UD5_Transformación de documentos XML

A_2. Transformacións XSLT sinxelas

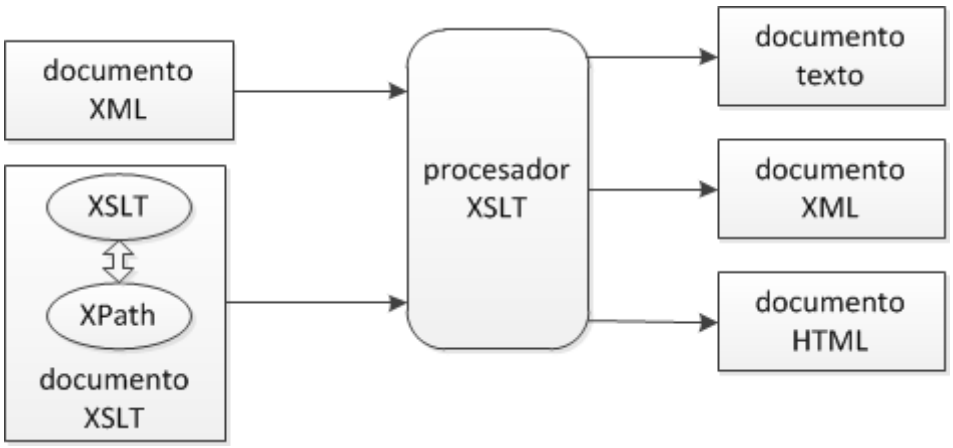
Índice

1. A2. Transformacións XSLT sinxelas

1.1 Que é XSL?

O termo XSL (*eXtensible Stylesheet Language*, Linguaxe de Follas de Estilos Extensible) xurdiu como agrupación dun conxunto de estándares co obxectivo común de transformar e aplicar estilos visuais aos documentos XML. Os estándares que abrangue o termo XSL son:

- XPath. Como xa vimos, é unha linguaxe que serve para construír expresións que busquen e accedan a partes concretas do documento XML.
- XSLT (*XSL Transformations*, Transformacións XSL). É unha linguaxe XML que empréga-se xunto con XPath para converter un documento XML nun documento HTML ou XHTML, nun documento de texto, ou noutro documento XML.



- XSL-FO (*XSL Formatting Objects*, Obxectos de Formato XSL). É outra linguaxe XML, pensada neste caso para dar formato aos datos contidos no documento XML. Traballa con áreas, bloques, rexións, anchuras e alturas das páxinas, das marxes, etc. É moi utilizado para xerar arquivos PDF a partir de documentos XML.

Nos imos a estudar as posibilidades da linguaxe XSLT para realizar transformacións de documentos XML. Existen tres versións de XSLT:

- XSLT 1.0, que foi publicada polo W3C como recomendación en novembro de 1999.
- XSLT 2.0, que aproveita as vantaxes de XPath 2.0, pero non é moi empregada (recomendación de xaneiro de 2007).
- XSLT 3.0, que a comezos de 2013 segue en desenrolo polo W3C (borrador de traballo).

Para facer as transformación precisamos un procesador XSLT, isto é, unha aplicación que comprenda as linguaxes XSLT e XPath e realice sobre o documento XML o procesamento que se indica no documento XSLT.

A maioría dos navegadores web modernos inclúen soporte para XSLT 1.0, pero non para as versións posteriores da linguaxe. Isto é, poden coller un documento XML orixinal e transformalo seguindo as instrucións XSLT nunha páxina web (ou noutro documento XML ou de texto) e amosala.

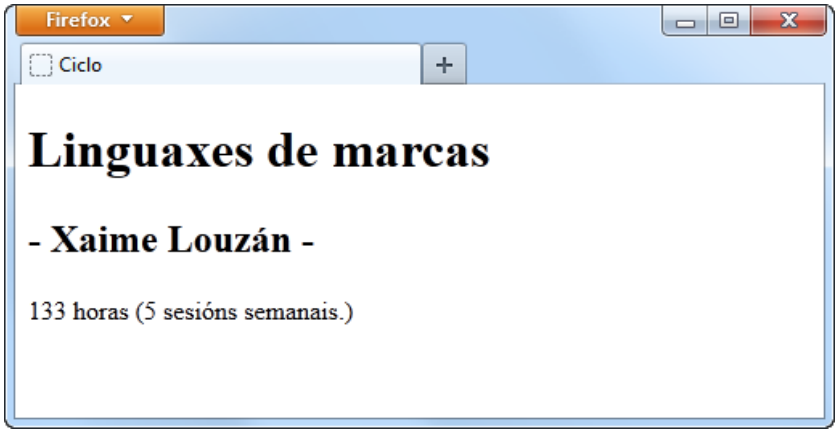
Existen outros procesadores XSLT dispoñibles como aplicacións independentes (non integrados nun navegador web), como Saxon, Xalan ou MSXML. Os tres soportan XSLT 1.0, e algunhas versións de Saxon soportan tamén versións posteriores da linguaxe.

1.2 Transformacións XSLT

Imos comezar cunha transformación sinxela, tomando como orixe o seguinte documento XML:

```
<?xml version="1.0" encoding="utf-8"?>
<ciclo>
   <módulo sesións="5" horas="133">Linguaxes de marcas</módulo>
   <profesor>Xaime Louzán</profesor>
</ciclo>
```

Empregando un navegador web, queremos obter unha páxina HTML como a seguinte:



