Transforming TEI documents

Date: February 2005

1 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

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

2 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

3 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

4 XSLT implementations

MSXML Built into Microsoft Internet Explorer

Saxon Java-based, standards leader (open source)

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)

5 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>
```

6 How do you express that in XSL?

```
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
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>
```

7 Structure of an XSL file

```
<xsl:stylesheet
xmlns:tei="http://www.tei-c.org/ns/1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
version="1.0">
<xsl:template match="tei:div">
<!-  ....  do something with TEI div elements....->

</xsl:template>
<xsl:template match="tei:p">
<!-  ....  do something with TEI p elements....->
```

```
</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.

8 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!

9 Building a TEI stylesheet (1)

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>
```

10 Building a TEI stylesheet (2)

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">
</xsl:template>
```

Notice how we avoid getting the heading text twice.

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

11 Building a TEI stylesheet (3)

Now for the lists. We'll need to 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>
```

Most list items are simple:

```
<xsl:template match="tei:item">
<li>
<xsl:apply-templates/></li>

</xsl:template>
```

12 Building a TEI stylesheet (5)

It would be nice to get those sections numbered, so let's 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>
```

and number the paragraphs as well

```
<xsl:template match="tei:p">
<p>

<xsl:number/><xsl:text>:
</xsl:text>
<xsl:apply-templates/>
</p>

</xsl:template>
```

13 Building a TEI stylesheet (6)

Lets make the lists more interesting by *sorting* ordered ones:

```
<xsl:template match="tei:list[@type='ordered']">
<ol>

<xsl:apply-templates select="tei:item">
<xsl:sort select="."/>
</xsl:apply-templates>
</ol>

</xsl:template>
```

and *counting* unordered ones

```
<xsl:template match="tei:list[@type='unordered']">
There are
<xsl:value-of select="count(tei:item)"/> items

</xsl:template>
```

and adding up gloss lists

```
<xsl:template match="tei:list[@type='gloss']">
The total is
```

```
<xsl:value-of select="sum(tei:label)"/>
</xsl:template>
```

14 Recap

We have met the following XSL basic controls:

```
<xsl:stylesheet>
<xsl:template match="...">
<xsl:apply-templates select="...">
<xsl:value-of select="...">
<xsl:text>
<xsl:choose>
```

... with the following 'extras'

```
<xsl:sort>
<xsl:number>
count()
sum()
XPath axes
qualified matches with []
```

15 Typical XPath expressions in TEI documents

`parent::tei:div/child::tei:head`, the `<head>` element which is a child of my parent `<div>`

`preceding-sibling::tei:p`, all the `<p>` elements before me with the same parent

`descendant::tei:footnote`, all the `<footnote>` elements below me, at any level

`ancestor::tei:div[1]/@id`, the ID attribute of the first `<div>` element above us

`ancestor::tei:TEI.2/child::tei:teiHeader//child::tei:revisionDesc/child::tei:list/p[1]/d`
the `<date>` child of the first `<p>` child of the `<list>` child of the `<revisionDesc>` descendant of the `<teiHeader>` child of the `<TEI.2>` somewhere above where we are now

16 Modes

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.

17 More functions

- `document`: (*string*)
- `generate-id`: (*string*)
- `name`: (*node*)
- `concat`: (*string*, *string*)
- `contains`: (*string*, *string*)
- `substring-before`: (*string*, *string*)
- `substring-after`: (*string*, *string*)

- `string-length`: (*string*)
- `translate`: (*node*,*string*, *string*)
- `normalize-space`: (*string*)

18 Named templates, parameters and variables

`<xsl:template name="...">`: define a named template

`<xsl:call-template>`: call a named template

`<xsl:param>`: specify a parameter in a template definition

`<xsl:with-param>`: specify a parameter when calling a template

`<xsl:variable name="...">`: define a variable

19 Variables

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

20 Named templates

```
 <xsl:template match="/div">
   <html>
    <xsl:call-template name="header">
     <xsl:with-param name="title" select="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>
```

21 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

22 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"`

23 An identity transform

```
<xsl:output
    method="xml"
    indent="yes"
```

```
    encoding="iso-8859-1"
    doctype-system="teixlite.dtd"/>

 <xsl:template match="/">
    <xsl:copy-of select="."/>
 </xsl:template>
```

24 A near-identity transform

```
<xsl:template match="*|@*|processing-instruction()">
 <xsl:copy>
 <xsl:apply-templates
    select="*|@*|processing-instruction()|comment()|text()"/>
 </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>
```

25 TEI Stylesheets

A library of stylesheets for transforming TEI documents to either HTML or XSL Formatting Objects, at http://www.tei-c.org/Stylesheets, with a form-filling interface for the HTML at http://www.tei-c.org/tei-bin/stylebear

26 TEI Stylesheets (example)

An example of an importing stylesheet:

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
<xsl:import
 href="http://www.tei-c.org/Stylesheets/teihtml.xsl"/>
<xsl:param
   name="splitLevel">1</xsl:param>
<xsl:param
   name="numberHeadings"></xsl:param>
<xsl:param
   name="topNavigationPanel">true</xsl:param>
<xsl:param
   name="bottomNavigationPanel">true</xsl:param>
</xsl:stylesheet>
```

27 What is XSLT good for?

Yes, it is useful for making web pages from TEI texts, but it is also a good tool for

- Selecting subsets of our texts for further processing
- Summarizing aspects of our texts
- Checking our text in ways that DTDs and schemas cannot
- Converting text into formats other than HTML or XSL FO

we can solve many of the problems with a small range of techniques.

28 Finding any occurrence of an element

Very often, we will sit on the root element and process all the occurrences of a specific element by using the descendant axis:

```
<xsl:template match="/">
 <html>
   <body>
    Pages: <xsl:value-of select="count(descendant::tei:pb)"/>
   </body>
 </html>
</xsl:template>
```

here we use the count function. We continue to generate an HTML document to provide a convenient reporting format.

29 Sorting occurrences of an element

Instead of just finding the elements, get them out in sorted order:

```
<xsl:template match="/">
  <html>   <body>    <table>
      <xsl:apply-templates select="descendant::tei:div">
        <xsl:sort select="tei:head"/>
      </xsl:apply-templates>
  </table>  </body> </html>
</xsl:template>
<xsl:template match="tei:div">
  <tr>
    <td><xsl:number level="any"/></td>
    <td><xsl:value-of select="tei:head"></td>
  </tr>
</xsl:template>
```

Notice here the `<sort>` child of `<apply-templates>` and the level="any" attribute for `<number>` which provides a count from the start of document regardless of depth.

30 Converting to other formats

In an earlier template, we listed the size of paragraphs:

```
<xsl:template match="tei:p">
    <xsl:variable name="contents">
      <xsl:apply-templates select=".//text()"/>
    </xsl:variable>
    <xsl:number level="any"/>:
    <xsl:value-of select="string-length($contents)"/>
</xsl:template>
```

Adding

```
<xsl:output method="text">
```

will produce pure text output which could be loaded into a spreadsheet or database.

31 XSLT extensions

Although we will not be covering them here, most XSLT processors support various extensions, which will be formalized in version 2.0:

- The ability to create *multiple* output files from one input
- The ability to 'escape' to another language (eg Java) for special purposes
- The ability to turn results into input trees for further processing