualquier nodo de contexto que queramos y el resultado será el mismo.

Una ventaja de utilizar path absoluto es que no tenemos que preocuparnos del nodo de contexto. Otro beneficio es que facilita al procesador XSLT encontrar el nodo que coincide con la expresión, ya que se da su ubicación exacta. Si algún elemento del path falla, entonces el procesador XSLT interrumpe la búsqueda en el árbol y devuelve una cadena vacía.

Un inconveniente de usar un path absoluto es que dificulta la reutilización de plantillas, ya que la plantilla se aplica a un único nodo con un path concreto.

3.2.3 Valores de los nodos XML

Hasta ahora, la transformaciones que hemos aplicado a los nodos XML han sido enviar a la salida texto fijo incluido en el contenido de la plantilla correspondiente. Esto está bien cuando queremos complementar el de la salida con nuevo contenido que no aparece en el nodo XML. Sin embargo, la transformación de un documento XML no puede implicar de ningún modo la pérdida de información del documento original en el resultado. Es decir, el contenido textual de los nodos XML también pueden, y en muchos casos deben, emitirse a

la salida.

Podemos acceder al valor de un nodo XML mediante el elemento `<xsl:value-of/>` para enviarlo a la salida.

Atributos obligatorios

`select`

Es una expresión XPath que especifica de qué elemento o atributo hay que extraer el contenido.

Atributos optativos principales

`disable-output-escaping`

Indica si los caracteres especiales, como < o &, deben ser enviados a la salida tal cual o en su forma de entidad, < o &. El valor por defecto es *no*, es decir, se generarán en su forma de entidad.

Vamos a completar el ejemplo anterior añadiendo a una plantilla la extracción del contenido de un elemento para enviarla a la salida.

Por ejemplo, supongamos que reescribimos la plantilla correspondiente al elemento titulo de la siguiente forma:

```
<xsl:template match="titulo">
   <div id="titulo">
      <h2><xsl:value-of select="/componentes/titulo"/></h2>
   </div>
</xsl:template>
```

Hemos reemplazado un texto estático que habíamos puesto antes por el contenido del nodo `titulo` en el documento XML. El resultado de transformar el nodo `titulo` sería el siguiente:

```
<div id="titulo"><h2>Componentes de un ordenador</h2></div>
```

Es fundamental entender el valor que debemos de indicar en el atributo `select` para saber el contenido de qué nodo estamos accediendo. Tal y como vimos en el apartado anterior para el atributo `match` del elemento `<xsl:apply-templates />` podemos indicar un path absoluto o relativo. Por tanto, con `<xsl:value-of />` podemos acceder al valor de cualquier nodo XML, no solo del nodo de contexto. De ahí el atributo obligatorio `select` en el que indicamos una expresión XPath. En el ejemplo anterior hemos puesto la ruta absoluta del nodo XML (`/componentes/titulo`). Esto nos permite acceder a cualquier nodo XML. También podemos poder una ruta relativa, lo que nos permitiría acceder al contenido de un nodo descendiente del nodo de contexto.

Si, lo que nos ocurre en este ejemplo, queremos acceder al contenido del propio nodo de contexto, podríamos haber puesto como valor del atributo `select` un punto. Así

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

También podemos acceder al valor de un atributo de un elemento XML. En este caso el valor del atributo `select` debería escribirse de la siguiente forma: `ruta/@atributo`.

Por ejemplo, supongamos que el nodo `titulo` del documento XML se redefine de la siguiente forma añadiendo un atributo `nivel` con el valor `1`:

```
<titulo nivel="1">Componentes de un ordenador</titulo>
```

Ahora redefinimos la plantilla y accedemos al valor del atributo.

```
<xsl:template match="titulo">
      <div id="titulo">
        <h2>
          <xsl:value-of select="./@nivel"/>.-
          <xsl:value-of select="/componentes/titulo"/>
        </h2>
      </div>
  </xsl:template>
```

Vemos que el valor `./@nivel` del `select` obtiene el valor del atributo `nivel` del nodo de contexto. El valor del nodo lo hemos accedido con su ruta absoluta. Ahora la salida sería así

```
<div id="titulo"><h2>1.- Componentes de un ordenador</h2></div>
```

3.2.4 Incluir texto en la salida

Con el elemento `<xsl:text>` podemos incluir en la salida texto literal, referencias de entidad y texto con marcado.

Atributos optativos principales

`disable-output-escaping`

> Indica si los caracteres especiales, como < o &, deben ser enviados a la salida tal cual o en su forma de entidad, < o &. El valor por defecto es *no*, es decir, se generarán en su forma de entidad.

