B336 Advanced Internet Computing

The eXtensible Style Language for Transformation (XSLT)

Learning Objectives

- Understand the role of XSL and XSLT in XML technologies.

- Know the basic syntax of XSLT to transform XML documents.

Learning Objectives

- In the scheme of what we are doing in this unit:
    - We are studying how to use XML as an important set of Internet technologies to use as solutions in different areas (for example, the problem defined in Assignment 2).
    - In most cases, information in XML documents will need to be processed in other formats. XSLT is one of the more standard and popular ways of doing that.

Lecture Outline

- What is XSL and XSLT.
- XPath expressions
- XSLT style sheet syntax.

The eXtensible Style-sheet Language (XSL)

- XSL is an XML-based language used to create style sheets.

- XML software can use an XSL style sheet to transform an XML document into another XML document format, or another non-XML document format.
    - The style sheet defines the format of the output document (**result tree**), and says where to get the data for different parts of the output document from the input document (**source tree**).

XSL

- There are two complete languages under XSL:
    - XSLT
    - XSL Formatting Objects - principally a languages to describe XML documents for display.
- The official recommendations and working-in-progress for both can be found at:

    http://www.w3.org/TR/xsl

Why is XSL and XSLT important?

- Extracting appropriate information.
  - Software and users would probably only require a small part of the information in one XML document.
  - Software and users would probably require information from more than one XML document.

- Having different software
  - Must be able to get the information in an XML document into and out of legacy systems which do not deal with XML.
  - Even systems which deals in XML may not use the same XML language.

XSLT

- The XSLT language basically defines a set of **templates**.

- The templates specifies
  - what to look for in the source tree, and
  - what to put in the result tree.

Associating and XML document to an XSLT style sheet

- An XML document can be associated with a given XSLT style sheet by putting a `<?xml-stylesheet…?>` tag in the *Processing Instructions* parts of the XML document.

- Eg.

```
<?xml version="1.0”?>
<?xml-stylesheet type=“text/xsl” href=“myxsl.xsl”?>
<!DOCTYPE poetry [
     ...
]>
...
```

Associating and XML document to an XSLT style sheet

- This gives the software processing the XML document the option of using that style sheet to transform the document if it deems appropriate.
    - The software will first have to know how to deal with “text/xsl” documents, though.

An XSLT Style Sheet

- Since and XSLT style sheet is also an XML document, it conforms to all the well-formed rules we discussed before.

  - It usually has the `<?xml …?>` declaration.
  - Its root element of the style sheet is `<xsl:stylesheet>.`
  - `<xsl:stylesheet>` contains the zero or more of the element `<xsl:template>` which defines the templates for the tranformations.

An XSLT Style Sheet

- An example:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <xsl:foreach select="/poetry/anthology/poem/stanza/line">
          <P>
            <xsl:value-of select="."/>
          </P>
        </xsl:foreach>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Matching the Source Tree using XPath Expressions

- As we mentioned, a major part of XSLT's job is to get data from a source tree (the original XML document).
- For it to do this, there must be some convenient way of navigating through the source tree. We do this using XPath expressions.
    - Think of XPath as a set of directions for the XSLT software as it is parsing the XML document - eg. "go to the next node", "go deeper into this node", etc.

