Transforming TEI documents

Sebastian Rahtz

February 2006

What is the XSL family?

- XPath: a language for expressing paths through XML trees
- XSLT: a programming language for transforming XML
- XSL FO: an XML vocabulary for describing formatted pages

XSLT

The XSLT language is

- Expressed in XML; uses namespaces to distinguish output from instructions
- Purely functional
- Reads and writes XML trees
- Designed to generate XSL FO, but now widely used to generate HTML

How is XSLT used? (1)

- With a command-line program to transform XML (eg to HTML)
  - Downside: no dynamic content, user sees HTML
  - Upside: no server overhead, understood by all clients
- In a web server *servlet*, eg serving up HTML from XML (eg Cocoon, Axkit)
  - Downside: user sees HTML, server overhead
  - Upside: understood by all clients, allows for dynamic changes

How is XSLT used? (2)

- In a web browser, displaying XML on the fly
  - Downside: many clients do not understand it
  - Upside: user sees XML
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations

XSLT implementations

MSXML	Built into Microsoft Internet Explorer
Saxon	Java-based, standards leader, implements XSLT 2.0 (basic version free)
Xerces	Java-based, widely used in servlets (open source)
libxslt	C-based, fast and efficient (open source)
transformiix	C-based, used in Mozilla (open source)

What do you mean, ‘transformation’?

Take this

```
<recipe>
 <title>Pasta for beginners</title>
 <ingredients>
  <item>Pasta</item>
  <item>Grated cheese</item>
 </ingredients>
 <cook>Cook the pasta and mix with the cheese</cook>
</recipe>
```

and make this

```
<html>
 <h1>Pasta for beginners</h1>
 <p>Ingredients:  Pasta Grated cheese</p>
 <p>Cook the pasta and mix with the cheese</p>
</html>
```

How do you express that in XSL?


```
<xsl:stylesheet version="1.0">
 <xsl:template match="recipe">
  <html>
   <h1>
    <xsl:value-of select="title"/>
   </h1>
   <p>Ingredients:
    <xsl:apply-templates select="ingredients/item"/>
   </p>
   <p>
    <xsl:value-of select="cook"/>
   </p>
  </html>
 </xsl:template>

</xsl:stylesheet>
```

Structure of an XSL file


```
<xsl:stylesheet version="1.0">
 <xsl:template match="tei:div">
 </xsl:template>
 <xsl:template match="tei:p">
 </xsl:template>

</xsl:stylesheet>
```

The `tei:div` and `tei:p` are *XPath expressions*, which specify which bit of the document is matched by the template.

Any element not starting with xsl: in a template body is put into the output.

The Golden Rules of XSLT

1. If there is no template matching an element, we process the elements inside it
2. If there are no elements to process by Rule 1, any text inside the element is output
3. Children elements are not processed by a template unless you explicitly say so
4. ```
   xsl:apply-templates select="XX"
   ```
   looks for templates which match element "XX";
   ```
   xsl:value-of select="XX"
   ```
   simply gets any text from that element
5. The order of templates in your program file is immaterial
6. You can process any part of the document from any template
7. Everything is well-formed XML. Everything!
   ```

Technique (1): `apply-templates`

Process everything in the document and make an HTML document:

```
<xsl:template match="/">
 <html>
  <xsl:apply-templates/>
 </html>

</xsl:template>
```

but ignore the `<teiHeader>`

```
<xsl:template match="tei:TEI">
 <xsl:apply-templates select="tei:text"/>

</xsl:template>
```

and do the `<front>` and `<body>` separately

```
<xsl:template match="tei:text">
 <h1>FRONT MATTER</h1>
 <xsl:apply-templates select="tei:front"/>
 <h1>BODY MATTER</h1>
 <xsl:apply-templates select="tei:body"/>

</xsl:template>
```

Technique (2): `value-of`

Templates for paragraphs and headings:

```
<xsl:template match="tei:p">
 <p>
  <xsl:apply-templates/>
 </p>
</xsl:template>
<xsl:template match="tei:div">
 <h2>
  <xsl:value-of select="tei:head"/>
 </h2>
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="tei:div/tei:head"/>
```

Notice how we avoid getting the heading text twice.

Why did we need to qualify it to deal with just `<head>` inside `<div>`?