Normalmente, cualquier texto en una plantilla se copia en la salida sin necesidad de incluirlo en un elemento `<xsl:text>`. Sin embargo, hay dos razones para usarlo:

- ✔ Control del espacio en blanco ➜ Un espacio en blanco extra se preserva al usar este elemento, es decir, el texto se visualiza tal y como se pone sin eliminar los espacios en blanco.
- ✔ Incluir caracteres especiales como `&` o `>` ➜ Por ejemplo, podríamos necesitar incluir `>` para visualizar el símbolo `>`. Este método es preferido a utilizar carácter de escape.

✔ El elemento tiene etiqueta de cierre obligatoria.

Veamos un ejemplo de uso. Imaginemos que deseamos que la transformación del documento XML sea HTML5. En este caso deberemos incluir en la plantilla correspondiente a la raíz del documento el siguiente elemento `<xsl:text>`.

```
<xsl:template match="/">
  <xsl:text>&gt;!DOCTYPE html&lt;</xsl:text>
  <html lang="es">
    <body>
      <h1>Transformación de los componentes de un PC</h1>
      <div><xsl:apply-templates /></div>
    </body>
  </html>
</xsl:template>
```

Como se puede apreciar se ha incluido una declaración HTML5 en la que tenemos que insertar `<!DOCTYPE html>` antes de la etiqueta `<html>`. Al tener que insertar símbolos < y > lo hemos hecho con sus correspondientes caracteres referenciados `&gt;` y `&lt;`. Al aplicar la plantilla saldrá lo siguiente

```
<!DOCTYPE html>
<html lang="es">
  <body>
    <h1>Transformación de los componentes de un PC</h1>
    <div>
    ...
    </div>
  </body>
</html>
```

3.2.5 Invocar una plantilla directamente

El elemento `<xsl:call-template>` permite invocar la aplicación de una plantilla saltándose el orden establecido de recorrido del árbol XML. Para ello tenemos, cuando queramos invocar una plantilla mediante este método tendremos que asignarle un nombre con el atributo `name` del elemento `<xsl:template>` y, posteriormente, utilizar este nombre en la invocación con `<xsl:call-template>`.

Atributos obligatorios

`name`

Indica el nombre de la plantilla a invocar.

Veamos un ejemplo. Tenemos la plantilla para el elemento /titulo de la siguiente forma:

```
<xsl:template match="/componentes/titulo" name="titulo">
  <div id="titulo">
  <h2><xsl:value-of select="/componentes/titulo"/></h2>
</div>
</xsl:template>
```

Vemos que solamente envía a la salida una capa con un encabezado de nivel 2 para el contenido del elemento título. Esta plantilla tiene el atributo `name` con el valor `título`. Ahora en la plantilla para el elemento `/componentes` podemos invocar la plantilla mediante `<call-template>` por su nombre.

```
<xsl:template match="/componentes">
  <div id="componentes">
    <xsl:call-template name="titulo"/>
    <h3>El PC lleva los siguientes componentes:</h3>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates select="componente"/>
  </div>
</xsl:template>
```

Hemos invocado la plantilla titulo antes de aplicar las plantillas para cada elemento componente. El resultado sería el siguiente.

```
<div id="componentes">
  <div id="titulo"><h2>Componentes de un ordenador</h2></div>
    <h3>El PC lleva los siguientes componentes:</h3>
    …
</div>
```

Hemos omitido la salida de la aplicación de la plantilla de cada componente por brevedad. Vemos que al invocarse el titulo se aplica su plantilla antes que la de los componentes. Los valores del atributo `name` deben ser únicos.

3.3 Reglas de resolución de conflictos

Si hay más de una plantilla para aplicar a un mismo elemento, existen una serie de reglas que resuelven en qué orden se aplicarán. Cada regla tendrá una prioridad, que puede ir de -9 a 9, siendo 9 la prioridad máxima, pero que al aplicarse automáticamente tendrá los siguientes valores.

Patrón	Prioridad
`"*"`, `"@*"`, `"text()"` y otros patrones para los que no se indica un nombre concreto.	-0,5
`"Prefijo:*"`, `"@prefijo:*"` y otros patrones para los que se indica el espacio de nombres, pero no un nombre concreto.	-0.25
`"producto"`, `"@unidades"` y otros patrones donde se indiquen nombres de elementos y atributos concretos	0
`"proveedor/representante"`, `"productos/producto/precio"`, `"precio/@unidades”` y otros patrones para los que se indique una jerarquía (además del nombre del elemento y/o atributo)	0.5

En resumen, cuanto más específico es el patrón, mayor prioridad tiene.