XPath

- XPath is an official W3C recommendation.
    - http://www.w3.org/TR/xpath

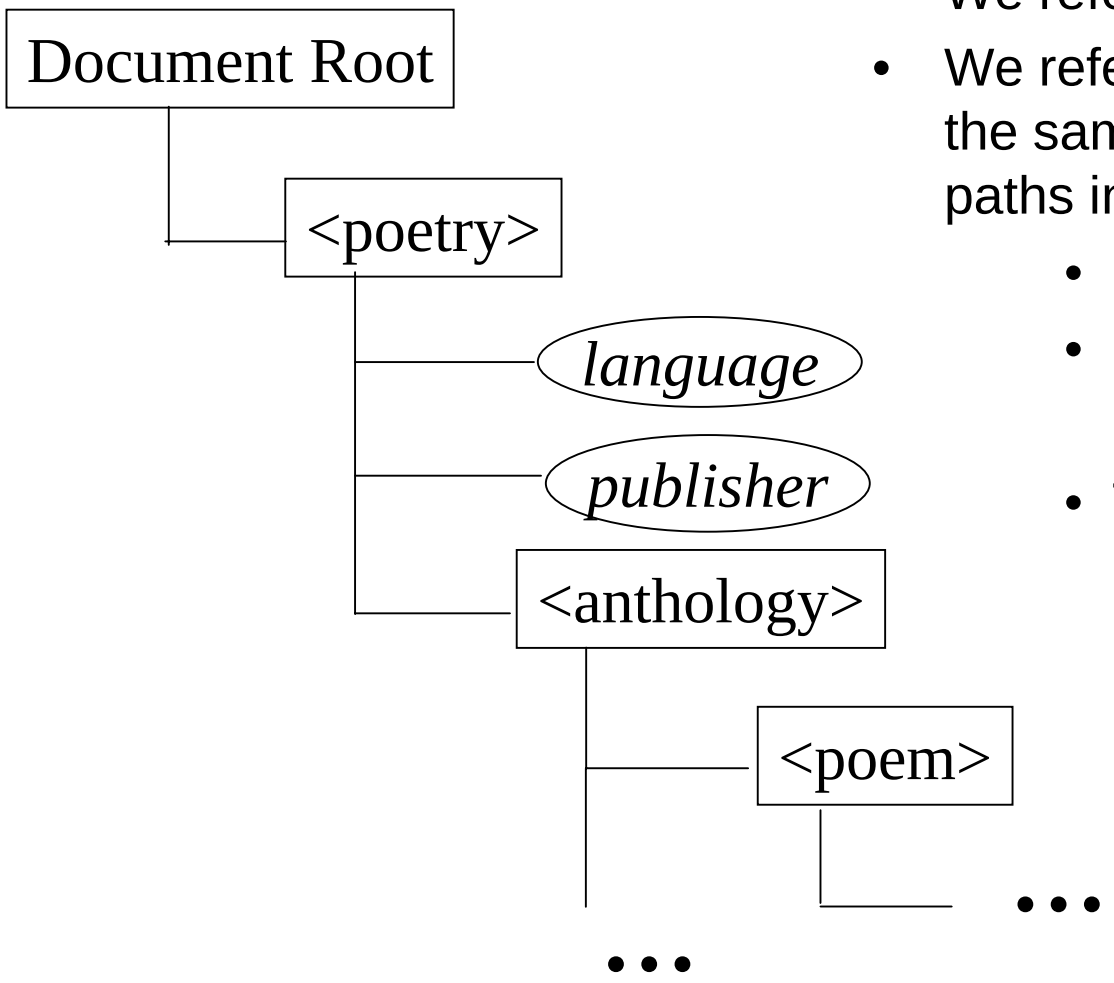
Nodes in XPath

- XPath to refer to anything in the source tree as a *node*
    - Elements
    - Attributes
    - Processing Instructions
    - etc

Example XML Document

```
<?xml version="1.0"?>
<!DOCTYPE poetry [
     <!ELEMENT poetry (anthology+)>
     <!ATTLIST poetry
          language  CDATA  "English"
          publisher (Oxford|Elsevier|PrenticeHall) "Oxford">
     <!ELEMENT anthology (poem*)>
     <!ELEMENT poem (title?, author?, stanza+)>
     <!ELEMENT title (#PCDATA) >
     <!ELEMENT author (#PCDATA) >
     <!ELEMENT stanza (line+) >
     <!ELEMENT line (#PCDATA) >
]>
<poetry language="English" publisher="Elsevier">
  <anthology>
     <poem>
          <title>Song of the Open Road</title>
          <author>Ogden Nash</author>
          <stanza>
               <line>I think that I shall never see</line>
               <line>A billboard lovely as a tree.</line>
               <line>Indeed, unless the billboards fall</line>
               <line>I'll never see a tree at all.</line>
          </stanza>
     </poem>
  </anthology>
</poetry>
```

Nodes in our example document



- We refer to the document root as “/”
- We refer to the rest of the tree using the same notation as a directory paths in a UNIX file system. Eg
  - “/poetry”
  - “/poetry/anthology/poem”
- These are called location paths in XSLT

Context Node

- At any single time, XPath will consider one of the nodes to be the context node (ie. “the node I am in now”).