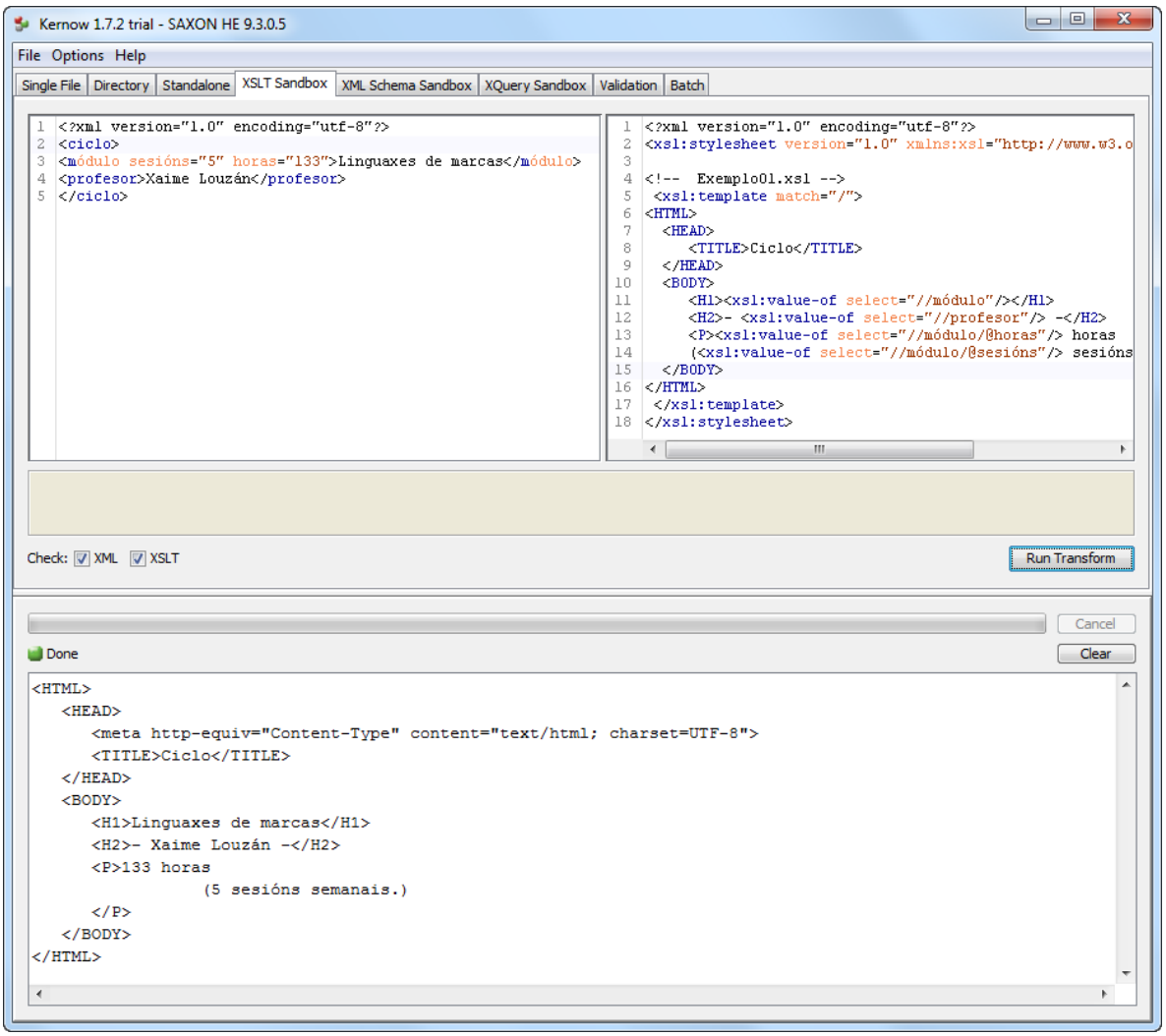
A especificación da transformación a realizar almacénase nun documento XSLT (normalmente con extensión "*.xsl*"). Para que o navegador coñeza o nome do documento XSLT onde se especifica a transformación, deberemos vincular o documento XML con este. Isto faise engadíndolle unha instrución de procesamento como a seguinte:

```
<?xml-stylesheet type="text/xsl" href="Exemplo01.xsl" ?>
```

Esta instrución debe figurar dentro do prólogo, concretamente despois da declaración XML e antes do elemento raíz do documento XML. Por tanto, o documento XML orixinal quedaría:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="Exemplo01.xsl" ?>
<ciclo>
   <módulo sesións="5" horas="133">Linguaxes de marcas</módulo>
   <profesor>Xaime Louzán</profesor>
</ciclo>
```

No caso de empregar outra ferramenta distinta a un navegador para realizar a transformación, tamén é común que exista algún outro xeito de indicar o documento XSLT que queremos aplicar. Por exemplo, se empregamos a ferramenta *Kernow* (baseada no procesador XSLT Saxon), podemos escribir directamente os documentos XML e XSLT e executar a transformación.



No texto de apoio 1 temos un resumo con algunhas das principais ferramentas que podemos empregar para a realización de transformacións XSLT.

En algúns casos queremos empregar un mesmo documento XML orixe para obter varios resultados de saída aplicando distintas transformacións. Por exemplo, a partir dun documento XML cos datos dunha venda, poderíamos xerar unha factura en formato HTML para presentar nun navegador, un documento XML cun resumo da mesma para engadir na contabilidade, e incluso un novo documento HTML específico para visualizar nun terminal móbil.

Tamén deberemos ter en conta que non calquera navegador vai ser capaz de realizar calquera transformación. Por exemplo, o navegador *Chrome* non soporta, por motivos de seguridade, transformacións XSLT cando estean aplicadas a documentos XML locais.

1.2.1 Documentos XSLT

Un documento XSLT é un documento XML cunha estrutura específica. Polo tanto, deberá comezar cun prólogo XML como o seguinte:

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

O elemento raíz dun documento XSLT é normalmente "`<stylesheet>`" (aínda que tamén se pode empregar "`<transform>`", son sinónimos e pódese empregar calquera deles indistintamente), indicando de xeito obrigatorio a versión empregada.

Tamén é preciso facer referencia no documento ao espazo de nomes "*http://www.w3.org/1999/XSL/Transform*". As etiquetas pertencentes a este espazo de nomes serán as que conterán as instrucións destinadas ao procesador XSLT que indican a transformación a realizar.

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

E como os documentos XSLT soen conter tamén outras etiquetas non destinadas directamente ao procesador XSLT, é aconsellable empregar un prefixo para o espazo de nomes (e non poñelo como espazo de nomes por defecto). Pódese coller calquera prefixo, pero é común empregar "*xsl*" (así figura nos exemplos de aquí en diante), de xeito que a estrutura base dun documento XSLT é a seguinte:

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

Por exemplo, o documento XSLT da transformación anterior é:

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

<!--  Exemplo01.xsl -->
 <xsl:template match="/">
<HTML>
  <HEAD>
  <TITLE>Ciclo</TITLE>
  </HEAD>
  <BODY>
  <H1><xsl:value-of select="//módulo"/></H1>
  <H2>- <xsl:value-of select="//profesor"/> -</H2>
  <P><xsl:value-of select="//módulo/@horas"/> horas
  (<xsl:value-of select="//módulo/@sesións"/> sesións semanais.)</P>
  </BODY>
</HTML>
 </xsl:template>
</xsl:stylesheet>
```

Como vemos, o espazo de nomes "`http://www.w3.org/1999/XSL/Transform`" emprégase para identificar as instrucións destinadas ao procesador XSLT, de forma que non intente interpretar as outras etiquetas do documento como "`<HTML>`".

