XSLT Language Overview and Processing Model

Prof. Jim Whitehead
CMPS 183, Spring 2006
May 22, 2006
XSLT Processing Model

• In XSLT processing, the source XML document is processed using a preorder list traversal
  – each XML element is examined before its children
  – until the first matching template
• Once the first template match has occurred, the sequence of XSLT statements inside that template is executed once for every node matching the template.
• Any output text/XML within the template is added to the output stream.
• Further processing is now controlled by additional apply-templates directives.
Processing Model Specification

• XSLT Processing model is described in Section 5.1, “Processing Model”, of the XSLT specification.

A list of source nodes is processed to create a result tree fragment. The result tree is constructed by processing a list containing just the root node. A list of source nodes is processed by appending the result tree structure created by processing each of the members of the list in order. A node is processed by finding all the template rules with patterns that match the node, and choosing the best amongst them; the chosen rule's template is then instantiated with the node as the current node and with the list of source nodes as the current node list. A template typically contains instructions that select an additional list of source nodes for processing. The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing.

• Oy, that’s dense. Let’s walk through each sentence.
• **A list of source nodes is processed to create a result tree fragment.**
  – This states that XSLT performs actions on one or more nodes (typically XML elements, but could be other kinds of nodes, such as the root node) in the source XML document.
  – The result of these actions is the creation of a (partial) result tree.

• **The result tree is constructed by processing a list containing just the root node.**
  – Since an XML document tree contains many nodes, the processing model needs to state which of these nodes is processed first.
  – This line states that the first node to be processed is the root node (that is, the initial list of source nodes contains only one node, the root node).
XSLT Processing (2)

• A list of source nodes is processed by appending the result tree structure created by processing each of the members of the list in order.
  – For cases where the current node list contains multiple nodes (this can happen when you have an XPath expression that evaluates to multiple nodes), this statement says that the list of nodes are evaluated in order.
  – This order is, by default, the order the nodes appear in the document, though this can be changed by using the sorting features of XSLT.
  – When each node in the list is processed (e.g., by the execution of statements in a template), it creates a result tree fragment that is appended to the current result tree.
  – That is, the output document is created by appending the results of the execution of templates, and hence the order of execution of templates controls the order of items in the result tree.
A node is processed by finding all the template rules with patterns that match the node, and choosing the best amongst them; the chosen rule's template is then instantiated with the node as the current node and with the list of source nodes as the current node list.

- Since the section hasn’t yet explicitly stated how processing relates to templates and patterns, this sentence spells it out. Processing is defined as executing template rules (i.e., executing the XSLT statements within a template).

- The sentence also points out that there can be multiple templates that might match a given node, hence the “best amongst them” phrase. Section 5.5, Conflict Resolution for Template Rules, describes how to resolve situations where multiple templates match a given node. In general, the most specific pattern wins (this is similar to CSS, where the same general rule applies).

- The second part of the sentence explicitly states how the current node is assigned. This is important for XPath processing, since all relative XPaths are computed relative to the current node. The list of source nodes for XSLT processing is defined to be the same as the current node list from evaluating an XPath expression, and hence this explicitly ties together XSLT and XPath.
XSLT Processing (4)

- A template typically contains instructions that select an additional list of source nodes for processing.
  - This describes how processing progresses in XSLT. In the sentence, “the result tree is constructed by processing a list containing just the root node,” we learn that processing begins with just the root node.
  - We now learn that each matching template causes processing to progress by selecting an additional source node, or list of source nodes, for processing. This selection of source nodes is performed by using one or more xsl:apply-template statements.
Since we now know that processing starts at the root node, and is progresses by templates that match, there is one riddle still to be explained.

From looking at examples, we know that there are XSLT stylesheets where the first template to be executed does not explicitly match the root node; instead it matches a node that is a child of the root, possibly many layers down.

How, then, do we get from the root node (which we are guaranteed to process according to the XSLT specification), and the interior node that is our first matching template?

