

UD5_Transformación de documentos XML

A_3. Transformacións XSLT avanzadas

1.	A3. Transformacións XSLT avanzadas.....	3
1.1	Aplicar varios patróns ao mesmo nodo	3
1.1.1	Prioridade dos patróns	4
1.2	Copiar nodos do documento orixe ao documento de saída	5
1.3	Variables.....	6
1.3.1	Empregar variables	7
1.4	Estruturas de control.....	8
1.4.1	Condicións simples	8
1.4.2	Condicións múltiples	9
1.5	Estruturas de iteración	10
1.6	Especificar unha orde para os nodos.....	11
1.7	Patróns con nome.....	12
1.8	Parámetros	13
1.8.1	Valores por defecto	13
1.8.2	Parámetros globais	14
1.9	Eliminando e mantendo espazos	14
1.10	Claves e identificadores.....	15
1.10.1	Acceder aos nodos da clave	16
1.10.2	Agrupamento de nodos	16

1. A3. Transformacións XSLT avanzadas

1.1 Aplicar varios patróns ao mesmo nodo

Nun documento XSLT non se deben crear varios patróns que fagan referencia ao mesmo nodo do documento orixe. Cando ao procesar un nodo do documento orixe, o procesador XSLT atopa dous ou máis patróns cun atributo "match" que lle corresponda, o seu comportamento pode ser diferente dependendo do procesador XSLT que esteamos a empregar. Pode:

- Xerar unha mensaxe de erro.
- Aplicar o patrón que apareza máis tarde no documento XSLT.

Por exemplo, se sobre o documento XML:

```
<?xml version="1.0" encoding="utf-8"?>
<módulo>
  <profesor>Xaime Louzán</profesor>
</módulo>
```

Aplicáramos a seguinte transformación:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/módulo/profesor">
    <patrón num="1" />
  </xsl:template>
  <xsl:template match="/módulo/profesor">
    <patrón num="2" />
  </xsl:template>
</xsl:stylesheet>
```

Poderíamos obter como resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
  <patrón num="2"/>
```

Pero se o que queremos é transformar un ou varios nodos de dúas ou máis formas diferentes, necesitamos que o procesador aplique máis de un patrón sobre o mesmo nodo.

Para lograr isto, empregamos o atributo "mode". Este atributo aplícase aos elementos "<xsl:template>" de xeito que cando varios patróns se aplican ao mesmo nodo, os valores dos seus atributos "mode" deben ser distintos.

Un patrón cun atributo "mode" concreto execútase cando é chamado empregando un elemento "<xsl:apply-templates>" con ese mesmo atributo "mode". Por exemplo, supoñamos que partindo do seguinte documento XML:

```
<?xml version="1.0" encoding="utf-8"?>
<ciclo siglas="ASIR">
  <módulo>Linguaxes de Marcas</módulo>
  <módulo>Servizos de Rede</módulo>
  <módulo>Fundamentos Hardware</módulo>
</ciclo>
```

Queremos obter unha páxina web como a seguinte, onde o texto correspondente a cada un dos módulos do documento orixinal transfórmase primeiro nun encabezado, e logo nunha lista.



A forma de facelo é cun documento XSLT como o seguinte:

```
<?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>
    <head></head>
    <body>
      <xsl:apply-templates mode="encabezados"/>
      <ul>
        <xsl:apply-templates mode="lista"/>
      </ul>
    </body>
  </html>
</xsl:template>
<xsl:template match="módulo" mode="encabezados">
  <h2><xsl:value-of select="text()" /></h2>
</xsl:template>
<xsl:template match="módulo" mode="lista">
  <li><xsl:value-of select="text()" /></li>
</xsl:template>
</xsl:stylesheet>
```

1.1.1 Prioridade dos patróns

En realidade, existe un sistema de prioridades que indica ao procesador XSLT cal é o patrón que debe coller para procesar un nodo do documento orixe, en caso de que teña que escoller entre varios. O procesador XSLT escollerá sempre o patrón con maior prioridade; no caso de non atopar un único patrón con maior prioridade cos outros, é cando escollerá o último que apareza ou xerará unha mensaxe de erro.

O sistema de prioridades pode ser explícito (nos mesmos indicamos cal é a prioridade de cada patrón) ou implícito. Cando queremos indicar nos mesmos a prioridade dun patrón, o faremos engadíndolle o atributo "priority".

```
<xsl:template match="módulo" priority="2">
```

```
...
</xsl:template>
```

As prioridades implícitas entran en xogo cando creamos un patrón sen atributo "priority". Neste caso, para poder escoller o procesador XSLT asígnalles a estes patróns unha prioridade implícita entre -0,5 e +0,5, que será maior canto máis específica sexa a expresión do seu atributo "match".