1.3 Formatos de saída

Coa etiqueta "`<xsl:output>`" definimos as características do documento de saída.

```
<xsl:output
 method="xml|html|text"
```

```
 indent="yes|no"
 version="string"
 encoding="string"
 omit-xml-declaration="yes|no"
 standalone="yes|no"
 doctype-public="string"
 doctype-system="string"
 cdata-section-elements="namelist"
 media-type="string"
/>
```

Tódolos atributos da etiqueta son opcionais. É importante o atributo "*method*", que indica o formato do documento resultante. Se non se inclúe, o formato de saída por defecto é XML (a non ser que o elemento raíz do documento resultante sexa "HTML"; nese caso o formato de saída será HTML se non se indica ningún outro, tal e como sucedía no exemplo anterior).

Cando o documento obtido está en formato HTML, normalmente o procesador XSLT emprega sangrado para mellorar a súa lexibilidade. Isto pódese evitar (ou aplicar tamén a outros formatos de saída) empregando o atributo "*indent*".

Aínda que depende do procesador XSLT que empreguemos, na maioría das ocasións se o documento resultante é XML incluirase no mesmo, de forma automática, unha declaración XML. Este comportamento pódese controlar empregando o atributo "*omit-xml-declaration*".

Os atributos "*version*" e "*encoding*" permiten indicar a versión XML e a codificación a empregar no documento XML resultante. Os tipos de codificación admitidos dependen do procesador XSLT, aínda que deberán soportar cando menos "*utf-8*" e "*utf-16*".

Por exemplo, poderíamos modificar a transformación anterior para obter un documento XML engadindo no documento XSLT despois do elemento "`<xsl:stylesheet>`" o seguinte elemento:

```
<xsl:output method="xml" version="1.1" encoding="utf-16" indent="yes"  />
```

Tamén é posible engadir unha declaración de tipo de documento (DTD) ao documento XML resultante, empregando os atributos "*standalone*", "*doctype-public*" e "*doctype-system*".

1.4 Patróns XSLT

O documento XSLT do exemplo anterior contén soamente un elemento como fillo directo do elemento raíz: o elemento "`<xsl:template>`". Estes elementos, que chamaremos patróns, son unha parte fundamental das transformacións XSLT.

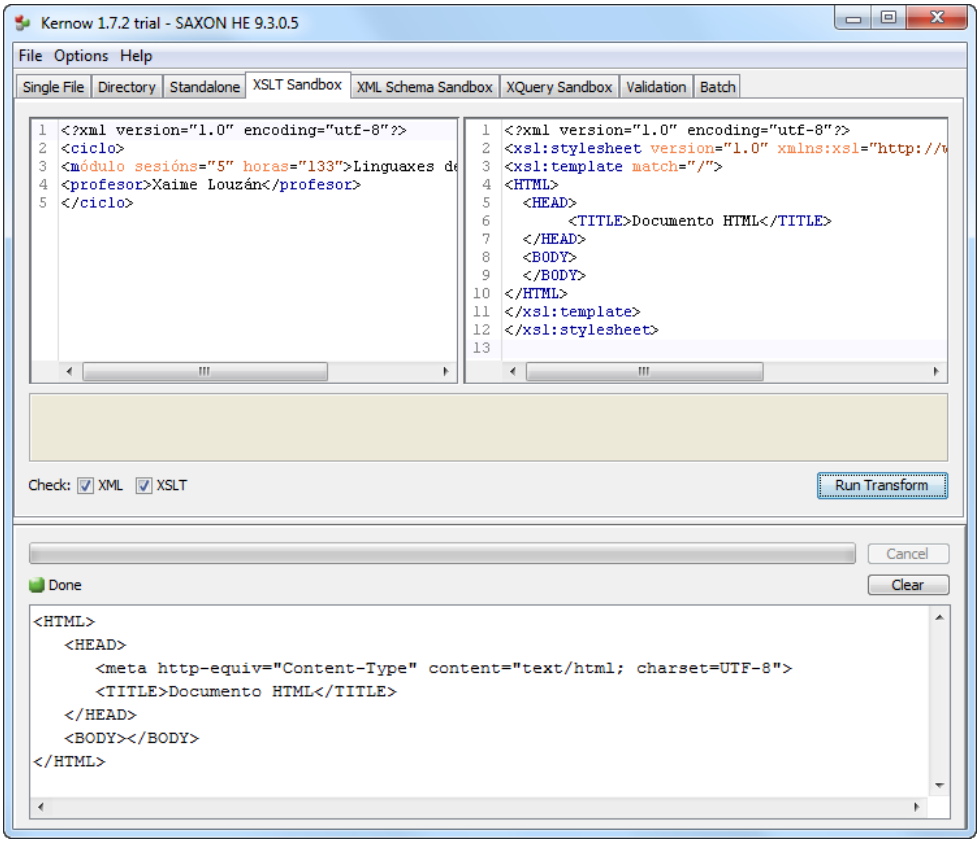
Os patróns indican unha transformación a realizar a certos elementos do documento orixinal. A forma máis habitual de indicar os elementos do documento orixinal aos que se vai a aplicar o patrón, é empregando unha expresión XPath co atributo "*match*". Por exemplo, se queremos aplicar o patrón ao documento XML completo, podemos facer:

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

O patrón pode conter texto e etiquetas (por exemplo, código en formato HTML), que pasarán a copiarse no documento de saída. Por exemplo, o seguinte documento XSLT obtén como resultado sempre o mesmo documento HTML baleiro a partires dun documento XML calquera.

```
<?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>
    <TITLE>Documento HTML</TITLE>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```



A expresión XPath do atributo "*match*" debe coincidir con algún elemento do documento orixinal. O procedemento concreto empregado polo procesador XSLT ao realizar unha transformación é o seguinte:

- O procesador XSLT vai collendo nodos do documento XML orixe comezando polo nodo raíz.
- Para cada nodo busca un patrón que o referencie no seu atributo "*match*".
- Unha vez atopado o patrón, aplica a transformación definida no mesmo ao nodo do documento orixe.

Por exemplo, obteríamos o mesmo resultado da transformación anterior se tivéramos posto:

```
<xsl:template match="/ciclo">
…
</xsl:stylesheet>
```

É moi importante ter en conta que cando o procesador atopa un patrón que fai referencia ao nodo que está a procesar, logo de aplicar a transformación correspondente marca ao nodo e a tódolos seus fillos como procesados, polo que non buscará outro patrón co que transformalos.

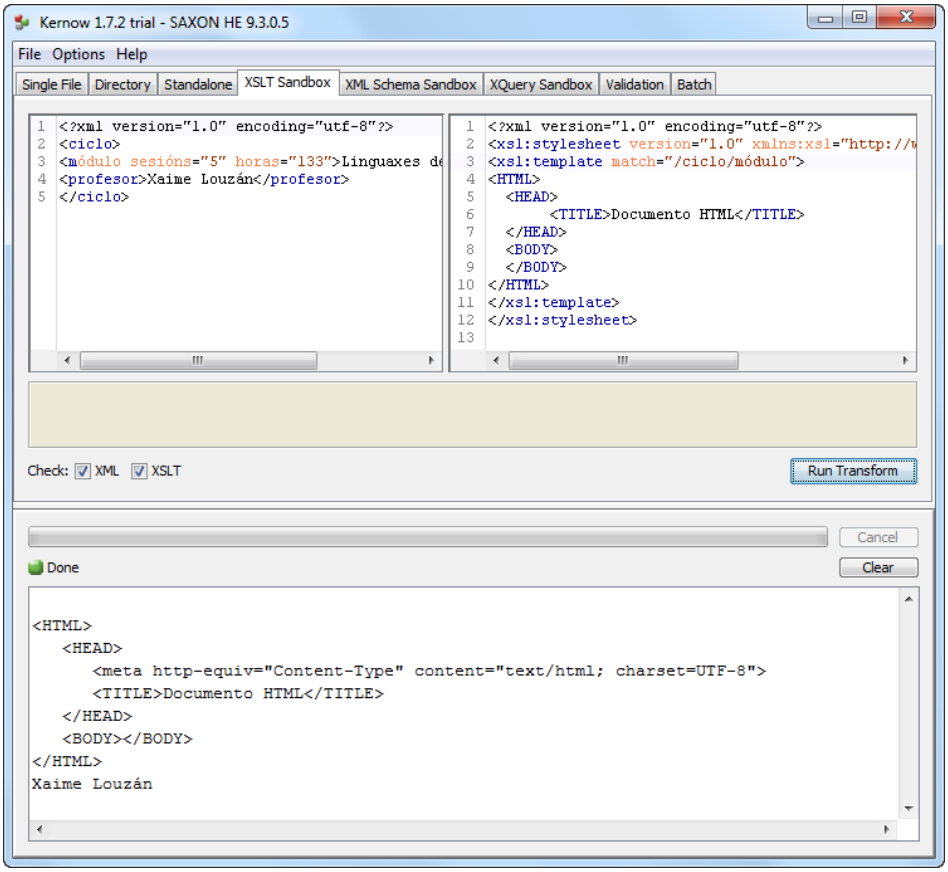
Por exemplo, se o documento XML do exemplo anterior o transformáramos cun XSLT como o seguinte:

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

O primeiro nodo do documento orixe que procesa é "`<ciclo>`", e unha vez aplicada a transformación que indica o seu patrón correspondente, os nodos "`<módulo>`" e "`<profesor>`" tamén quedan marcados como procesados, polo que o segundo patrón do documento XSLT nunca se executará (xa veremos posteriormente como se levan a cabo as transformacións XSLT con varios patróns).

1.4.1 Patróns predefinidos

Cando empregamos a expresión XPath "`/ciclo/módulo`" no atributo "*match*", o resultado obtido non é o que en principio poderiamos esperar.



Isto é debido a unha característica de XSLT chamada patróns predefinidos, que fai que se apliquen uns patróns fixos a aqueles elementos non procesados por ningún outro patrón.

Cando empregábamos as expresións "`/`" ou "`/ciclo`", estas abranguían a tódolos seus fillos e polo tanto ao documento orixinal completo. Non quedaba ningún elemento do documento orixinal sen procesar.

Agora ao empregar a expresión "`/ciclo/módulo`" no patrón, os elementos "`<ciclo>`" e "`<profesor>`" non se atopan cubertos por el; por isto o procesador XSLT aplícalles un patrón predefinido. Basicamente, o comportamento destes patróns predefinidos é procesar o elemento e tódolos seus fillos, copiando ao documento de saída o texto que conteñen.

No noso exemplo:

- Ao procesar o elemento "`<ciclo>`", o procesador XSLT non atopa un patrón e aplica o patrón predefinido. Este copia ao documento de saída o salto de liña que existe antes do elemento "`<módulo>`". Por iso aparece unha liña baleira ao comezo do documento resultante.
- O elemento "`<módulo>`" ven recollido no patrón "`/ciclo/módulo`", e ao procesalo cópianse no documento de saída as etiquetas correspondentes ao documento HTML baleiro.
- Por último, o procesador XSLT chega ao elemento "`<profesor>`" que tampouco ten un patrón específico, co cal copia tamén o seu contido (o nome do profesor) ao documento de saída.

Fíxate que ao contrario que sucede cos patróns que nos definimos no documento XSLT, ao empregar un patrón predefinido non se marcan como procesados os fillos do nodo correspondente do documento XML.

Imos facer a tarefa 1, na que crearemos a nosa primeira especificación dunha transformación XSLT.

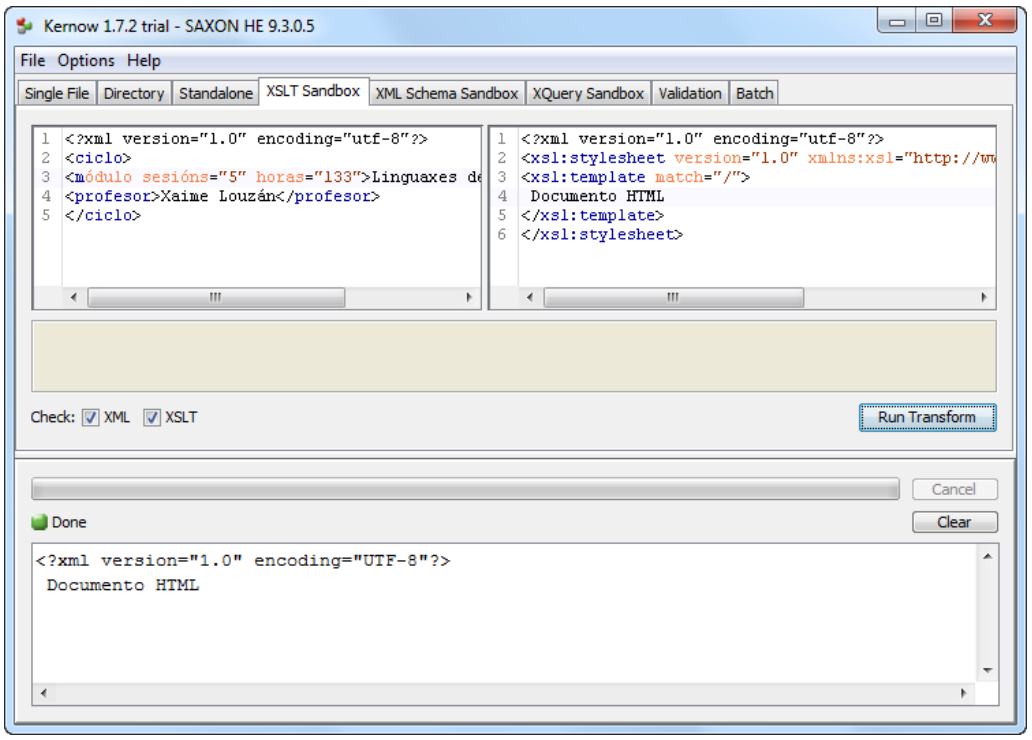
1.5 Creación de texto e elementos

Existen algúns elementos XSLT que podemos engadir dentro dos patróns, e que nos permiten crear novo contido no documento resultante.

1.5.1 Texto

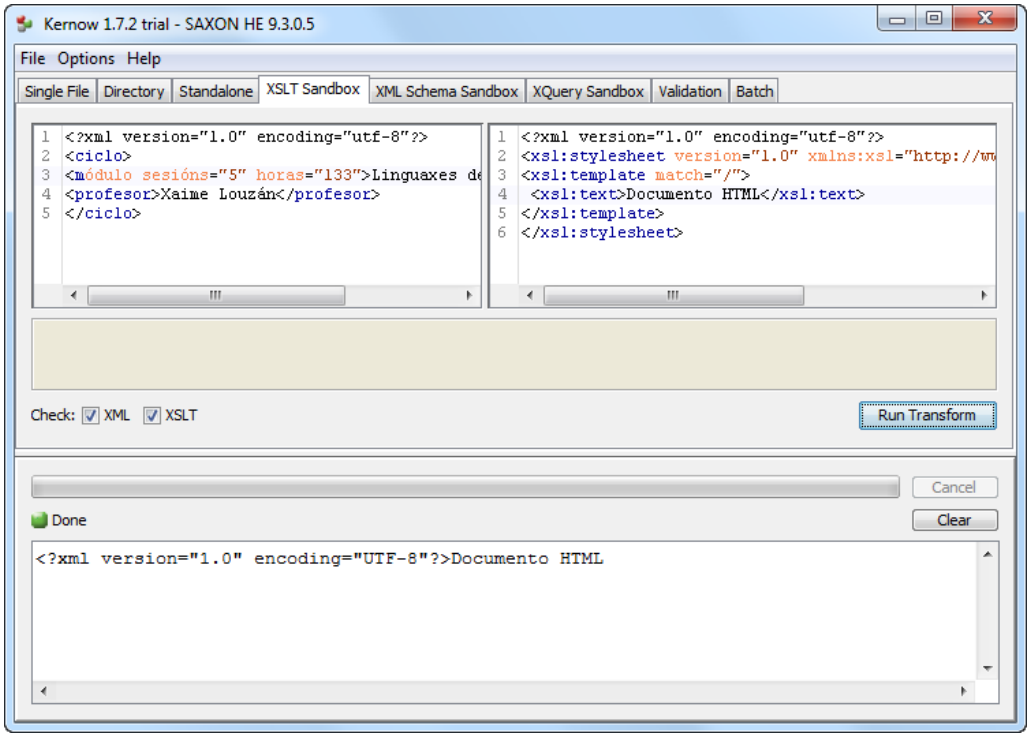
O elemento "`<xsl:text>`" emprégase para incluír texto no documento resultante. O resultado é semellante a escribir directamente o texto dentro do patrón, pero neste caso temos maior control sobre os espazos e os saltos de liña. Por exemplo, o resultado da transformación:

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



E o resultado da seguinte diferéncianse soamente nun salto de liña e uns espazos:

```
<?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:text>Documento HTML</xsl:text>
</xsl:template>
</xsl:stylesheet>
```



O elemento "<xsl:text>" deberá conter soamente texto, non outras etiquetas. Por exemplo, o seguinte documento XSLT dará error ao procesalo.

```
<?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:text><HTML></xsl:text>
</xsl:template>
</xsl:stylesheet>
```

1.5.2 Elementos

Dentro dun patrón podemos empregar "<xsl:element>" para crear un novo elemento no documento de saída. É obrigatorio indicar o nome do novo elemento mediante o atributo "name". Por exemplo, se quixeramos obter un documento de saída cun elemento raíz "<documento>" baleiro, poderiamos facer:

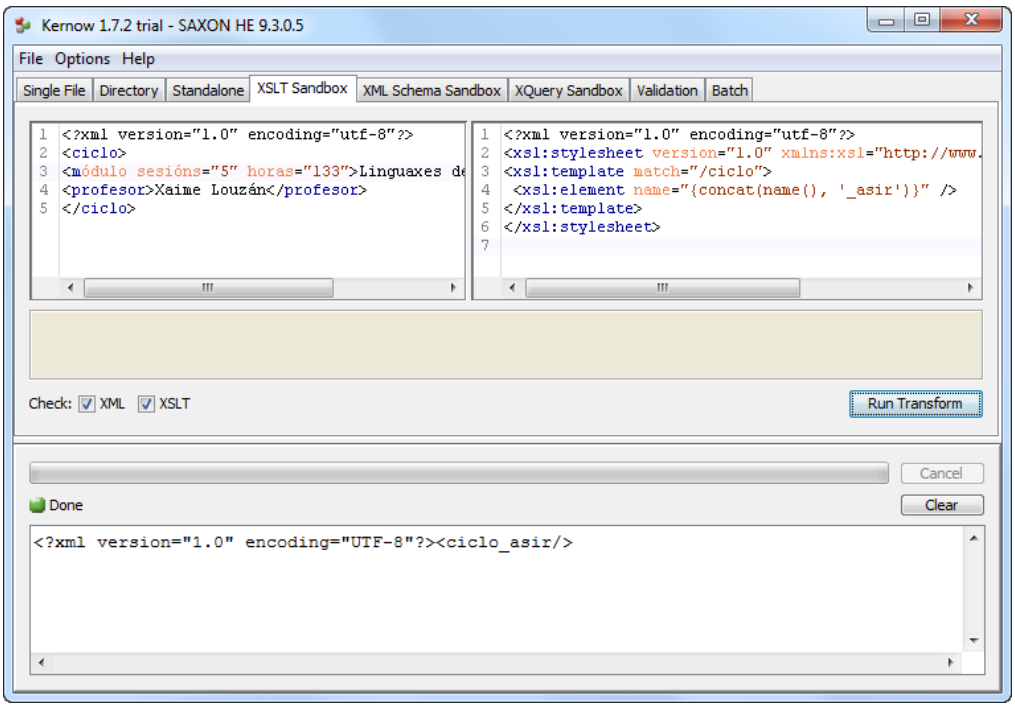
```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <documento />
</xsl:template>
</xsl:stylesheet>
```

Ou tamén:

```
<?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:element name="documento" />
</xsl:template>
</xsl:stylesheet>
```

Un bo motivo para empregar "<xsl:element>" é que o contido do atributo "name" pode estar calculado empregando expresións XPath (por exemplo, a partires de texto, variables, valores retornados por funcións, etc). Neste caso, a expresión debe figurar entre chaves.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/ciclo">
 <xsl:element name="{concat(name(), '_asir')}" />
</xsl:template>
</xsl:stylesheet>
```



1.5.3 Atributos

Cando creamos un novo elemento no documento de saída empregando "`<xsl:element>`", tamén podemos especificar os seus atributos. Simplemente teremos que engadir como fillos deste tantos elementos "`<xsl:attribute>`" como necesitemos, indicando en cada un deles o seu nome co atributo "`name`". O seu valor será o texto que conteña.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/ciclo">
 <xsl:element name="{concat(name(), '_asir')}">
  <xsl:attribute name="duración">2000 horas</xsl:attribute>
 </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

1.6 Selección de valores do documento orixe

Nalgúns casos necesitamos obter o valor dun elemento ou atributo do documento orixe para empregalo como parte do documento de saída. Para isto empregaremos "`<xsl:value-of>`", indicando co atributo "`select`" a expresión XPath que identifique o contido a extraer.

Por exemplo, se partindo do seguinte documento XML:

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

Queremos crear unha transformación que obteña:

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

Teremos que aplicarlle un 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="/módulo/profesor">
 <xsl:element name="{name()}">
  <xsl:attribute name="nome">
   <xsl:value-of select="text()" />
  </xsl:attribute>
 </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

O elemento "`<xsl:value-of>`" tamén admite o atributo opcional "`disable-output-escaping`", que pode tomar os valores "`yes`" ou "`no`".

Co valor "`yes`" indícase que os caracteres especiais do documento orixe, como "`<`", deben ser pasados tal cual ao documento de saída. Co valor "`no`", que é a opción por defecto se non se inclúe o atributo, indícase que os caracteres especiais deben ser transformados nas súas respectivas entidades (por exemplo, "`&lt;`").

Na tarefa 2 procesaremos un documento XML creando novos elementos e atributos para o documento de saída.

1.7 Conxuntos de atributos

Tamén é posible definir un conxunto de atributos para despois empregalo en un ou varios elementos. O conxunto de atributos defínese empregando "`<xsl:attribute-set>`". Débese indicar o nome do conxunto cun atributo "`name`", por exemplo:

```
<xsl:attribute-set name="atr_módulo">
  <xsl:attribute name="nome">
    <xsl:value-of select="."/>
  </xsl:attribute>
  <xsl:attribute name="sesións_anuais">
    <xsl:value-of select="@horas * 1.2"/>
  </xsl:attribute>
</xsl:attribute-set>
```

A definición do conxunto de atributos debe facerse como fillo directo do elemento raíz "`<xsl:stylesheet>`", fora de calquera patrón.

Para empregar o conxunto na definición dun elemento, engádeselle o atributo "`use-attribute-sets`". Por exemplo:

```
<xsl:element name="módulo" use-attribute-sets="atr_módulo">
```

Deste xeito, aplicando a seguinte transformación ao documento XML co que vimos traballando.

```
<?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:attribute-set name="atr_módulo">
	<xsl:attribute name="nome">
		<xsl:value-of select="ciclo/módulo" />
	</xsl:attribute>
	<xsl:attribute name="sesións_anuais">
		<xsl:value-of select="ciclo/módulo/@horas * 1.2"/>
	</xsl:attribute>
</xsl:attribute-set>
<xsl:template match="/">
	<xsl:element name="módulo" use-attribute-sets="atr_módulo">
	</xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Obteríamos como resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
<módulo nome="Linguaxes de marcas" sesións_anuais="159.6"/>
```

1.8 Creación de comentarios e instrucións de procesamento

1.8.1 Comentarios

Para crear un comentario no documento de saída teremos que empregar "`<xsl:comment>`". Por exemplo, poñendo:

```
<xsl:comment>Primeiro exemplo</xsl:comment>
```

Obteremos:

```
<!--Primeiro exemplo-->
```

1.8.2 Instrucións de procesamento

Engádense empregando "`<xsl:processing-instruction>`" e indicando obrigatoriamente o seu atributo "`name`". Por exemplo, poderiamos asociar o documento XML resultante cun documento XSLT facendo:

```
<?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:processing-instruction name="xml-stylesheet">href="profes.xsl"
type="text/xml"</xsl:processing-instruction>
 <!--
</xsl:template>
</xsl:stylesheet>
```

O resultado obtido é:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="profes.xsl" type="text/xml"?>
…
```

1.9 Documentos XSLT con varios patróns

Xa vimos o que ocorre cando non existe un patrón para algún elemento do documento XML orixe. E ata o de agora estivemos a empregar documentos XSLT cun único patrón. Imos ver que ocorre cando temos un documento XSLT con varios patróns, 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:output method="xml" indent="yes" />
<xsl:template match="/módulo/profesor">
   <Profesor />
</xsl:template>
<xsl:template match="/módulo">
   <Módulo />
</xsl:template>
</xsl:stylesheet>
```

Cando aplicamos a anterior transformación ao documento XML.

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

Obtemos como saída.

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

Isto é, o primeiro patrón non chega a procesarse. Cal é a explicación a este comportamento? O motivo é que o procesador XSLT vai collendo os nodos do documento orixinal de un en un, comezando co elemento raíz, e buscando algún patrón que se lle poida aplicar.

Polo tanto o elemento raíz do documento orixe, "`<módulo>`", é o primeiro en ser procesado, e con el tódolos seus fillos. O patrón seleccionado polo procesador XSLT para este elemento é "`<xsl:template match="/módulo">`", e como resultado cópiase "`<Módulo />`" no documento de saída.

