Distributed Software Development

XSLT

Chris Brooks

Department of Computer Science

University of San Francisco

9-3: XSLT

- 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.

9-3: Output

- You can transform an XML document into:
  - Plain text
  - HTML
  - XML (or any flavor thereof)

9-4: Our CD database

9-7: Emitting HTML

- We can also emit other markup languages, such as HTML. (XHTML, actually).
- Just indicate the tags to be produced by a template match.

9-8: Emitting XML

- 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.

9-9: Copying Nodes

- 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.

9-10: Incorporating CSS

- We can still use CSS to control presentational elements.
- With HTML, we can just embed a 'link' tag in the generated HTML.

9-11: Incorporating CSS

- If we're emitting XML, we can instead embed a *processing instruction* into the output document.
- Note: this will work best if we do the XSLT on the server side.

9-13: Xpath

- 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.

9-14: XPath

- 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.

9-15: Axes

- 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

9-16: Node test

- 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 books/volumes/volume::"The Two Towers"

9-17: Shortcuts

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

9-18: Examples

- /songlist/child::node( ) - matches all song elements, plus the comment.
- //comment()/following-sibling::*/title - matches 'Tomorrow Never Knows'.
- /*/* - matches all song elements
- id('x1')/.. - matches the songlist element
- id('x1')/ancestor-or-self::* - matches the songlist element and the song element for 'Tomorrow Never Knows'
- id('x1')/genre::country - matches nothing. (tells you text node type).