3. Styling and Transformation; XPath and XSLT [1]

XML Foundations (IS 290A-1) - 6 September 2005

Copyright © 2005 Robert J. Glushko

Plan for Today's Class

- Content {and,vs} Presentation
- The Function and Architecture of XML Transformation
- XPath

Content and Presentation

- Content is the semantically meaningful information contained in a document and we can often give names to different parts or components of the content that have no inherent formatting or presentation:
  - Name, Address, Summary, Ingredient, Price, Quantity, Title, ...

- When documents are printed or displayed, they also contain presentation information such as page numbers, type fonts and sizes, indentation, column organization, color, underlining, and so on.

- Content and presentation can be bound together or confounded, often implicitly

Content vs. Presentation

- But separating content and presentation is inherent in XML since it is used to generate new tag sets rather than being a fixed tag set like HTML

- Since a browser can't know what tags to expect in an XML instance, no presentation semantics are pre-defined

- This feature of XML makes a requirement out of what is a good habit to practice in any case

Why Transform? [1]

- You have smart information, but it is too smart for the web or end users to handle

- You need to Re-purpose information – extracting and / or formatting the same piece of information in many different ways, producing a different document type targeted for a different user or purpose

- You have to support a variety of output devices that have different capabilities— often called Re-packaging

Why Transform? [2]
• You need to conform to a structural or formatting standard that is different from your company or organization's information model

• Your "web service" needs to convert an "inbound" non-XML document to XML, or convert XML to a non-XML format for the "outbound" document

---

Transforming is "Manipulating Models"

• All of these reasons for transforming can be summarized as "Manipulating Models"

• Some transformations involve turning one XML content model into another XML content model

• Other transforms are from XML to a content model expressed in a non-XML syntax

• Other transforms involve turning an XML content model into a presentation model
  
  o This often means transforming XML to HTML or XHTML
  
  o Best practice is to create simple, device-independent XHTML and then apply Cascading Style Sheets to create a device- or user-tailored presentation

• Sometimes you need a transformation "pipeline" - XML -> XML -> XHTML

---

"Dressing a Pig"

Don't "Dress a Pig"

• Separating transformation from styling lets people skilled with XML do what they do best, and lets people...
skilled in presentation design do what they do best

- The navigation and presentation design of a user interface can and ideally should reflect the information architecture embodied in the XML.

- But good presentation design can't make up for poor information architecture; if the information being displayed or elicited isn't the "right" information, the user interface designer is just "dressing a pig".

- This is why model-based approaches to generating user interfaces from XML schemas are promising - the topic of the last lecture in this class.

---

**Conceptual Example of Re-purposing**

- Multiple versions or models of the same product
- Multiple languages
- Similar information among different output media
- Frequent multiple releases of products on short time schedules

---

**Technical Example of Re-purposing – Product Database / Catalog**
- The XML instance: product-db.xml
- The target HTML instance viewed as an internal database: product-db.htm
- The target HTML instance viewed as a catalog for external customers: product-catalog.htm
- The XSLT transformation for the database: product-db.xsl
- The XSLT transformation for the catalog: product_catalog.xsl

**The Architecture of XML Transformation**

- A particular transformation may apply to more than one document – it might be used to enforce standards for all instances of a document type
  
  ```xml
  <?xml-transform type="text/xsl" href="standard_style_for_this_doctype.xsl"?>
  ```

- A given document instance may have different transforms applied to it in different contexts (like for different audiences, output devices, etc)
  
  ```xml
  <?xml-transform type="text/xsl" href="transform_for_pda.xsl"?>
  ```

  ```xml
  <?xml-transform type="text/xsl" href="transform_for_verbose_mode.xsl"?>
  ```

**Where Should You Transform?**

- One way to look at this is in an XML-centric way to see where transformation to and from XML might be necessary

- But this is a more general architectural question about controlling access to information. How much do you need to or want to reveal about how what you know or how you do business?