Por exemplo, se transformamos o documento XML.

```
<?xml version="1.0" encoding="utf-8"?>
<módulo>
  <profesor>Xaime Louzán</profesor>
</módulo>
```

Empregando o seguinte patrón.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/módulo">
    <xsl:apply-templates />
  </xsl:template>
  <xsl:template match="profesor">
    <patrón num="1" />
  </xsl:template>
  <xsl:template match="*">
    <patrón num="2" />
  </xsl:template>
</xsl:stylesheet>
```

O nodo "<profesor>" transformarase empregando o patrón "<xsl:template match='profesor'>", que aínda que aparece antes no documento XSLT, é máis específico que o patrón "<xsl:template match='*'>", e por tanto ten maior prioridade implícita. Obteremos como resultado de saída.

```
<?xml version="1.0" encoding="UTF-8"?>
<patrón num="1"/>
```

1.2 Copiar nodos do documento orixe ao documento de saída

Nos patróns de XSLT temos dúas opcións para copiar nodos presentes no documento orixe ao documento de saída:

- "<xsl:copy>" realiza unha copia sinxela do elemento ao que se refire o patrón. Non copia os seus atributos, o seu texto ou os seus fillos. Soamente o elemento. Este tipo de copia recibe o nome de copia superficial (*shallow copy*). Se quixéramos procesar tamén o seu contido, habería que facelo de forma específica.
- "<xsl:copy-of>" polo contrario fai unha copia profunda (*deep copy*) do elemento e trasladada ao documento resultante tamén o texto que contén, os seus fillos e atributos. A diferenza do anterior, este elemento debe estar baleiro e é obrigatorio indicar cun atributo "select" a expresión XPath correspondente ao nodo ou nodos que queremos copiar.

Así, seguindo co noso documento XML de exemplo.

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

```
<profesor>Xaime Louzán </profesor>
</módulo>
```

O resultado obtido por un patrón que empregue "<xsl:copy>" como o seguinte.

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

Será:

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

E cando empregamos no patrón "<xsl:copy-of>".

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

O resultado pasa a ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<módulo>
  <profesor>Xaime Louzán</profesor>
</módulo>
```



Na tarefa 1 traballaremos con transformacións que apliquen varios patróns a un mesmo elemento do documento orixe.

1.3 Variables

Ao igual que noutros linguaxes de programación, en XSLT podemos definir e empregar variables. O elemento "<xsl:variable>" define unha variable. O nome da variable establece-se empregando o atributo obrigatorio "name="nome da variable", e o seu valor pódese obter a partires de:

- unha expresión XPath empregando o atributo "select".

```
<xsl:variable name="num_profesores" select="count(profesor)" />
```

- o contido do elemento "<xsl:variable>".

```
<xsl:variable name="var">valor da variable</xsl:variable>
<!-- Tamén podería poñerse o valor nunha cadea de texto dentro do atributo
select -->
<xsl:variable name="var" select="'valor da variable'" />
```

É importante destacar que o valor dunha variable en XSLT non se pode cambiar. Asígnase-lle cando se declara e mantense ese mesmo valor de aí en diante.

As variables pódense definir no interior dun patrón ou fora deles, como fillo directo do elemento "<xsl:stylesheet>". Segundo sexa o caso, teremos:

- unha variable local a un patrón (non existirá fora dese patrón).
- unha variable global ao documento XSLT (poderase empregar dende calquera patrón do documento).

Entre as utilidades que podemos darlles ás variables en XSLT están:

- Definir valores que poidan cambiarse de xeito doado para alterar o comportamento da transformación.

```
<xsl:variable name="cor_fondo">#dedede</xsl:variable>
```

- Almacenar valores de conxuntos de nodos para empregalos posteriormente.

```
<xsl:variable name="encabezado">
  <tr>
    <th>Nome</th>
    <th>Descrición</th>
  </tr>
</xsl:variable>
```

- Almacenar expresións complexas, para aumentar a lexibilidade do código e o rendemento do procesador XSLT.

```
<xsl:variable name="pelis"
  select="//película[actúa/@id=//actor[nome='Elisabeth Shue']/@id]" />
```

