

Lecture 3 XSLT

In which we examine some of the features and uses of the extensible style sheet language XSLT.

1. Overview

An XSL transformation can be used to transform an XML document into another document. We design the XML format to be data-oriented – that is the definition of the nodes and attributes are geared toward the data that we need to represent. In order to view this data we need to transform it.

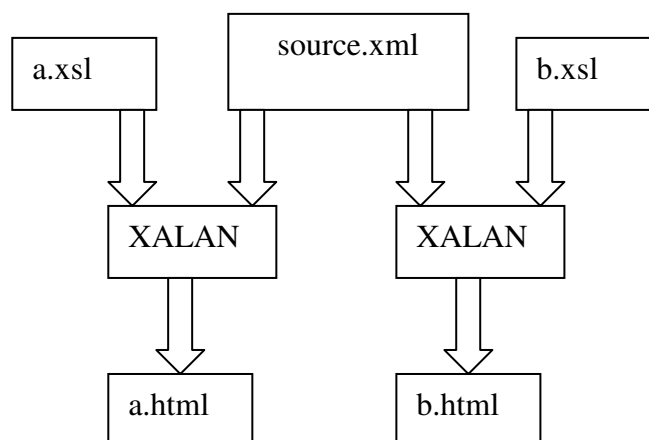
XSL can be used to transform XML into any format we want – perhaps one of the most obvious uses is to convert XML data into HTML that may be easily viewed on a web browser. The XSL document describes this transformation.

The behaviour of the XSL transformation is defined by W3C, there are a number of implementations – many claim to be complete. We will be using the XALAN processor from the Apache/XML project.

XSL looks like XML – sadly there is no DTD for XSL.

2. Typical uses of XSL

We have a source of XML. This source can be filtered by one or more XSL documents to provide different views of the data suitable for different users.



2.1 Generating HTML

In this application of XSL the document source.xml contains all of the data that we have. User A requires to see only some of the data – her requirements are specified in the style sheet a.xsl – when we run xalan against source.xml and a.xsl we get a.html as a result.

Some important results:

- The document a.html is normal html and can be viewed with any browser.
- The users requirements are encapsulated in the xsl document – large numbers of these documents may be built up if that is appropriate.
- The format of the original data source is (mostly) independent of the end users requirements.
- We can allow the XML format to expand over time – in many cases trivial changes to the DTD for source.xml can be made and the style sheets will still work.

3. Get on with it: legend example

We examine some simple XSL documents which may be used to transform our supermarket XML source stock.xml, the style sheet legend.xsl and the result legend.html:

```
<?xml version="1.0"?>
<stock>
<item price="50" legend="Pr-Burger" BarCode="E1"/>
<item price="15" legend="Crisp S+V" BarCode="E5"/>
<item price="15" legend="Crisp C+O" BarCode="E6"/>
<item price="50" legend="Flat Cola" BarCode="E7"/>
</stock>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="item">
    <p><b><xsl:value-of select="@legend"/></b></p>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p><b>Pr-Burger</b></p>
<p><b>Crisp S+V</b></p>
<p><b>Crisp C+O</b></p>
<p><b>Flat Cola</b></p>
```

3.1 The stylesheet: preamble

The preamble to the XSL declares that this is an xml document and provides a namespace declaration. We will not trouble ourselves with this other than to note that it is fixed and is required:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  style sheet nodes...
</xsl:stylesheet>
```

3.2 Template nodes

In this case the stylesheet node contains a single template. This template is set to match any item node in the stock.xml file. As the stock.xml file includes four such nodes it “kicks-off” four times during processing.

```
<xsl:template match="item">
  content...
</xsl:template>
```

Had we chosen `match = "/"` or `match = "stock"` the template would fire off just once.

3.3 Template content

The content of the template is a mixture of HTML tags and an xsl node.

```
<p><b><xsl:value-of select="@legend"/></b></p>
```

The html is copied straight to the output. The `xsl:value-of` node is replaced with a corresponding value.

4. Some XSL rules

- The xsl document is a well formed xml document – this means that the embedded html must be well formed xml as well as being html. It is sensible (but not obligatory) to use xhtml. There are tricks that can be used to output non-xml.
- There are a large number of complicated ways to match nodes. In our example we simply pick out a particular type of node. Other options include:
 - Matching nodes with particular parentage (match a only if it occurs within a <table> for example).
 - Matching nodes in particular positions (match the first node in each list)
 - Matching nodes with specific attributes (match those nodes and <link href= ".." > nodes)

4.1 Questions

- Write an xsl document that will take a well formed xhtml page and output all of the links on that page.
- Write an xsl document that will output a list of the addresses of the images on an xhtml page.
- Write an xsl document that will output just the major headings (h1 tags) – you need to know that <xsl:value-of select="."> gives the text content of a tag

(Beware – we have not discussed name-spaces which are used in many cases. To make these work in practice you can precede matches with htm: as in
 <xsl:template match="htm:a">)

You will also need to declare the namespace htm using a text such as:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:htm="http://www.w3.org/1999/xhtml" >
```

