The XSLT Processing Model

Julia Flanders and Syd Bauman
So let’s come back now and consider in more detail how an XSLT stylesheet runs and what it does. What we’re talking about here is something called the XSLT processing model and it is essentially the set of rules that direct how the stylesheet will run and what it will do, in what order. These rules are actually fairly simple once you’re familiar with them, although they’re not perfectly intuitive at the outset—so we’re going to step through them in detail with an actual example and follow how they work.
Essentially we are starting with the root of the input document, and working our way through that tree, based on the templates we find in the stylesheet.

The XSLT processing model: matching the root

So what's the first rule? we remember this from our earlier, more informal look:
• start with the root of the input document (what is it?)
The XSLT Processing Model

• and then ask: is there a template that matches this element I'm considering? (is there?)

The XSLT processing model: applying a template

The Rules
0. Start with the root of the Input Document
1. Consider the input element you’re processing
2. Is there a template that matches that element?
3. If YES, apply that template...
   • if there are output elements, put them in the output
   • if there is literal text, write it into the output
4. If NO, apply built-in processing rules...
   • if what we’re processing is text, split out the text
   • if what we’re processing are child elements, process those children

The Output Document

• in this case, what is it doing? (writing out the first few layers of the output tree, and including a little bit of literal text)

<TEI>
  <text>
    <div type="chapter">
      <head>Chapter 1: The Beginning</head>
      <p>It began.</p>
      <p>That's what</p>
      <em>noveis</em></div>
  </text>
</TEI>

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:ms="http://www.tei-c.org/ns/1.0" xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="TEI">
    <html>
      <head>
        <title>Test Document</title>
      </head>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
</xsl:stylesheet>

<html>
  <head>
    <title>Test Document</title>
  </head>

  <body>
    <h1>Test Document</h1>
    <p>The printer had nothing to say.</p>
    <p>The End: A Short Novel</p>
  </body>
</html>
The XSLT processing model: processing children

**The Rules**

0. Start with the root of the Input Document
1. Consider the input **element** you’re processing
2. Is there a template that matches that element?
3. If YES, apply that template...
   ...if there are output elements, put them in the output
   ...if there is literal text, write it into the output
   ...if there are instructions to apply templates, process the children of the matched element
4. If NO, apply built-in processing rules...
   ...if what we’re processing is text, split out the text
   ...if what we’re processing are child elements, process those children

---

**The Output Document**

```
<html>
  <head><title>Test Document</title></head>
  <xsl:apply-templates/>
</html>
```

---

So what did we just do? We applied a template, which entails:
- putting out the output elements
- writing out any literal text
- and one thing further: if there are instructions to apply templates, then process the children of the matched element

So what is the element we just matched? (**TEI**)
And it does include instructions to apply templates, which means that we... (process the children of the matched element, i.e. the children of <TEI>)

So what are these children? <text>: what do we do with this? What rule applies in this case? (no template matches it: so we apply built-in processing rules, which say that we... (spit out any text, and process any children)

So what are the children of <text>?...<front>: what do we do with this? What rule applies in this case?

There's a template that matches <front>, but what does it tell us to do? What rule applies in this case?... there are no instructions to apply templates, so this part of the process stops there. There's no output from <front>.

Does it stop altogether? Or are there other loose ends that will keep the process going?
The XSLT processing model: chugging along

The Rules

1. Consider the input element you're processing
2. Is there a template that matches that element?
3. If YES, apply that template...
   ...if there are output elements, put them in the output
   ...if there is literal text, write it into the output
   ...if there are instructions to apply templates, process the children of the matched element
4. If NO, apply built-in processing rules...
   ...if what we're processing is text, spit out the text
   ...if what we're processing are child elements, process those children

There's another child of `<text>`, namely `<body>`, so our built-in stylesheet rule of "process the children" applies here and allows us to proceed to the `<body>` element.

So what happens here? (another output element is generated, and inside it, additional templates will be applied)
Next we start processing the children of `<body>`, and we have two templates here that do somewhat similar things; what are we matching here?

What if we had wanted to just match any `<head>` in the input document?

Why do it this way? Why distinguish between two different locations for `<head>`
The XSLT processing model: a final round

The XSLT Processing Model

Finally we're getting to the last of the children in the input document... What is happening here? What rules apply when we get to `<emph>`? (There's no template that matches, so we apply the built-in rules, which say...if what we're processing is text, spit out the text)
Now we can back up again and study the whole thing as a finished product: the input, the stylesheet, and the output. Any questions?
You’ve probably already noticed that we’re dealing here with three languages:

• The language of the XSLT stylesheet itself (which is a language containing elements like `<template>` and `<apply-templates>`)  
• The language of the input document (in this case, TEI)  
• The language of the output document (in this case, HTML)  

Within the stylesheet itself, we need to keep these three different languages distinct from one another, so that the processor always knows what piece of what tree it is dealing with. We do this with something called namespaces. (Does everyone understand namespaces? Quick review on the next slide if necessary...)  

Each of these languages plays a specific role in the stylesheet ecology and gets referenced in a distinctive way:

• Let’s take the simplest first: the output tree, which is being treated transparently in our examples: it doesn't use a namespace prefix, and this is because we have declared that the entire stylesheet is in the HTML namespace (we did this with the namespace declaration attribute-like thingy: xmlns)  
• The next fairly simple case is the input tree, which also looks as if it's not getting a namespace. How are we keeping this separate from the output tree? The trick here is that the input tree is always accessed via these @match and @select (and similar) attributes. These attributes all access the input tree via XPath, and up at the top, we provided a default namespace for all XPaths (via xpath-default-namespace)
The XSLT Processing Model

• And finally, the stylesheet document has its namespace specified as the XSL namespace (the default namespace for the stylesheet is already set to HTML) so all of the stylesheet elements have a namespace prefix.

Namespaces review

Without the genus, we don't know what animal these species are:
• glauca: a pine tree (Picea glauca) or a small yellow flower (Agoceris glauca)?
• leucocephalus: a cactus (Pilosocereus leucocephalus) or a bald eagle (Haliaeetus leucocephalus)?

Without knowing the language, we don't know these words mean:
• the (English definite article or a French hot drink?)
• bad (English adjective or German noun for "bath"?)

Without a namespace designation, we don't know what these elements mean:
• <p> (TEI paragraph or HTML block element?)
• <div> (TEI textual division or HTML grouping element?)
• <fileDesc> (TEI or EAD?)

With the namespace, all is clear:

• <tei:p>

• <html:div>

• <ead:fileDesc>

The namespace prefix is somewhat like a genus or language name: it tells us more precisely what language we are speaking (and hence what the semantics of the element are)