- Almacenar o valor devolto por elementos XSLT.

```
<xsl:variable name="encabezado">
  <xsl:element name="tr">
    <xsl:element name="th">Nome</xsl:element>
    <xsl:element name="th">Descrición</xsl:element>
  </xsl:element>
</xsl:variable>
```

1.3.1 Empregar variables

Para obter o valor dunha variable, anteponse ao seu nome o símbolo "\$". Ao executar a transformación, a expresión "\$variable" substituirase polo valor da variable. Por exemplo:

```
<xsl:variable name="cor_fondo">#dedede</xsl:variable>
<xsl:element name="body">
  <xsl:attribute name="bgcolor">
    <xsl:value-of select="$cor_fondo" />
  </xsl:attribute>
</xsl:element>
```

Se queremos obter o valor dunha variable dentro dun atributo, tamén podemos empregar chaves:

```
<xsl:variable name="cor_fondo">#dedede</xsl:variable>
<body bgcolor="{ $cor_fondo }" />
```

Cando a variable contén un conxunto de nodos, poderíamos copialos ao documento de saída empregando "<xsl:copy-of>".

```
<xsl:variable name="pelis"
```

```

    select="//película[actúa/@id="//actor[nome='Elisabeth Shue']/@id]" />
<xsl:copy-of select="$pelis" />

```

1.4 Estruturas de control

Como parte dos patróns, podemos usar certas estruturas de control no procesamento dos elementos do documento orixinal.

1.4.1 Condicións simples

O elemento "<xsl:if>" permítenos procesar unha parte do patrón unicamente cando se cumpre unha condición. A condición especificase mediante unha expresión dentro do atributo obrigatorio "test". Cando o resultado da expresión é certo, execútanse o contido do elemento.

Por exemplo, partindo do seguinte documento XML.

```

<?xml version="1.0" encoding="UTF-8"?>
<alumnos>
  <alumno>
    <nome>Perico dos Palotes</nome>
    <nota>8</nota>
  </alumno>
  <alumno>
    <nome>Arsenio Lupin</nome>
    <nota>9.7</nota>
  </alumno>
  <alumno>
    <nome>Frodo Bolson</nome>
    <nota>4.6</nota>
  </alumno>
  <alumno>
    <nome>Smeagol</nome>
    <nota>6.2</nota>
  </alumno>
  <alumno>
    <nome>Pulgarcito</nome>
    <nota>7.6</nota>
  </alumno>
</alumnos>

```

Poderíamos transformalo nunha táboa HTML na que as filas correspondentes a alumnos con nota < 5 teñan o fondo vermello.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html"/>
  <xsl:template match="/">
    <html>
      <head></head>
      <body>
        <table border="1">
          <tr bgcolor="gray">
            <th>Alumno</th>
            <th>Nota</th>
          </tr>
          <xsl:apply-templates select="alumnos/alumno" />
        </table>
      </body>
    </html>
  </template>

```

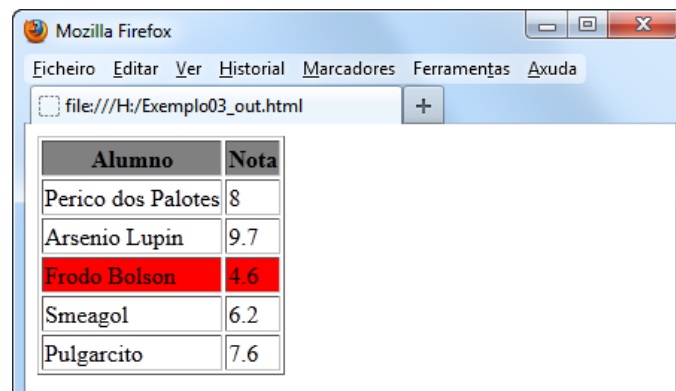