Technique (3): `choose`

Now for the lists. We will look at the 'type' attribute to decide what sort of HTML list to produce:

```
<xsl:template match="tei:list">
 <xsl:choose>
  <xsl:when test="@type='ordered'">
   <ol>
    <xsl:apply-templates/>
   </ol>
  </xsl:when>
  <xsl:when test="@type='unordered'">
   <ul>
    <xsl:apply-templates/>
   </ul>
  </xsl:when>
  <xsl:when test="@type='gloss'">
   <dl>
    <xsl:apply-templates/>
   </dl>
  </xsl:when>
 </xsl:choose>
</xsl:template>
```

Technique (4): number

It would be nice to get sections numbered, so let us change the template and let XSLT do it for us:

```
<xsl:template match="tei:div">
 <h2>
  <xsl:number level="multiple" count="tei:div"/>
  <xsl:text>.  </xsl:text>
  <xsl:value-of select="tei:head"/>
 </h2>
 <xsl:apply-templates/>

</xsl:template>
```

Technique (5): `number`

We can number notes too:

```
<xsl:template match="tei:note">
 [<xsl:number level="any"/>
 <xsl:text>:  </xsl:text>
 <xsl:apply-templates/>]

</xsl:template>
```

Technique (6): `sort`

Let's summarize some manuscripts, *sorting* them by repository and ID number

```
<xsl:template match="tei:TEI">
 <ul>
  <xsl:for-each select=".//tei:msDescription">
   <xsl:sort select="tei:msIdentifier/tei:repository"/>
   <xsl:sort select="tei:msIdentifier/tei:idno"/>
   <li>
    <xsl:value-of select="tei:msIdentifier/tei:repository"/>:
    <xsl:value-of select="tei:msIdentifier/tei:settlement"/>:
    <xsl:value-of select="tei:msIdentifier/tei:idno"/>
   </li>
  </xsl:for-each>
 </ul>
</xsl:template>
```

Technique (7): @mode

You can process the same elements in different ways using modes:

```
<xsl:template match="/">
 <xsl:apply-templates select=".//tei:div" mode="toc"/>
 <xsl:apply-templates/>
</xsl:template>
<xsl:template match="tei:div" mode="toc">
 Heading <xsl:value-of select="tei:head"/>
</xsl:template>
```

This is a very useful technique when the same information is processed in different ways in different places.

Technique (8): `variable`

Sometimes you want to store some information in a variable

```
<xsl:template match="tei:figure">
 <xsl:variable name="n">
  <xsl:number/>
 </xsl:variable>
 Figure <xsl:value-of select="$n"/>
 <a name="P$n"/>
 <xsl:apply-templates/>
</xsl:template>
```

Technique (9): `template`

You can store common code in a named template, with parameters:

```
<xsl:template match="tei:div">
 <html>
  <xsl:call-template name="header">
   <xsl:with-param name="title" select="tei:head"/>
  </xsl:call-template>
  <xsl:apply-templates/>
 </html>
</xsl:template>
<xsl:template name="header">
 <xsl:param name="title"/>
 <head>
  <title>
   <xsl:value-of select="$title"/>
  </title>
 </head>

</xsl:template>
```

Top-level commands

`<xsl:import href="...">`: include a file of XSLT templates, overriding them as needed

`<xsl:include href="...">`: include a file of XSLT templates, but do not override them

`<xsl:output>`: specify output characteristics of this job

Some useful `xsl:output` attributes

`method="`xml | html | text"
`encoding="`*string"*
`omit-xml-declation="`yes | no"
`doctype-public="`*string"*
`doctype-system="`*string"*
`indent="`yes | no"

An identity transform


```
<xsl:output method="xml" indent="yes" encoding="iso-8859-1" do
<xsl:template match="/">
 <xsl:copy-of select="."/>

</xsl:template>
```

A near-identity transform


```
<xsl:template match="*|@*|processing-instruction()">
 <xsl:copy>
  <xsl:apply-templates select="*|@*|processing-instruction()|co
 </xsl:copy>
</xsl:template>
<xsl:template match="text()">
 <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="tei:p">
 <para>
 <xsl:apply-templates/>
 </para>
</xsl:template>
```

Summary

The core techniques:

- template rules for nodes in the incoming XSL
- taking material from other nodes
- processing nodes several times in different modes
- variables and functions
- choosing, sorting, numbering
- different types of output