4.2 More matching examples from <http://www.w3.org/TR/xpath#path-abbrev>

para	selects the para element children of the context node
*	selects all element children of the context node
text()	selects all text node children of the context node
@name	selects the name attribute of the context node

@*	selects all the attributes of the context node
para[1]	selects the first para child of the context node
para[last()]	selects the last para child of the context node
*/para	selects all para grandchildren of the context node
/doc/chapter[5]/section[2]	selects the second section of the fifth chapter of the doc
chapter//para	selects the para element descendants of the chapter element children of the context node
//para	selects all the para descendants of the document root and thus selects all para elements in the same document as the context node
//olist/item	selects all the item elements in the same document as the context node that have an olist parent
.	selects the context node
./para	selects the para element descendants of the context node
..	selects the parent of the context node
../@lang	selects the lang attribute of the parent of the context node
para[@type="warning"]	selects all para children of the context node that have a type attribute with value warning
para[@type="warning"][5]	selects the fifth para child of the context node that has a type attribute with value warning
para[5][@type="warning"]	selects the fifth para child of the context node if that child has a type attribute with value warning
chapter[title="Introduction"]	selects the chapter children of the context node that have one or more title children with string-value equal to Introduction
chapter[title]	selects the chapter children of the context node that have one or more title children
employee[@secretary and @assistant]	selects all the employee children of the context node that have both a secretary attribute and an assistant attribute

5. Data oriented XSL processing with one match

For the sake of your sanity you are advised not to delve too deeply into the node tree. There are often many ways to achieve a particular result. In many cases the following will suffice:

Match only the root node. This means that you will have one large `xsl:template` node. Typically:

- Values from the source document are referenced using `xsl:value-of` nodes where the `select` attribute addresses the required value.
- This works well with source documents of known depth – there are no recursively nested structures (such as a table within a table...)
- Repeated elements may be processed using `xsl:for-each`

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

  <xsl:template match="/">
    <p>Number of product lines in stock:
    <xsl:value-of select="count(/stock/item)"/></p>
    <p>Here they are: </p>
    <ul>
      <xsl:for-each select="/stock/item">
        <li><xsl:value-of select="@legend"/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>Number of product lines in stock:
  4</p><p>Here they are: </p><ul><li>Pr-Burger</li><li>Crisp S+V</li><li>Crisp C+O</li><li>Flat
Cola</li></ul>
```

6. Data oriented processing with pattern matching

We can use template rules to achieve something similar to the above. We create a template to match the stock node and another to match each item node.

We must explicitly specify in the stock template that further processing is required:

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

  <xsl:template match="stock">
    <p>Number of product lines in stock:
    <xsl:value-of select="count(item)"/></p>
    <p>Here they are: </p>
    <ul>
    <xsl:apply-templates/>
    </ul>
  </xsl:template>

  <xsl:template match="item">
    <li><xsl:value-of select="@legend"/></li>
  </xsl:template>

</xsl:stylesheet>
```

7. Declarative vs. Imperative

XSL is an attempt at a declarative style of processing.

In imperative (traditional) computing we write programs that go through a sequence of operations – each of which changes the state of some data structure or object.

In a declarative system we attempt to describe the desired outcome and rely on another system to realise the transformation.

8. Built-in template

There are a number of built-in template rules that are included in even a blank stylesheet. One of these causes all text nodes to be copied to the output –

```
<xsl:template match="text()|@"*>
  <xsl:value-of select="."/>
</xsl:template>
```

Commonly we override this with our own version:






```
<xsl:template match="text()|@"*/>
```

9. Narrative oriented processing

Applications such as html have arbitrarily nested structures. Commonly structures will be nested and we can have no guarantees about how deep the nesting will be. In such cases the single matching approach is likely to be more difficult.

It may be that in these cases we are not interested in such precise processing and a simpler stylesheet such as css will suffice.

End of unit summary

-  XSL allows us to define stylesheets which transform XML documents
-  Transformations of appalling complexity are possible
-  DTD plus XSL neatly allow us to specify and separate the data from the application of the data
-  XSL is commonly split into three components XSLT (the transformation) XPath (roughly the expressions that can appear in match and select attributes) and Formatting Objects (we have not covered these)
-  The frightening complexity of XSL may be reduced using the same sort of techniques that we are used to. Templates may be seen as sub-routine. Also we can allow for more than one XSL transformation in order to break down complex procedures and save work.