```

</html>
</xsl:template>
<xsl:template match="alumno">
  <xsl:element name="tr">
    <xsl:if test="nota < 5">
      <xsl:attribute name="bgcolor">red</xsl:attribute>
    </xsl:if>
    <td><xsl:value-of select="nome" /></td>
    <td><xsl:value-of select="nota" /></td>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>

```

O resultado obtido sería:



Alumno	Nota
Perico dos Palotes	8
Arsenio Lupin	9.7
Frodo Bolson	4.6
Smeagol	6.2
Pulgarcito	7.6

Fíxate que para evitar problemas cos caracteres de apertura de elementos, os operadores "<" e ">" deben substituírse respectivamente por "<" e ">".

1.4.2 Condicións múltiples

O elemento "<xsl:if>" permítenos avaliar o resultado dunha expresión booleana. Cando queremos crear unha condición máis complexa, na que o resultado da expresión poda ser comparado con varios posibles valores, teremos que empregar o elemento "<xsl:choose>".

Dentro do elemento "<xsl:choose>" aníñanse tantos elementos "<xsl:when>" como se- xa necesario, cadanseu co seu respectivo atributo "test". Cando a expresión correspondente a un elemento "<xsl:when>" é certa, procésase o seu contido e detense o procesamento do resto.

Se non se cumpre a condición asociada a ningún dos elementos "<xsl:when>", procesa- rase o contido do elemento "<xsl:otherwise>" en caso de que exista.

Por exemplo, para transformar o mesmo documento anterior poñendo unha cor distinta para cada nota poderíamos facer:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html"/>
  <xsl:template match="/">
    <html>
      <head></head>
      <body>
        <table border="1">
          <tr bgcolor="gray">
            <th>Alumno</th>
            <th>Nota</th>

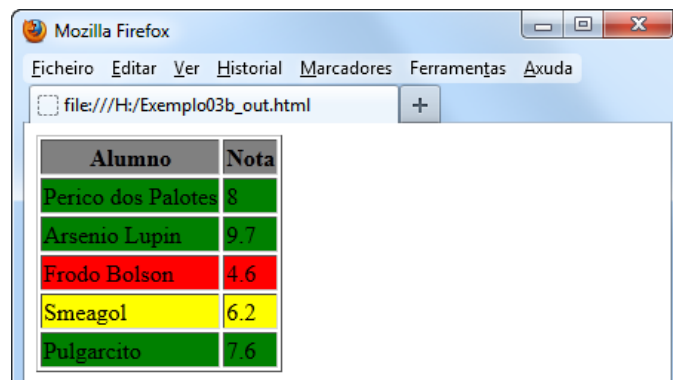
```

```

        </tr>
        <xsl:apply-templates select="alumnos/alumno" />
    </table>
</body>
</html>
</xsl:template>
<xsl:template match="alumno">
    <xsl:element name="tr">
        <xsl:choose>
            <xsl:when test="nota < 5">
                <xsl:attribute name="bgcolor">red</xsl:attribute>
            </xsl:when>
            <xsl:when test="nota < 7">
                <xsl:attribute name="bgcolor">yellow</xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name="bgcolor">green</xsl:attribute>
            </xsl:otherwise>
        </xsl:choose>
        <td><xsl:value-of select="nome" /></td>
        <td><xsl:value-of select="nota" /></td>
    </xsl:element>
</xsl:template>
</xsl:stylesheet>

```

E obteríamos o seguinte resultado:



Alumno	Nota
Perico dos Palotes	8
Arsenio Lupin	9.7
Frodo Bolson	4.6
Smeagol	6.2
Pulgarcito	7.6

1.5 Estruturas de iteración

Ata o de agora estivemos a empregar patróns para procesar os conxuntos de nodos contidos nun elemento. Pero en XSLT existe outro xeito de facelo: empregando o elemento "<xsl:for-each>".

O elemento "<xsl:for-each>" debe ir acompañado dun atributo "select" que faga referencia a un conxunto de nodos. O contido do elemento "<xsl:for-each>" procesarase unha vez por cada nodo no conxunto de nodos referenciado.

Por exemplo, poderíamos refacer a transformación anterior procesando os alumnos de forma iterativa do seguinte xeito.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="UTF-8" indent="yes" method="html"/>
    <xsl:template match="/">
        <html>
            <head></head>
            <body>
                <table border="1">

```

```

        <tr bgcolor="gray">
            <th>Alumno</th>
            <th>Nota</th>
        </tr>
        <xsl:for-each select="alumnos/alumno">
            <xsl:element name="tr">
                <xsl:choose>
                    <xsl:when test="nota < 5">
                        <xsl:attribute name="bgcolor">red
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="nota < 7">
                        <xsl:attribute name="bgcolor">yellow
                        </xsl:attribute>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:attribute name="bgcolor">Green
                        </xsl:attribute>
                    </xsl:otherwise>
                </xsl:choose>
                <td><xsl:value-of select="nome" /></td>
                <td><xsl:value-of select="nota" /></td>
            </xsl:element>
        </xsl:for-each>
    </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

1.6 Especificar unha orde para os nodos

Como acabamos de ver, temos dúas formas para procesar un conxunto de nodos: empregando un patrón cun atributo "match" ou empregando un elemento "<xsl:for-each>" cun atributo "select".

