

XPath and XSLT Introduction

Prof. Jim Whitehead

CMPS 183, Spring 2006

May 19, 2006

XSLT Overview

- XSLT – Extensible Stylesheet Language for Transformations
- XSLT is a language for converting XML into other forms
 - Can convert XML into HTML, XML, SVG, VRML, programming language source code, text files, etc.
 - Helps answer the question: how do I display XML?
 - Can use CSS with XML
 - Can also use XSLT to convert XML to HTML
- XSLT is a *graph transform* language

XSLT as Compared to CSS

- CSS is used to change display characteristics of primarily HTML documents (modern browsers can also use CSS with XML):
- But, CSS has limitations:
 - CSS can't change the order of elements in a document
 - CSS cannot perform computations (summing together multiple element values, for example)
 - CSS cannot combine multiple documents
- Related standard: XSL-FO – XML Stylesheet Language, Formatting Objects
 - a language for converting XML into page descriptions, such as PDF documents.
 - Similar goals as Latex, but very different syntax/semantics.

XSLT Observations

- XSLT stylesheets are XML documents
 - XSLT uses XML to represent the XSLT language itself
- XSLT uses rule-based pattern matching.
- XPath is used in the creation of the patterns.
 - Typical form: if you see part of the document that looks like pattern, perform action.
 - XPath is used to identify portions of an XML document
 - Addresses the problem of how to precisely, and uniformly, describe a particular element, set of elements, or attribute in an XML document.

Uses for XSLT

- Ways you might want to use XSLT:
 - Convert XML into multiple display representations (for Web browser, handheld, cell phone, print)
 - Convert XML data retrieved from the Web (e.g., RSS feeds) into HTML for display in a browser
 - Data conversion for data interchange among different formats
 - Have Web site content available in logical form, and have the actual Web site generated from the XML. More easily supports site redesign.
 - Have documents in a relatively neutral format that is richer than text.
 - Automatically generate source code. Create a model of the source code in XML (for example the XML representation of UML models), then transform to code using XSLT.

XSLT Hello World

- Hello World example (From XSLT, Doug Tidwell, O'Reilly, 2001)
- Input XML:

```
<?xml version="1.0"?>  
  <greeting>  
    Hello, World!  
  </greeting>
```

- Output HTML:

```
<html>  
  <body>  
    <h1>  
      Hello, World!  
    </h1>  
  </body>  
</html>
```

XSLT Hello World (2)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
  </xsl:template>
  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XSLT Hello World – Steps Performed

- Steps performed during the example:
 - XSLT stylesheet is read and parsed
 - XML source document is read and parsed into a tree structure
 - XSLT processor is at the root of the source document (context set to root)
 - Since there is a template that matches the document root, evaluate that template.
 - The template's instruction is to evaluate any templates that apply to greeting elements
 - There is one greeting element at the document root, so evaluate the second template instruction
 - Elements that are not in the XSL namespace are passed through to the output (these are the HTML tags html, body, h1)
 - The `xsl:value-of` puts into the output the value of the node matching the XPath expression, which in this case is the current context (i.e., the greeting node)
 - Remaining HTML tags pass through to output
 - No other templates match, are done.

XSLT Hello World - Notes

- `xsl:stylesheet` element provides `xsl` namespace declaration, and version of XSLT.
 - XSLT processors complain if these are missing.
- `xsl:output` describes which output format is being used
 - in this case HTML, but many other are possible – “xml” is a common one
- Built-in template rule:

```
<xsl:template match="*/">  
  <xsl:apply-templates/>  
</xsl:template>
```
- Ensures that XML documents will be processed, even if your stylesheet only includes more specific matching rules, and omits a matching rule for the root element.

XPath Overview

- XML documents have a tree structure
- XPath allows you to concisely describe sets of XML elements and attributes in this tree.
- Analogy to file system paths.
 - A file system hierarchy is a tree of directories and files.
 - A filesystem path describes how to reach a specific file, starting either from the root, or the current working directory.
- XPath exploits this intuitive notion of a path, and extends it to XML documents.

XPath Data Model

- In the XML tree available for XPath expressions, there are 7 node types:
 - The root node (one per document)
 - Element nodes
 - Attribute nodes
 - Text nodes
 - Comment nodes
 - Processing instruction nodes
 - Namespace nodes
- XPath expressions are evaluated against XML after it has been read by an XML processor
 - Hence character entity references and CDATA sections have already been parsed, and hence are unavailable to the XPath processor.

XPath Data Model (2)

- Root node contains only one element node, and possibly many PI nodes and/or comment nodes.
- Element nodes correspond to element tags (but not to the enclosed content)
- Attribute nodes correspond to attributes defined on elements
- Text nodes are the textual content of a node, and its children
- Although an element node is the parent of its attribute nodes, those attribute nodes are not children of their parent. The children of an element are the text, element, comment, and processing instruction nodes contained in the original element. If you want a document's attributes, you must ask for them specifically.

Context

- Context is similar to the notion of a current working directory in filesystems.
 - Path expressions are evaluated with respect to the context.
- There are two forms of path expression: abbreviated, and unabbreviated

- Consider the following XML document:

```
<A>
  <B>
    <C/>
  </B>
  <B>
    <D>
      <C id="123"/>
      <C id="456"/>
    </D>
  </B>
</A>
```

- The following are abbreviated path expressions (all assume the root is the context)
 - /AB/C – the C element that is a child of a B element that is a child of an A element
 - //C – all three C elements
 - //B – both B elements
 - //D/C – the two C elements that are a child of D
 - /D/C – the empty set (no D element under root)
 - D/E – the empty set (no matching path expressions)

Unabbreviated Path Expressions

- **Unabbreviated path expressions:**
 - /child::A/child::B/child::C – same as /A/B/C
 - //child::C – same as //C
 - //child::D/child::C – same as //D/C
- **Accesses surrounding a context node are divided into axes:**
 - Parents: “parent::” or “..” selects the parent of the context node
 - Attributes: attributes of the current node
 - “attribute::type” or “@type” are equivalent
 - Self: “self::” or “.”
 - preceding-sibling – all nodes that precede the start of the context node, and have the same parent node
 - following-sibling – all nodes that follow the end of the context node, and have the same parent node if any

Predicates

- **Predicates:**

- Can additionally constrain the number of nodes returned.
- number – select only a given element number, as in /D/C[2] – the second C that is a child of D.
- attributes – select only elements with a given attribute, as in C[@id="123"] – the C element that has the id “123”.