- You might limit the kinds of queries that can be asked of a database, which hides the complexity of your information model, in effect "transforming" it to a simpler one

- Even if your "business objects" or "business services" have access to all the information in your enterprise, you might expose only part of it in external interfaces

- Access to information can be controlled when it is rendered for presentation in a user interface

**Why Do Server Side Transformation?**

- The most common XML transformation is probably XML to HTML, and is most often done on the server side rather than in the browser

- In principle, XML makes it possible to distribute processing to the client side, but this is happening less than might have been predicted

- The client may not be capable of handling semantically-rich information

- You have "smart" information, and you don't want anyone else to know how smart it is (Ken Holman calls this a
Styling = Transformation + Formatting [1]

- It is better to conceive of "applying a style" to an XML document as two separate processes of "transformation" and "formatting"

- **Transformation** involves selection, reordering, or restructuring of a source document based on business rules and information requirements

- **Formatting** is applying presentation semantics or rules for rendering a document in some output context (print, display, etc.)

- Turning XML to HTML is a transformation for which the formatting is delegated to the browser (e.g., using the default presentation behavior or with CSS)

Styling = Transformation + Formatting [2]

- This way of thinking lines up conceptually with two complementary W3C Recommendations:
  - XSLT – Extensible Stylesheet Language for Transformation
  - XSL FO – Extensible Stylesheet Language Formatting Objects (for typesetting-quality control of printed XML output)

XPath

- A standard way of addressing parts of XML documents

- Defines the structures and patterns used by XML transformations, queries, and forms

- Similar in concept to addressing files on the filesystem, i.e., at a UNIX shell or MS-DOS command prompt

- Key idea is to view an XML document as a tree of information items called "nodes" - this is more abstract than thinking of it as a stream of marked-up text

The Node Tree

- XPath describes the locations of addresses of parts of XML documents by navigating through the "node tree" along a "node axis"

- There are seven types of nodes, corresponding to the different kinds of "stuff" in XML documents

- There are thirteen different axes that specify different ways of following relationships among the nodes

The Seven "Node Types"
• Element – branch of node tree, can contain other nodes
• Attribute – a leaf node of an element node
• Text – a leaf node, always a child of an element node, for holding content
• Root – the top of the node tree, contains the document element and everything else
• Comment – a leaf node that XSLT doesn't usually care about
• Processing Instruction – likewise
• Namespace – likewise

---

**Looking at the Node Tree**

• We can use an XML stylesheet to depict the node tree; a good one is at http://skew.org/xml/stylesheets/treeview/html/

  • [A simple XML document](http://skew.org/xml/stylesheets/treeview/html/)
  • [Node tree depicted in HTML](http://skew.org/xml/stylesheets/treeview/html/)

---

**The Node Axes [1]**

• There are thirteen different *axes* that define different directions of "walking the node tree" depth-first starting from the *context node*;

  • Depth-first means visiting all the children recursively throughout the entire document, shown using the numbering of the nodes in the following graphs

  • The [Self](#) Axis identifies the context node

  • The [Child](#) Axis identifies the children of the context node. This is the default so if the axis is omitted the child axis is assumed

  • The [Attribute](#) Axis identifies the attributes of the context node

  • The [Parent](#) Axis identifies the parent of the context node

---

**The Node Axes [2]**

• The [Following](#) Axis identifies all nodes after the context node in document order, excluding its attributes and descendants

  • The [Following-Sibling](#) Axis identifies all nodes that follow the context node in document order and that have the same parent

  • The [Preceding](#) Axis identifies all nodes before the context node in document order, excluding its ancestors and any attributes
Navigating and Selecting from the "Node Tree" Using "Location Paths" [1]

- A Location Path selects a Node Set by following an axis and making a test of some kind.
- A location path either begins with the root node / – absolute paths
  - Don't confuse the root node with the root element. The root node is the absolute root of the document and contains everything else, including PIs and comments
  - The root element, /RootElement, is the first element node in the document root
- Or at a context node – relative paths

Navigating and Selecting from the "Node Tree" Using "Location Paths" [2]

- Location paths can be chained together to "walk" the node tree one step at a time and refine the set of selected nodes with each step
- The first step moves along some axis from the starting node
- At each step or node test the context node changes
- The last step is what is being addressed
- // (double forward slash) looks through all the descendants of the current context... useful, but computationally wasteful