- XPath expressions will be evaluated based on the current context node.

    - Let’s look more closely at an XSLT style sheet to see what this means.

The XSLT style sheet again

- So the basic structure of the XSLT style sheet is normally:

```
<?xml version=“1.0”?>
<xsl:stylesheet ...>

  <xsl:template match=XPath expression >
    ... Something to do ...
  </xsl:template>

  <xsl:template match=XPath expression >
    ... Something to do ...
  </xsl:template>

  ...

</xsl:stylesheet>
```

Evaluating the Templates

- What XSLT will do is set the current context node to the document root, and start finding the template that matches the current node.
    - It will then produce the result tree according to what is specified in the body of the template.
- If more than one template matches the current node, the last one (counting from top to bottom) will be evaluated.

Evaluating the Templates

- If you want the templates to be applied to the nodes below the document root, you must specify the element `<xsl:apply-templates …>` within the body of the template which matches the document root.

Evaluating the Templates

- Eg. The following will not work:

```
<xsl:template match="/">
  <!-- Do nothing -->
</xsl:template>

<xsl:template match="line">
  The line is: <xsl:value-of select="."/>
</xsl:template>
```

This template is never matched!

- But this will:

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

<xsl:template match="line">
  The line is: <xsl:value-of select="."/>
</xsl:template>
```

Match the document root or any other node - so this recursively applies ALL templates to every node.

Creating the Result Tree

- We have seen the basis of traversing through the source tree. Now lets look at getting interesting information out of the source tree to put in the result tree.

- The most commonly used element for doing that is `<xsl:value-of>`

<xsl:value-of>

- The common format for the element is:

```
<xsl:value-of select="XPath Expression">
```

  - This puts into the result tree the value as specified by the XPath expression.
  - The expression can contain a lot more than just a location path.

Functions in XPath

- To get interesting information out of the source tree to use in elements like <xsl:value-of>, we use the set of functions available in XPath.

- Eg.

name()	The name of the node.
text()	The #PCDATA of the current node.
sum()	The sum of the numbers given in the #PCDATA of specified nodes.
concat()	Concatenate two strings.

Functions in XPath

- Egs.

```
<xsl:template match="*|/">
  The node is <xsl:value-of select="name()"/>
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="/">
  The sum of all numbers in this document is
  <xsl:value-of select="sum(/spreadsheet/numbers)"/>
</xsl:template>
```

Functions in XPath

- XPath functions are also useful for specifying location paths
- Eg. Finding different lines

```
<xsl:template match="/poetry/anthology/poem/stanza/line(position()=1)">
```

```
<xsl:template match="/poetry/anthology/poem/stanza/line[1]">
```

```
<xsl:template
match="/poetry/anthology/poem/stanza/line(position()=last)">
```

Conditional Processing

- When we are traversing the source tree, we can also do processing based on conditions. We do so using the elements such as `<xsl:if>` and `<xsl:choose>`.

Conditional Processing

- The basic syntax:

```
<xsl:if test="Boolean expression"> ... </xsl:if>


<xsl:choose>
  <xsl:when test="Boolean expression"> ... </xsl:when>
  <xsl:when test="Boolean expression"> ... </xsl:when>
  ...
</xsl:choose>
```

Loops

- You can also do loops (iterations) using the element `<xsl:for-each>`.

- The basic syntax:

```
<xsl:for-each select="XPath expression">
```

Loops

- Eg.

```
<xsl:template match="stanza">
    <xsl:for-each select="line">
       A line here!
    </xsl:for-each>
</xsl:template>
```

Default Templates

- By default, the following templates will already exist:

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

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

  - So even if you have no templates in your style sheet, the #PCDATA in every node your document will be put in the result tree. If a node doesn't have #PCDATA, a new-line character will be put.
  - Also, if you didn't define any templates to override the default behaviours, they will be invoked.

Read the Unit Reader

- I have covered only a small percentage of what is given in section 9 and 10 of your unit reader
    - Just enough to give you a glimpse of what XSLT is.
- To do more interesting and sophisticated transformations, read the whole of the two sections.
- If you get beyond even that, read the original specifications from referenced in this lecture.