Empregando calquera destes dous métodos, o procesador XSLT escolle os nodos do conxunto de nodos para procesalos na mesma orde na que aparecen no documento orixinal. No exemplo anterior, procesará aos alumnos tal e como aparecen no documento XML.

En XSLT podemos cambiar a orde na que se procesan os nodos dun conxunto de nodos engadindo, con calquera das dúas formas indicadas, o elemento "<xsl:sort>". Este elemento debe aparecer dentro dun "<xsl:apply-templates>" ou xusto a continuación dun "<xsl:for-each>", sen outros elementos intermedios.

O elemento "<xsl:sort>" ten unha serie de atributos opcionais que indican o tipo de ordenamento a levar a cabo:

- "select" indica a chave de ordenación. Se non figura este atributo, suponse como chave de ordenación "select=". "".
- "order" pode tomar dous valores para indicar se queremos facer ordenación ascendente ("ascending", que é o valor por defecto) ou ordenación descendente ("descending").
- "case-order" tamén pode tomar dous valores ("upper-first" ou "lower-first") en función de se queremos que as letras maiúsculas se ordenen antes ou despois das minúsculas.
- "data-type" indica o tipo dos datos que imos a ordenar. Por defecto trátanse como texto ("text") pero ás veces é útil tratalos como números ("number"), de xeito por exemplo que "7" vaia antes de "12".

Por exemplo, no exemplo anterior para ordenar os nodos pola nota de xeito descendente teríamos que engadir despois de "<xsl:for-each select='alumnos/alumno'>" un elemento:

```
<xsl:sort select="nota" order="descending" data-type="number" />
```

E obteríamos:




The screenshot shows a Mozilla Firefox browser window displaying a table with two columns: 'Alumno' and 'Nota'. The table contains five rows of data, sorted by grade in descending order. The rows are: Arsenio Lupin (9.7), Perico dos Palotes (8), Pulgarcito (7.6), Smeagol (6.2), and Frodo Bolson (4.6). The background colors of the rows are green, green, yellow, yellow, and red respectively.

Alumno	Nota
Arsenio Lupin	9.7
Perico dos Palotes	8
Pulgarcito	7.6
Smeagol	6.2
Frodo Bolson	4.6

É posible engadir máis de un criterio de ordenación, de xeito que os nodos que teñan a mesma orde segundo o primeiro criterio pase a ordenarse seguindo o segundo.

Por exemplo, para ordenar polo nome a aqueles alumnos coa mesma nota (supoñendo que houbera algún), teríamos que facer:

```
<xsl:sort select="nota" order="descending" data-type="number" />
<xsl:sort select="nome" />
```

 Agora imos facer a tarefa 2, na que traballaremos con algúns destes conceptos.

1.7 Patróns con nome

Ata o de agora traballamos con patróns que o procesador XSLT executaba cando o seu atributo "match" correspondíase cun nodo ou nodos do documento XML orixe. Pero existe outro xeito de executar un patrón: dándolle un nome e dicíndolle ao procesador XSLT que o procese.

Neste caso o patrón deberá ter un atributo "name" que o identifique, e poderá non ter atributo "match" (debe ter un cando menos, pero tamén pode ter ambos atributos). Os patróns que identifiquemos cun atributo "name" poderán executarse dende calquera punto doutro patrón empregando un elemento "<xsl:call-template>" que o identifique co seu propio atributo "name".

Empregar patróns con nome é semellante a empregar funcións nunha linguaxe de programación. Así, no exemplo anterior poderíamos crear un patrón para introducir na táboa os valores de cada fila:

```
<xsl:template name="fila">
  <td><xsl:value-of select="nome" /></td>
  <td><xsl:value-of select="nota" /></td>
</xsl:template>
```

E o executaríamos dende o patrón anterior substituíndo as liñas anteriores por:

```
<xsl:call-template name="fila" />
```

1.8 Parámetros

Os patróns teñen tamén a posibilidade de recibir parámetros cando son executados. Isto é, dentro dun elemento "`<xsl:apply-templates>`" ou dun elemento "`<xsl:call-template>`" podemos indicar o nome e valor dunha ou varias variables que recibirá o patrón que se execute.

Cando creemos un patrón que reciba parámetros, deberemos indicalo empregando un elemento "`<xsl:param>`" por cada un dos parámetros, no que figure o seu nome nun atributo "name".

