Introduction to XSLT

Sebastian Rahtz
February 2004

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

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
  - 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: most clients do not understand it
  - Upside: user sees XML
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations

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>Cook the pasta and mix with the cheese
</html>
```

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>
```

Structure of an XSL file

```
<xsl:stylesheet
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
    version="1.0">
<xsl:template match="XXX">
....  do something with XXX elements....
</xsl:template>
<xsl:template match="YYY">
....  do something with YYY elements....
</xsl:template>
</xsl:stylesheet>
```

The XXX and YYY 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!

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.2">
<xsl:apply-templates select="text"/>
</xsl:template>
```

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

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

Building a TEI stylesheet (2)

Templates for paragraphs and headings:

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

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

<xsl:template match="div/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>`?

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

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

and number the paragraphs as well

```
<xsl:template match="p">
<p> <xsl:number/>
<xsl:text>:  </xsl:text>
<xsl:apply-templates/>
</p>
</xsl:template>
```

Building a TEI stylesheet (6)

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

```
<xsl:template match="list[@type='ordered']">
<ol> <xsl:apply-templates select="item">
<xsl:sort select="."/>
</xsl:apply-templates>
</ol>
</xsl:template>
```

and *counting* unordered ones

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

and adding up gloss lists

```
<xsl:template match="list[@type='gloss']">
The total is <xsl:value-of select="sum(label)"/>
</xsl:template>
```

XPath

The **XPath** language is how you write the *match* and *select* attributes of `<xsl:template>` and `<xsl:apply-templates>`. It consists of

- An *axis name*, following by ::
- An element name
- A qualifier of that element (inside [])
- (optionally) a / and another XPath expression related to the element

The simplest *child axis* can be omitted, and a // means that any number of child levels can be skipped.

XPath axes

- self
- attribute (shorthand form: `@`)
- child (shorthand form:)
- descendant (shorthand form: `//`)
- descendant-or-self
- ancestor
- ancestor-or-self
- namespace
- following
- preceding
- following-sibling
- preceding-sibling
- parent (shorthand form: `..`)

Typical XPath expressions in TEI documents

`parent::div/head`, the `<head>` element which is a child of my parent `<div>`
`preceding-sibling::p`, all the `<p>` elements before me with the same parent
`descendant::footnote`, all the `<footnote>` elements below me, at any level
`ancestor::div[1]/@id`, the ID attribute of the first `<div>` element above us
`ancestor::TEI.2/teiHeader//revisionDesc/li`
`/p[1]/date`, 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

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