Unha vez procesados o elemento raíz xunto cos seus fillos, xa non quedan máis nodos por procesar no documento orixinal, co cal o patrón "`<xsl:template match="/módulo/profesor">`" non chega a executarse.

1.9.1 Procesar un elemento e os seus fillos

Existe unha forma de facer que no procesamento dun elemento se procesen tamén os patróns correspondentes aos seus fillos. Isto faise empregando "`<xsl:apply-templates>`" dentro dun patrón.

Por exemplo, se aplicásemos a seguinte transformación ao documento anterior.

```
<?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" method="text"/>
    <xsl:template match="/">
       <xsl:apply-templates />
    </xsl:template>
</xsl:stylesheet>
```

Ao elemento "profesor" aplícalle o patrón predefinido, e polo tanto copia o texto que contén na saída, obtendo:

```
Xaime Louzán
```

Outro exemplo; no caso anterior poderíamos modificar o documento XSLT poñendo:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" />
<xsl:template match="/módulo/profesor">
   <Profesor />
</xsl:template>
<xsl:template match="/módulo">
   <xsl:element name="Módulo">
     <xsl:apply-templates select="profesor" />
   </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Neste caso, ao procesar o elemento raíz "<módulo>", crea o elemento "<Módulo>" no documento de saída, e despois busca algún patrón adecuado para procesar ao elemento "<profesor>", o que fai que se execute o patrón "<xsl:template match="/módulo/profesor">". O resultado sería:

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

É moi habitual atopar documentos XSLT coa seguinte estrutura:

```
<?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:apply-templates />
   …
</xsl:template>

<xsl:template match="nodos1">
   …
  <xsl:apply-templates />
   …
</xsl:template>

<xsl:template match="nodos2">
   …
  <xsl:apply-templates />
   …
</xsl:template>
</xsl:stylesheet>
```

En ocasións non queremos que se procesen todos os fillos dun elemento. Nestes casos podemos empregar o atributo "select" de "<xsl:apply-templates>" para indicar a expresión XPath que se corresponda con aqueles nodos que queremos que se procesen. Por exemplo:

```
<xsl:apply-templates select="profesor" />
```

Neste caso, se o elemento "<módulo>" tivera varios fillos, soamente continuaría o procesamento de "<profesor>".

Imos facer agora as tarefas 3 e 4, nas que crearemos especificacións de transformacións compostas por varios patróns.

2. TA1. Software para a realización de transformacións XSLT

Existen moitas aplicacións que nos permiten aplicar unha transformación XSLT a un documento XML. A máis común delas é o navegador. Cando abrimos un documento XML nun navegador moderno, na gran maioría dos casos buscará a folla de estilo XSLT asociada e realizará a transformación amosando o resultado obtido. Pero cando queiramos comprobar o documento de saída obtido, na opción de ver o código fonte asociado á páxina poderemos ver so o documento XML orixinal.

O resto de ferramentas existentes baséanse maiormente no uso de tres procesadores XSLT: Xalan (http://xalan.apache.org/), Saxon (http://saxon.sourceforge.net/) e MSXML. A súa principal vantaxe é facilitar o uso do procesador, evitando o emprego da liña de comandos, e amosando inmediatamente o documento de saída. Nalgúns casos poderemos incluso escoller o procesador a empregar entre dúas ou máis opcións.

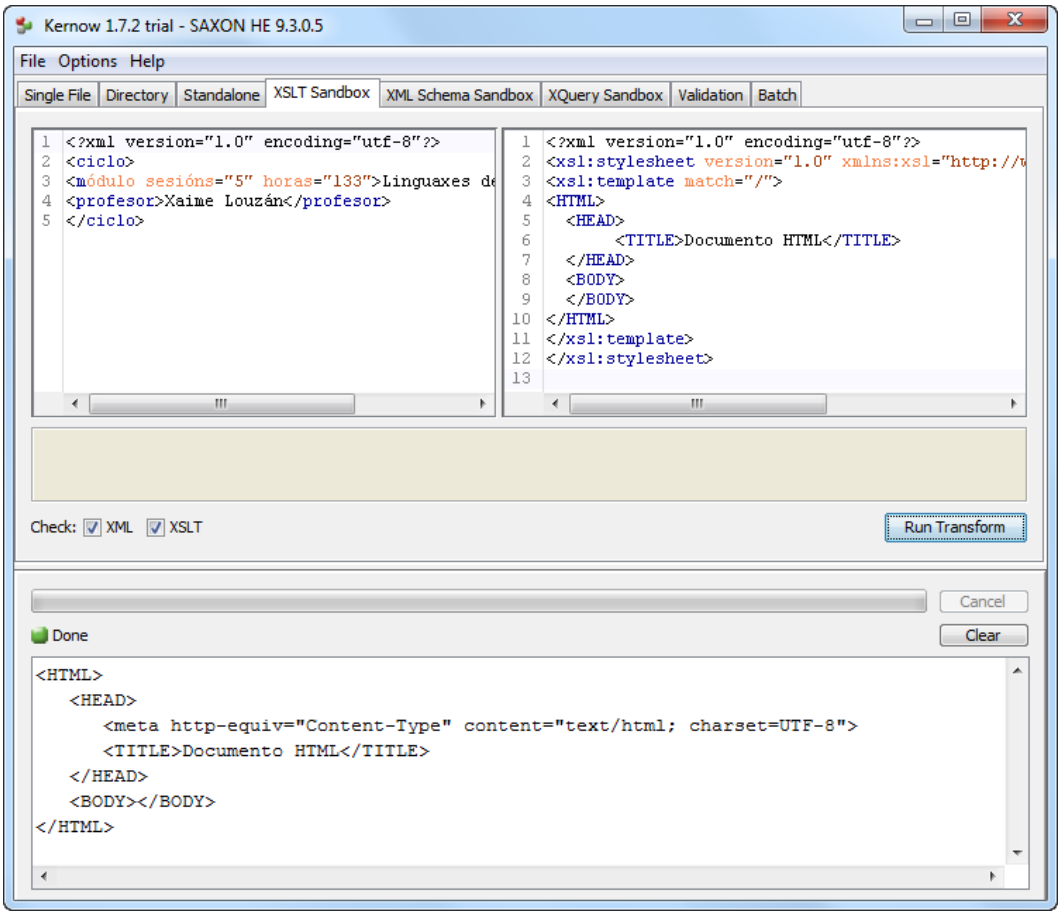
Na páxina "http://edutechwiki.unige.ch/en/XML_editor" temos unha lista moi extensa de aplicacións de edición de documentos XML. Unhas cantas permiten tamén outras opcións, como a comprobación da validez dun documento en base a un DTD ou a un XML Schema, e a realización de transformacións XSLT. Imos ver algunhas das máis empregadas.