Según la especificación de XSLT, cuando aparecen varias plantillas con la misma prioridad, o se produce un error o se aplica la última plantilla en aparecer, En la práctica, los

procesadores XSLT aplican la última regla que aparece y siguen procesando.

Se puede asignar a las plantillas prioridades de manera explicita mediante el atributo `priority`. Si a una plantilla se le asigna una prioridad mayor que 0.5 se asegura que esta plantilla tendrá prioridad en su aplicación que cualquier otra plantilla que no incluya una declaración explicita de prioridad.

3.4 Instrucciones de control

Hay situaciones en las que necesitamos aplicar plantillas en función del resultado de la evaluación de una o varias condiciones. Para ello tenemos unos elementos que permiten controlar la aplicación de plantillas.

3.4.1 Instrucción condicional

El elemento `<xsl:if>` aplica una plantilla solamente si se cumple una condición indicada en su atributo `test`.

Atributos obligatorios

`test`

Indica la condición a evaluar antes de aplicar la plantilla.

Veamos un ejemplo. Queremos enviar a la salida únicamente los componente cuyo precio es superior a 100 €. Tendríamos que poner la siguiente plantilla para el elemento `componente`.

```
<xsl:template match="/componentes/componente">
  <xsl:if test="precio &gt; 100">
    <xsl:value-of select="."/>
  </xsl:if>
</xsl:template>
```

Vemos que la condición `precio > 100` estamos utilizando el elemento precio que es descendiente directo del elemento componente. La salida excluiría el componente `Disco duro` ya que tiene un precio de `90`.

3.4.2 Condiciones múltiples

Podemos evaluar una expresión y enviar diferentes salidas dependiendo del valor de la expresión. Para ello utilizaremos los elementos `<xsl:choose>`, `<xsl:when>` y `<xsl:otherwise>`. Su comportamiento es muy similar al selector múltiple switch de los lenguajes de programación Java o C.

La forma de evaluar la expresión es mediante el atributo test del elemento `<xsl:when>` que funciona igual que vimos en el apartado anterior con `<xsl:if>`.

Veamos un ejemplo. Supongamos que queremos cambiar el color de fondo de los componentes en función de su precio. Si es inferior a 100 € será verde, si está entre 100 € y 200 € será azul, y si es superior a 200 € será rojo. La plantilla para el elemento componente

sería la siguiente:

```
<xsl:template match="/componentes/componente">
  <xsl:choose>
    <xsl:when test="precio &lt; 100">
      <span style="bgcolor:green;">
      <xsl:value-of select="."/>
      </span>
    </xsl:when>
    <xsl:when test="precio &lt; 200">
      <span style="bgcolor:blue;">
      <xsl:value-of select="."/>
      </span>
    </xsl:when>
    <xsl:otherwise>
      <span style="bgcolor:red;">
      <xsl:value-of select="."/>
      </span>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

La salida que produciría esta plantilla sería la siguiente:

```
<span style="bgcolor:red;">
  Placa base
  ASUS
  H81-M
  215
</span>
<span style="bgcolor:blue;">
  Tarjeta gráfica
  Nvidia
  RTX-2650
  175
</span>
<span style="bgcolor:green;">
  Disco duro
  Samsung
  EVO 850
  90
</span>
<span style="bgcolor:red;">
  Monitor
  ASUS
  180K
  290
</span>
```

En cuanto una condición en un elemento `<xsl:when>` es cierta, se aplica su contenido que será enviado a la salida y no se siguen evaluando condiciones en subsecuentes elementos `<xsl:when>`. Si ninguna condición es cierta, entonces se envía a la salida el contenido del elemento `<xsl:otherwise>` si lo hubiera.

3.4.3 Iterar una lista de elementos

El elemento `<xsl:for-each>` permite iterar sobre un conjunto de nodos y aplicar transformaciones en cada uno de ellos.

Atributos obligatorios

`select`

Expresión XPath que indica el conjunto de elementos sobre los que se itera.

Por ejemplo. Queremos presentar los nodos de nuestro documento XML de ejemplo en formato de tabla. Podemos crear la siguiente plantilla para la raíz del árbol.

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">&lt;!DOCTYPE
html&gt;</xsl:text>
  <html lang="es">
    <body>
      <h1><xsl:value-of select="componentes/titulo"/></h1>
      <table>
        <tr>
          <th>Item</th>
          <th>Fabricante</th>
          <th>Modelo</th>
          <th>Precio</th>
        </tr>
        <xsl:for-each select="componentes/componente">
          <tr>
            <td><xsl:value-of select="item"/></td>
            <td><xsl:value-of select="fabricante"/></td>
            <td><xsl:value-of select="modelo"/></td>
            <td><xsl:value-of select="precio"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

El resultado de aplicar esta plantilla sería la siguiente:

```
<!DOCTYPE html>
<html lang="es">
  <body>
    <h1>Componentes de un ordenador</h1>
    <table>
      <tr>
        <th>Item</th>
        <th>Fabricante</th>
        <th>Modelo</th>
        <th>Precio</th>
      </tr>
      <tr>
```

```
        <td>Placa base</td>
        <td>ASUS</td>
        <td>H81-M</td>
        <td>215</td>
      </tr>
      <tr>
        <td>Tarjeta gráfica</td>
        <td>Nvidia</td>
        <td>RTX-2650</td>
        <td>175</td>
      </tr>
      <tr>
        <td>Disco duro</td>
        <td>Samsung</td>
        <td>EVO 850</td>
        <td>90</td>
      </tr>
      <tr>
        <td>Monitor</td>
        <td>ASUS</td>
        <td>180K</td>
        <td>290</td>
      </tr>
    </table>
  </body>
</html>
```

Vemos que al iterar sobre los elementos `componente` hemos cambiado el recorrido del árbol XML. En lugar de recorrerlo aplicando plantillas sobre los nodos hijo de arriba a abajo y de izquierda a derecha, hemos iterado sobre todos los elementos `componente` y empleando el elemento `<xsl:value-of>` hemos accedido a los valores de cada nodo hijo de cada elemento `componente`.

En el ejemplo anterior hemos podido distribuir los valores de los elementos en una tabla de forma fácil porque cada elemento hijo de `componente` es diferente y hemos podido acceder a su valor con `<xsl:value-of>`. Sin embargo, si tenemos varios elementos del mismo tipo no podríamos utilizar este último. En este caso es mejor utilizar otro elemento `<xsl:for-each>` anidado para recorrerlos todos. Por ejemplo, supongamos el siguiente documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="05_componentes.xsl"?>
<componentes>
  <titulo>Componentes de un ordenador</titulo>
  <componente>
    <item tipo="nombre">Placa base</item>
    <item tipo="marca">ASUS</item>
    <item tipo="modelo">H81-M</item>
    <item tipo="precio">215</item>
  </componente>
  <componente>
    <item tipo="nombre">Tarjeta gráfica</item>
```

```
    <item tipo="marca">Nvidia</item>
    <item tipo="modelo">RTX-2650</item>
    <item tipo="precio">175</item>
  </componente>
  <componente>
    <item tipo="nombre">Disco duro</item>
    <item tipo="marca">Samsung</item>
    <item tipo="modelo">EVO 850</item>
    <item tipo="precio">90</item>
  </componente>
  <componente>
    <item tipo="nombre">Monitor</item>
    <item tipo="marca">ASUS</item>
    <item tipo="modelo">180K</item>
    <item tipo="precio">290</item>
  </componente>
</componentes>
```

Vemos que cada elemento `componente` tiene varios elementos `item`. En este caso no podemos acceder a cada uno de ellos con `<xsl:value-of>` utilizando el atributo `tipo`. Sin embargo, también podemos utilizar un elemento `<xsl:for-each>` anidado como en la siguiente plantilla.

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">&lt;!DOCTYPE
html&gt;</xsl:text>
  <html lang="es">
    <body>
      <h1><xsl:value-of select="componentes/titulo"/></h1>
      <table>
        <tr>
          <th>Item</th>
          <th>Fabricante</th>
          <th>Modelo</th>
          <th>Precio</th>
        </tr>
        <xsl:for-each select="componentes/componente">
          <tr>
          <xsl:for-each select="item">
            <td><xsl:value-of select="."/></td>
          </xsl:for-each>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

El resultado de aplicar la plantilla anterior es el mismo. La diferencia estriba en el uso de un segundo elemento `<xsl:for-each>` dentro del primero, cuyo atributo `select` tiene el valor `item` para que acceda a todos los elementos `item` dentro del elemento componente actual (de ahí el path relativo). El elemento `<xsl:value-of>` con valor `.` en el atributo `select` permite acceder al valor del nodo actual.

3.4.4 Ordenar elementos

El elemento `<xsl:sort>` envía a la salida datos ordenados por algún criterio. Este elemento solo puede estar dentro de un elemento `<xsl:for-each>` o `<xsl:apply-templates>`.