Cando fagamos a chamada ao patrón, cada un dos parámetros que lle pasemos deberá ir como primeiro contido do elemento "`<xsl:apply-templates>`" ou "`<xsl:call-template>`", no seu propio elemento "`<xsl:with-param>`", que funciona do mesmo xeito que vimos antes coas variables: indicando o seu nome cun atributo "name" e o seu valor ben cun atributo "select" ou ben definíndoo co seu propio contido.

Por exemplo, se queremos que o patrón con nome anterior reciba o nome e a nota do alumno como parámetros, teríamos que definilo do seguinte xeito:

```
<xsl:template name="fila">
  <xsl:param name="nome" />
  <xsl:param name="nota" />
  <td><xsl:value-of select="$nome" /></td>
  <td><xsl:value-of select="$nota" /></td>
</xsl:template>
```

Fíxate que agora os atributos "select" dos elementos "`<xsl:value-of>`" fan referencia a parámetros, e debemos indicalo antepoñéndolles un signo "\$".

Agora na chamada debemos pasarlle ao patrón os valores dos parámetros requiridos.

```
<xsl:call-template name="fila">
  <xsl:with-param name="nome"><xsl:value-of select="nome" /></xsl:with-param>
  <xsl:with-param name="nota"><xsl:value-of select="nota" /></xsl:with-param>
</xsl:call-template>
```

1.8.1 Valores por defecto

Do mesmo xeito que facemos nunha chamada para definir o valor dun parámetro no elemento "`<xsl:with-param>`", dentro dun patrón podemos definir un valor por defecto de cada parámetro. Por exemplo:

```
<xsl:template name="fila">
  <xsl:param name="nome">(descoñecido)</xsl:param>
  <xsl:param name="nota">(non presentado)</xsl:param>
  <td><xsl:value-of select="$nome" /></td>
  <td><xsl:value-of select="$nota" /></td>
</xsl:template>
```

Estes valores empregaranse cando na chamada ao patrón non se defina o parámetro respectivo.

1.8.2 Parámetros globais

En XSLT tamén poden crearse parámetros globais, isto é, parámetros que afectan á totalidade do documento XSLT. Para isto, defínese un elemento "<xsl:param>" como fillo directo do elemento raíz "<xsl:stylesheet>". Estes parámetros poden ter valores por defecto do mesmo xeito que os parámetros dos patróns.

Os parámetros globais poden empregarse para cambiar o comportamento dunha transformación. A forma de indicar os nomes e valores dos parámetros globais cando se executa unha transformación depende do procesador que empreguemos. En Saxon podemos facelo engadindo os nomes dos parámetros e os seus valores a continuación do nome dos documentos XML e XSLT:

```
saxon alumnos.xml taboa_notas.xsl cor_fondo=#e4e4e4
```

En Xalan o procedemento consiste en engadir por cada parámetro unha opción "-param" seguida do seu nome e do seu valor. Por exemplo, "-param cor_fondo #e4e4e4".

1.9 Eliminando e mantendo espazos

No seu comportamento por defecto, un procesador XSLT mantén os espazos que contén o documento orixe. Se empregamos unha transformación como a seguinte, que simplemente copia un documento orixe directamente ao de saída.

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

O documento resultante será unha copia idéntica do documento orixe, incluíndo aos nodos que conteñan soamente texto.

Para poder eliminar os espazos innecesarios do documento orixe, podemos empregar o elemento XSLT "<xsl:strip-space>". Este elemento sitúase como fillo directo do elemento "<xsl:stylesheet>" e leva un único atributo "elements", no que se indica unha lista separada por espazos de elementos do documento orixe.

O procesador XSLT eliminará os espazos destes elementos antes de procesalos. Tamén podemos incluír na lista de elementos comodíns como o "*" para indicar varios nodos.

Por exemplo, tomando como documento orixe o seguinte.

```
<?xml version="1.0" encoding="UTF-8"?>
<novas>
  <nova>
    <título>Microsoft multado por non permitir escoller navegador en Windows 7
    Service Pack 1</título>
    <contido>As autoridades europeas da Competencia, veñen de multar a <empresa>
    Microsoft</empresa> por non incluír a posibilidade de escoller navegadores
    alternativos (a <navegador>IE Explorer</navegador>) en <sistema_operativo>
    Windows</sistema_operativo> <versión>7</versión> <releasa>Service Pack 1</
    releasa>, aínda que se tiña comprometido a elo.</contido>
  </nova>
</novas>
```

Podemos aplicar a seguinte transformación.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:strip-space elements="*" />
  <xsl:template match="/">
    <xsl:copy-of select="." />
  </xsl:template>
</xsl:stylesheet>

```

E obteremos a seguinte saída:

```

<?xml version="1.0" encoding="UTF-8"?><novas><nova><título>Microsoft multado
por non permitir escoller navegador en Windows 7 Service Pack
1</título><contido>As autoridades europeas da Competencia, veñen de multar a
<empresa>Microsoft</empresa> por non incluír a posibilidade de escoller na-
vegadores alternativos (a <navegador>IE Explorer</navegador>) en <siste-
ma_operativo>Windows</sistema_operativo><versión>7</versión><release>Service
Pack 1</release>, aínda que se tiña comprometido a
elo.</contido></nova></novas>

```

En ocasións queremos eliminar tódolos espazos dun documento agás algúns. Por exemplo, o documento orixe anterior contén algúns nodos que conteñen soamente espazos entre "<sistema_operativo>", "<versión>" e "<release>". Estes nodos forman parte do texto da nova e deberían preservarse.

Neste caso podemos botar man do elemento XSLT "<xsl:preserve-space>", semellante a "<xsl:strip-space>" pero cun significado oposto.

Cando un mesmo elemento está referenciado por un elemento "<xsl:strip-space>" e por outro elemento "<xsl:preserve-space>", aplícase aquela na que o contido do atributo "elements" sexa máis específico.

Por exemplo, cambiando a transformación anterior pola seguinte.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:strip-space elements="*" />
  <xsl:preserve-space elements="contido" />
  <xsl:template match="/">
    <xsl:copy-of select="." />
  </xsl:template>
</xsl:stylesheet>

```

Agora manteranse os espazos dentro do elemento "<contido>".

```

<?xml version="1.0" encoding="UTF-8"?><novas><nova><título>Microsoft multado
por non permitir escoller navegador en Windows 7 Service Pack
1</título><contido>As autoridades europeas da Competencia, veñen de multar a
<empresa>Microsoft</empresa> por non incluír a posibilidade de escoller na-
vegadores alternativos (a <navegador>IE Explorer</navegador>) en <siste-
ma_operativo>Windows</sistema_operativo> <versión>7</versión> <relea-
se>Service Pack 1</release>, aínda que se tiña comprometido a
elo.</contido></nova></novas>

```

1.10 Claves e identificadores

Ao procesar un documento XML, podemos crear claves para identificar elementos do documento orixinal. Os procesadores XSLT soen crear índices internos para as claves, o que mellora o acceso aos elementos.

Para crear unha clave emprégase o elemento "<xsl:key>". Este elemento debe ser fillo

directo de "<xsl:stylesheet>". Non pode empregarse en ningún outro sitio, como por exemplo formando parte dun patrón.

Ten 3 atributos requiridos:

- "name". Asigna un nome á clave para poder empregala con posterioridade.
- "match". Indica os nodos que serán indexados pola clave.
- "use". Define o elemento ou a propiedade dos nodos que se empregará para crear o índice. O valor dese elemento ou propiedade será o que usaremos para acceder aos nodos do índice.

Por exemplo, se quixéramos crear unha clave para as máquinas do documento XML orixe que figura na tarefa 5, empregando a propiedade "nome", faríamos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="c_máquinas" match="máquina" use="@nome" />
  ...
```

1.10.1 Acceder aos nodos da clave

Unha vez creada a clave, empregando a función "key" obtemos nodos da clave. Debemos indicar como parámetros o nome da clave, e o valor do elemento ou propiedade definido no atributo "use".

Por exemplo, se máis adiante (dentro dun patrón) quixéramos obter o "fabricante" da máquina con nome "COPERNICO", poderíamos facer:

```
<xsl:value-of select="key('c_máquinas', 'COPERNICO')/hardware/fabricante" />
```

Cando o valor da clave non é único, a función "key" devolverá o conxunto de nodos que se corresponden co valor que indicamos, non un único nodo. Por exemplo, coa clave:

```
<xsl:key name="t_máquinas" match="máquina" use="tipo" />
```

A función "key('t_máquinas', 'Semitorre')" devolverá tódolos nodos correspondentes a aquelas máquinas de tipo "Semitorre".

1.10.2 Agrupamento de nodos

Unha aplicación interesante das claves é o agrupamento de nodos que non están directamente relacionados no documento orixe.

Para facelo necesitamos coñecer unha función específica de XSLT: "generate-id". Esta función recibe como parámetro un nodo, e devolve un valor único xerado aleatoriamente. Este valor será único en todo o documento, e ademais será sempre o mesmo para ese nodo e esa execución, é dicir, se máis adiante empregamos de novo a función "generate-id" co mesmo nodo obteremos o mesmo valor.

Esta función pode empregarse por exemplo para xerar identificadores e enlaces internos aos mesmos nun documento HTML.