XML Copy Editor

Xa falaramos desta ferramenta na UD correspondente á linguaxe XML. Soporta tamén a realización de transformacións XSLT, pero non indica o procesador que emprega nin permite escoller outro.

Kernow

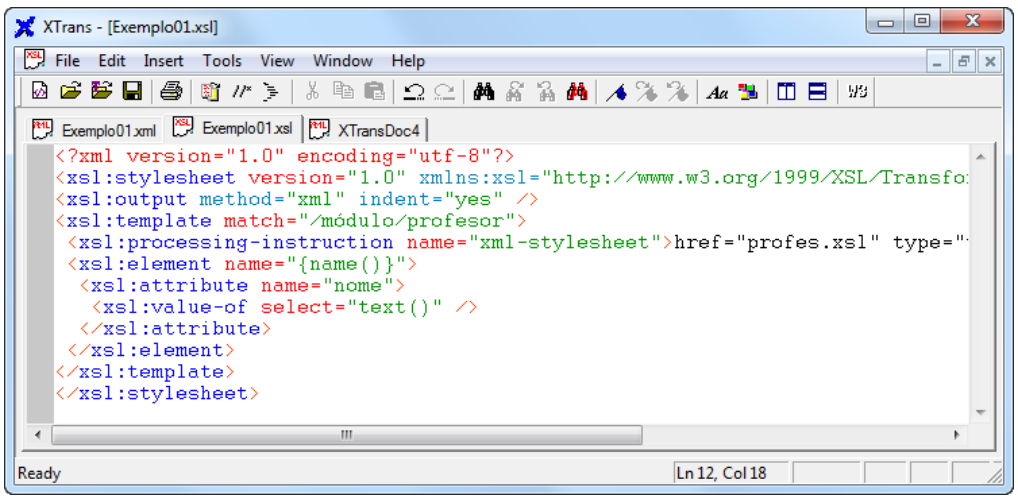
Kernow (http://kernowforsaxon.sourceforge.net/) é unha ferramenta programada en Java, que abrangue un amplo conxunto de funcións relacionadas coa linguaxe XML e pode executarse dende múltiples sistemas operativos. Non é gratuíta, aínda que ten un período de proba e o seu autor ofrece licenzas sen custo para uso non comercial.



Baséase no procesador XSLT Saxon, e a execución das transformacións é moi intuitiva, amosando na mesma pantalla o documento XML orixe, o documento XSLT e o resultado obtido.

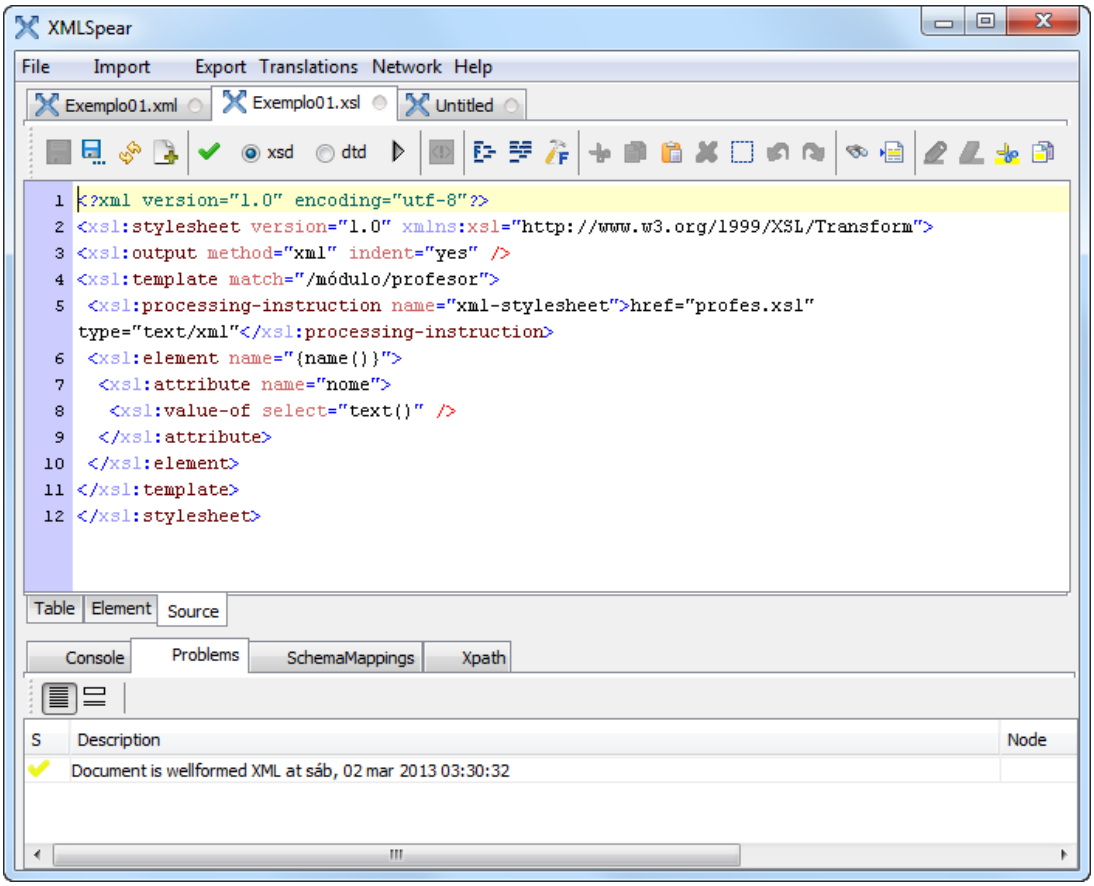
XTrans

É unha ferramenta gratuíta (http://sourceforge.net/projects/xtrans/) para sistemas operativos Windows orientada á realización de transformacións XSLT. Non permite escoller o procesador XSLT a empregar (nin indica o que emprega). Tampouco permite traballar con sistemas de codificación distintos ao "ISO-8859-1", o que ven a ser unha limitación bastante importante.



XMLSpear

XMLSpear (http://www.donkeydevelopment.com/) é un editor de documentos XML gratuíto programado en Java e dispoñible polo tanto para a maioría de sistemas operativos. Ofrece algunhas funcións interesantes, como a validación ou a avaliación de expresións XPath.



Traballa sen problemas con diversos sistemas de codificación de caracteres. No relativo ás transformacións XSLT, permite empregar indistintamente os procesadores Xalan e Saxon.