XSLT Processing Model
CS 183 – Hypermedia and the Web

The XSLT processing model is described in Section 5.1, Processing Model, of the XSLT specification. This section states:

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.

Let’s analyze this sentence by 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).

_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.

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>
```

This 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, and 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.

*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).

In a nutshell, nodes are processed in a preorder traversal until the first user-specified
template match occurs. At this point, processing is directed by the user-specified
templates, and apply-template statements.

A brief note on 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, and then
the resulting node list is sorted according to the sort specification given in the xsl:sort
statement:

An example from the XSLT specification:

*For example, suppose an employee database has the form*

```xml
<employees>
  <employee>
    <name>
      <given>James</given>
      <family>Clark</family>
    </name>
  </employee>
  ...
</employees>
```

*Then a list of employees sorted by name could be generated using:*

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