Atributos optativos principales

`select`

Es una expresión XPath que indica el nodo o conjunto de nodos por los que constituirán el criterio de ordenación.

`lang`

Indica qué idioma se utiliza al ordenar. Es un código de idioma.

`data-type`

Especifica el tipo de datos de los datos que se ordenarán. Posibles valores: `text`, `number`, `QName`

`order`

Concreta si el orden es ascendente o descendente. Posibles valores: `ascending`, `descending`

`case-order`

Especifica si las letras en minúscula (`lower-first`) o mayúscula (`upper-first`) aparecerán primero al ordenar. Posibles valores: `lower-first`, `upper-first`.

Veamos un ejemplo sencillo. Supongamos que queremos hacer el mismo listado anterior en formato tabular de los componentes anteriores ordenados por precio. Necesitamos la siguiente plantilla.

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">&lt;!DOCTYPE
html&gt;</xsl:text>
  <html lang="es">
    <body>
      <h1><xsl:value-of select="componentes/titulo"/></h1>
      <table>
        <tr>
          <th>Item</th>
          <th>Fabricante</th>
          <th>Modelo</th>
          <th>Precio</th>
        </tr>
        <xsl:for-each select="componentes/componente">
          <xsl:sort select="precio" data-type="number"
order="ascending"/>
          <tr>
            <td><xsl:value-of select="item"/></td>
            <td><xsl:value-of select="fabricante"/></td>
```

```
            <td><xsl:value-of select="modelo"/></td>
            <td><xsl:value-of select="precio"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

Si aplicamos la transformación veremos que los elementos están ordenados ascendentemente por precio.

4 Elementos avanzados

Existen una serie de elementos cuyo manejo permite afinar el diseño de la hoja de transformación, pero no son esenciales para su creación.

4.1 Modos de las plantillas

Al procesar una plantilla aplicamos las plantillas de los nodos hijo mediante `<xsl:apply-templates/>`. Ya hemos visto en ejemplos previos que podemos hacerlo para todos los nodos hijo omitiendo el atributo `select`, o indicando que nodos hijos se procesarán estableciendo algún valor para el atributo `select`.

Sin embargo, también es posible procesar un nodo hijo más de una vez, enviando a la salida diferente resultados en cada una de ellas. Esto obligaría a definir más de una plantilla para el mismo nodo con un modo diferente, e invocarla explícitamente con ese modo. Para estas situaciones disponemos del atributo `mode`, el cual se utiliza en la definición de una plantilla para establecer su modo, y en la invocación de la aplicación de plantillas para aplicar solamente aquella que tenga el mismo modo definido. Veamos un ejemplo.

Supongamos que queremos hacer un listado de los componentes del PC completo y posteriormente un resumen que indique que elementos hay. La siguiente plantilla del elemento `componentes` especifica que se aplique las plantillas de los nodos `componente` hijo de dos formas. Para ello en cada invocación de `<xsl:apply-templates />` utiliza el atributo `mode` para especificar qué plantilla se invoca.

```
<xsl:template match="/componentes">
  <div id="componentes">
    <xsl:apply-templates select="componente" mode="detallado"/>
    <h3>El PC lleva los siguientes componentes:</h3>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates select="componente" mode="resumido"/>
  </div>
</xsl:template>
```

Además, el elemento componente dispone de dos plantillas, cada una con un modo diferente. Serían las siguientes:

```
<xsl:template match="/componentes/componente" mode="detallado">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="/componentes/componente" mode="resumido">
  <xsl:value-of select="item"/><xsl:text>&#10;</xsl:text>
</xsl:template>
```

Al realizar la transformación tendremos la siguiente salida:

```
<html lang="es">
  <body>
    <h1>Transformación de los componentes de un PC</h1>
    <div id="componentes">
    Placa base
    ASUS
    H81-M
    215 €

    Tarjeta gráfica
    Nvidia
    RTX-2650
    175 €

    Disco duro
    Samsung
    EVO 850
    90 €

    Monitor
    ASUS
    180K
    290 €
  <h3>El PC lleva los siguientes componentes:</h3>
Placa base
Tarjeta gráfica
Disco duro
Monitor
</div>
  </body>
</html>
```

4.2 Preservar o suprimir espacios

5 Bibliografía

CASTRO,J.M. y RODRÍGUEZ, J.R., *Lenguajes de Marcas y Sistemas de Gestión de Información*. Ed Garceta 2012 – ISBN: 978-84-1545-217-1