```
...
<!-- Poñemos unha áncora -->
<a name="{generate-id(.)}">
  <xsl:value-of select="nome" />
</a>
...
```



```
<!-- e xeramos un enlace á ancora -->
<a href="#{generate-id(.)}">Para ir ao elemento, preme aquí.</a>
...
```

Para crear grupos de nodos empregando claves e a función "generate-id", deberemos seguir estes pasos:

- Definir unha clave para a propiedade ou elemento que queremos empregar para agrupar.
- Seleccionar os primeiros nodos de cada un dos grupos que imos facer (os primeiros nodos para cada valor distinto da clave).
- Seleccionar os nodos co mesmo valor que o nodo do paso anterior.

Por exemplo, partindo do seguinte documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<alumnos>
  <alumno>
    <nome>Perico dos Palotes</nome>
    <nota>8</nota>
  </alumno>
  <alumno>
    <nome>Arsenio Lupin</nome>
    <nota>7</nota>
  </alumno>
  <alumno>
    <nome>Frodo Bolson</nome>
    <nota>7</nota>
  </alumno>
  <alumno>
    <nome>Smeagol</nome>
    <nota>6</nota>
  </alumno>
  <alumno>
    <nome>Pulgarcito</nome>
    <nota>8</nota>
  </alumno>
</alumnos>
```

Queremos facer un documento de saída con **un grupo por cada valor da nota.** Por tanto, deberemos definir a seguinte clave:

```
<xsl:key name="c_alumno" match="alumno" use="nota"/>
```

Despois seleccionaremos o primeiro nodo de cada valor, isto é, o primeiro nodo co valor 8, o primeiro nodo co valor 7, e o primeiro nodo co valor 6. Para isto necesitamos unha expresión un tanto complexa:

```
<xsl:for-each select="/alumnos/alumno[generate-id(.)=generate-id(key('clave_alumno',nota))]">
```

O funcionamento da expresión é o seguinte:

- O bucle recorre tódolos nodos dos alumnos, quedándose soamente con aqueles nos que se cumpra a condición da expresión XPath.
- A función "key('clave_alumno',nota)" devolve tódolos alumnos coa mesma nota do nodo que esta a procesar o bucle.
- Cando lle aplicamos a función "generate-id(key('clave_alumno',nota))" obtemos o ID único correspondente ao primeiro nodo con esa mesma nota.

- Por último, se o ID do nodo que estamos a procesar coincide co ID de ese nodo (é o mesmo nodo), cómprese a condición e se procesa o nodo (é o primeiro nodo correspondente ao grupo de nodos coa súa mesma nota).

Para rematar, dentro deste bucle temos que empregar outro bucle para percorre tódolos nodos coa súa mesma nota. Para isto podemos empregar a función "key":

```
<xsl:for-each select="key('clave_alumno',nota)">
```

A transformación completa, ordenando os grupos de maior a menor nota, podería quedar entón:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="xml"/>
  <xsl:key name="clave_alumno" match="alumno" use="nota"/>

  <xsl:template match="/">
    <xsl:element name="Notas">
      <xsl:for-each select="/alumnos/alumno[generate-id(.)=generate-
id(key('clave_alumno',nota))]">
        <xsl:sort select="nota" order="descending" />
        <xsl:element name="Nota">
          <xsl:attribute name="valor">
            <xsl:value-of select="nota" />
          </xsl:attribute>
          <xsl:for-each select="key('clave_alumno',nota)">
            <xsl:element name="Alumno">
              <xsl:value-of select="nome" />
            </xsl:element>
          </xsl:for-each>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

E ao realizar a transformación, obteríamos:

```
<?xml version="1.0" encoding="UTF-8"?>
<Notas>
  <Nota valor="8">
    <Alumno>Perico dos Palotes</Alumno>
    <Alumno>Pulgarcito</Alumno>
  </Nota>
  <Nota valor="7">
    <Alumno>Arsenio Lupin</Alumno>
    <Alumno>Frodo Bolson</Alumno>
  </Nota>
  <Nota valor="6">
    <Alumno>Smeagol</Alumno>
  </Nota>
</Notas>
```



Para rematar faremos as tarefas 3 e 4, onde veremos algúns patróns avanzados para realizar transformacións XSLT.