Distributed Software Development

XSLT

Chris Brooks

Department of Computer Science

University of San Francisco
XSLT is an XML-based language that allows you to declaratively specify how a document should be changed or transformed.

- You specify the output for a particular element; no need to manage tree traversal.

Useful for:
- Emitting an HTML display of an XML document
- Converting between tag vocabularies
- Extracting plain text from an XML document
- Automatically modifying or filtering an XML document.
You can transform an XML document into:
- Plain text
- HTML
- XML (or any flavor thereof)
<catalog>
    <song>
        <title>Tomorrow Never Knows</title>
        <artist>Beatles</artist>
        <album>Revolver</album>
        <genre>Rock</genre>
        <rating>5</rating>
        <length>2:57</length>
        <date>
            <day>6</day>
            <month>Feb</month>
            <year>2005</year>
        </date>
    </song>
    ...
</catalog>
To begin, let’s use XSLT to print a plaintext version of our catalog.

We can run XSLT from the command line or within a browser.

- /usr/bin/4xslt on nexus
- Most modern browsers have XSLT support
- Debugging is easier from the command line
That’s fine, but pretty dull.

Let’s select just artist, title, and album to display.

We do that through the use of apply-templates.

What if we left out apply-templates in the song template?
We can also emit other markup languages, such as HTML. (XHTML, actually).

Just indicate the tags to be produced by a template match.
We can also use XSLT to create new XML documents with different tag names or contents.

For example, let’s say we want to change the tags to be in Spanish.
When transforming from XML to XML, often, it’s useful to copy sections of a document without changing it.

copy makes a shallow copy of a node.
  - Useful if you want to change a bunch of values or attributes.

copy-of makes a deep copy and lets you specify a path.

For example, let’s make a new database with just artist, album and title.
We can still use CSS to control presentational elements.

With HTML, we can just embed a ‘link’ tag in the generated HTML.
6 If we’re emitting XML, we can instead embed a `processing instruction` into the output document.

6 Note: this will work best if we do the XSLT on the server side.

```
<xsl:processing-instruction name="xml-stylesheet">
  href="songs.css" type="text/css"
</xsl:processing-instruction>
```
The command line is great for debugging, but much of the time, we want the client to do the work.

Most web browsers have at least some support for XSLT.
- More advanced features are not universally supported.
- In particular, the browser’s XSLT processor may make a single pass and not apply the CSS. (firefox)
The examples we’ve seen so far match templates to elements based solely on the element’s tag name.

Often, you want something more flexible:
- Match the root element
- Match all text nodes
- Match all children of an author node

Essentially, we want to specify matching rules based on an element’s position in the DOM tree.

XPath is a language for doing this.
In XPath, everything is dealt with as a path from the root of the tree.

To find a node, we’ll use a location path, which consists of a series of location steps.

A location step consists of:

- An axis that tells us which direction to travel
- A node test that specifies which types of nodes apply
- Predicates that use boolean tests to help filter nodes.
Axes consist of:

- Children and parents, which have their usual meanings.
- Ancestor, which means any node above the node of interest.
- Descendant: any node below the node of interest.
- Following: following siblings and their descendants.
- Preceding: preceding siblings and their descendants.
- Self
The second component of the location step is the node test.

This is joined to the axis by a ::

Some tests:
- / - root node
- * - any element
- author - any node named “author”
- text() - any text node

In our Tolkien example, we might use book/volumes/volume::”The Two Towers”
7-15: Shortcuts

- // - descending from the root. //volume matches all volume nodes below the root.
- ../* - all siblings
- .. - parent
- /* - document element
- @name - matches attribute named 'name'
7-16: Examples

/songlist/child::node( ) - matches all song elements, plus the comment.

//comment()/following-sibling::*/title - matches 'Tomorrow Never Knows'.

/*/* - matches all song elements

id('s1')/.. - matches the songlist element

id('s1')/ancestor-or-self::* - matches the songlist element and the song element for 'Tomorrow Never Knows'

id('s1')/genre::country - matches nothing. (lets you test node type).
If you need more flexibility in specifying nodes of interest, you can use a predicate.

Predicates are contained inside square brackets.

To be included in final node set, a node must pass both axis and predicate tests.
//song/[id="s1"]/title/text - text for all 's1' songs.
//song[title] - all quotations that have a source subelement.
//song[not(source)] - songs that do not have a title sub-element.
7-19: So what’s all this good for?

6 XPath is very useful for allowing users to query an XML document.
6 Even more useful for specifying which transformations should be applied in an XML document.
6 gives us a way to easily specify transformations that should take place based on a node’s context.
XSLT also has built-in support for sorting and processing your elements.
You can also pass parameters into an XSLT stylesheet.

You can also define them at the top of your XSLT program.

Parameters can be referenced with a $.

```xml
<xsl:param name="discount" select="0.10"/>
...
<discount><xsl:value-of select="$discount"/></discount>
  <discountPrice>
    <xsl:value-of select="price - (price * $discount)"/>
  </discountPrice>
```
The DOM API makes it easy to create new Nodes for an existing document.