The answer is a built-in rule that automatically progresses the processing, essentially doing nothing, until it reaches a user-defined matching template. This built-in rule is:

```xml
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```
XSLT Built-In Rule (2)

• The built-in rule has the effect of matching all element nodes.
• The internal xsl:apply-templates statement doesn’t explicitly describe an XPath, or a template to match.
  – The default behavior of xsl:apply-templates in this situation is to cause the processing of all children of the current node.
• Hence, this built-in rule causes a preorder traversal of the source document tree.
• It is this rule that bridges the gap between the root node, and the first user-specified template.
• However, once the first user-defined template is reached, it has higher precedence than the built-in rule, and will be preferentially applied.
  – At this point, the user-specified processing takes over.
XSLT Processing (5)

- The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing.
  - This last statement states that the sequence of matching nodes, executing the instructions associated with the template that best matches each node, and then selecting additional nodes to process, is continued recursively until all of the possible matches have been found in the source document.
  - That is, you’ve reached the end of the source document.
Result Sorting

• The xsl:sort statement can be used to change the order of processing nodes in a node list from document order, to alphabetical order.
  – xsl:sort is a child element of apply-templates
  – The matching node list is found using the XPath expression in apply-templates
  – The resulting node list is sorted according to the sort specification given in the xsl:sort statement.
XSLT Sorting Example

• From XSLT Specification.
• Suppose an employee database has the form:
  <employees>
    <employee>
      <name>
        <given>James</given>
        <family>Clark</family>
      </name>
    </employee>
    ...
  </employees>
XSLT Sorting Example (2)

<xsl:template match="employees">
  <ul>
    <xsl:apply-templates select="employee">
      <xsl:sort select="name/family"/>
      <xsl:sort select="name/given"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="employee">
  <li>
    <xsl:value-of select="name/given"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="name/family"/>
  </li>
</xsl:template>
Setting Attribute Values

- **Problem: Putting dynamic content into the value of attributes**
- In the examples we have looked at so far, the XSLT stylesheets have grabbed content from attributes and elements, and then placed them as the content of elements. There haven’t been any attributes set so far.
- Consider if you wanted to make a link to a book in Amazon.com’s website. You need to tack the ISBN number onto the URL of the book in the `href` attribute of the `a` element, such as:
- The straightforward approach won’t work:
  ```xml
  <!-- won’t work: -->
    Literary Machines
  </a>
  ```
- XML doesn’t allow the “<” character inside of an attribute value.
Approach 1: xsl:attribute

• **xsl:attribute**
• Adds attributes to elements in the output stream of an XSLT document.
  
  ```xml
  <a>
    <xsl:attribute name="href">
      <xsl:value-of select="@isbn"/>
    </xsl:attribute>
  </a>
  
  • Generates:
  
  ```xml
  <a href="http://www.amazon.com/exec/obidos/ASIN/0893470562">
  
  • … or whatever was in the value of the isbn attribute of the source.
Approach 2: Attribute Value Templates

• A simpler syntax for setting attribute values:
  
  `<a href="
  http://www.amazon.com/exec/obidos/ASIN/{$@isbn}"">`

• The "{“ and “}” instruct the XSLT processor to dynamically fill-in the contents (essentially similar to a value-of instruction)

• According to the XSLT specification, the actual behavior is:
  
  – The attribute value template is instantiated by replacing the expression together with surrounding curly braces by the result of evaluating the expression and converting the resulting object to a string as if by a call to the `string` function.
XSLT Attribute Sets

• Allows the definition of a set of attributes that can be reused.
• xsl:attribute set creates a named set of attributes – this set of attributes can be referenced in other locations in the stylesheet.

```xml
<xsl:attribute-set name="body-style">
  <xsl:attribute name="bgcolor">white</xsl:attribute>
  <xsl:attribute name="text">green</xsl:attribute>
</xsl:attribute-set>
```

• This is a “top level element” indicating it is a direct child of xsl:stylesheet.
Creating Elements: xsl:element

- Output elements are often directly represented in the input, and then passed through to the output, with the value being filled-in with “xsl:value-of” directives. xsl:element allows elements with computed names to be put into the output.

```xml
<xsl:element name="{qname}" namespace="{uri-reference}" use-attribute-sets="qnames"> <!-- Content: template --> </xsl:element>
<xsl:element name="body" use-attribute-set="body-style">
  <h1>Title text</h1>
  <p>some paragraph text…</p>
</xsl:element>
```

- Outputs:

```xml
<body bgcolor="white" text="green">
  <h1> Title text</h1>
  <p>some paragraph text…</p>
</body>
```
Conditional Evaluation xsl:if

• Consider if you wanted to output a name that is concatenated from the contents of first, middle, and last elements. The algorithm would be something like:
  – output value-of(first) {space}
  – output value-of(middle) {space}
  – output value-of(last)
• If the person had no middle name, this would result in two spaces in the output. Want to check to see if the middle name element exists first.
• Can do this with:
  <xsl:if test="middle">
      … conditional output
  </xsl:if>
• Can also check for existence of an attribute:
  <xsl:if test="@arbitrary-attribute">
      … conditional output
  </xsl:if>
• The value inside the test clause can be an arbitrary XPath expression. There are a number of XPath functions (not covered in class) that can be useful here.
• No else clause for xsl:if
XSLT Variables

- Variables can be set as top-level elements, in which case they have global scope.
- Otherwise, can set variables inside a template. Such variables are visible to elements that follow the declaration, and to their descendants.
- Ways of setting variables:
  - `<xsl:variable name="foo">value of variable foo</xsl:variable>`
  - `<xsl:variable name="bar" select="/arbitrary/XPath/expression"/>`
- Using variables:
  - `<xsl:value-of select="$foo"/>`
- Can also use within Attribute Value Templates (AVTs):
  - `<a href="${link_location}">this is a link</a>`
- The braces inform the XSLT processor to use the value of the variable – otherwise, will include “$link_location” as a string value.