Location Path Examples

- computercatalog.xml is XML instance
  - /ComputerCatalog
    - /ComputerCatalog/RetailerName
    - /ComputerCatalog/ListOfComputers
    - /ComputerCatalog/*
    - /ComputerCatalog/*/Computer
    - //ListOfComputers/*
    - //Computer
    - //Computer/*
    - //Computer/OEM

Axes and Predicates in Location Paths
Optional predicates can be appended to further refine the selection:

- axis :: nodetest [predicate]
- If no axis is specified then "child:" is assumed
- "." is an abbreviation for "self:"
- "./" is an abbreviation for "parent:"
- "/@" is an abbreviation for "attribute:"
- "/" is a nodetest that matches any node
- "position()=" is implied if you say [≠]

### Location Path Examples - Axes and Predicates

- `/ComputerCatalog/child::*`
- `/ComputerCatalog/*`
- `//Computer/Specs/*`
- `//Computer/Specs/Specs/*`
- `//Computer[2]`
- `//Computer[2]/OEM`
- `//Computer[2]/following::*`
- `//Computer/Specs/Speed/@unit`
- `//Computer/Specs/Speed[@unit]`
- `//Specs/parent::*`
- `//Specs/[@unit]`
- `//Specs/[@unit]`
- `//Specs/Price[@*]`
- `//Specs/Price[@currency='USD']`
- `//Specs/Price[@*][1]`
- `//Specs/Price[@*][last()]`

### Interactive Location Path Examples

### Multiple Predicates

- Predicate expressions can be combined to further hone in on the data you're looking for

  - `//Computer/Specs/Speed[@unit = 'Ghz'][. > 1]`
  - `//Computer/Specs/Speed[@unit = 'Ghz'][. > 2]`
  - `//Computer/Specs[Speed/@unit = 'Ghz'][Price/@currency = 'USD'][Price < 3250]`
  - `//Computer/OEM and ProductLine and Model][Specs/Speed and Specs/MemorySize and Specs/DiskSize]`

### XPath and Namespaces

- XPath expressions can use a QName but be careful
- An expression without a prefix *WILL NOT* match anything that has one

```xml
<ev:EventCategory>
```
XPath and Namespaces - Examples

- //*
- //bk:*
- /*Title
- /*:Title
- */:
- (//bk:Title//hon:Title)

XPath Functions [1]

- XPath also defines functions that you can use in your predicates
- The XPath 1.0 Specification contains 27 built-in functions for string manipulation, arithmetic, type conversion, and further nodeset traversal
- Boolean functions make up for XPath's lack of Boolean literals

boolean(//Computer[Specs/Speed[@unit = 'Ghz'][. > 1]])
boolean(//Computer[Specs/Speed[@unit = 'Ghz'][. > 3]])
not(//Computer[Specs/Speed[@unit = 'Ghz'][. > 3]])

XPath Functions [2]

- Arithmetic operators and functions allow for numerical manipulation and calculations
- Besides the regular addition (+), subtraction (-), multiplication (*), division (div, the '/' character has special meaning in XPath), and mod, there are functions allowing for count, floor, ceiling, and so forth. Mostly these are relatively self-explanatory.

count(//Computer[Specs/Speed[@unit = 'Ghz'][. > 2]])

The expression returns the number of 'Computer' elements with greater than 2 GHz clock speeds.
XPath Functions [3]

- String manipulation functions allow for string comparison, concatenation, space normalization, substring queries, etc. Again, the functions are self-explanatory and resemble those found in the Java String classes (in quality, not quantity).

- 
  ```
  //Computer[starts-with(normalize-space(ProductLine), 'T')] 
  ```

- The first expression returns all 'Computer' elements whose space-normalized contents begin with 'T'
- The second returns 'IBM Thinkpad T22'

Readings for XML Foundations Lecture 4

- page 310 to end of chapter 8 of Beginning XML

Assignment 2 - XML to XML Transformation

- Write two transformations from one XML document model to another
  - A simple one that renames and rearranges elements
  - A more complex one that manipulates element and attribute contents
- Due on 13 September
- You won't know enough to start on this until after Thursday's lecture, but looking at the assignment before then will help you focus your reading for that lecture