Announcements (March 24)
- Homework #3 will be assigned next Tuesday
- Reading assignment due next Wednesday
  - XML processing in Lore (VLDB 1999) and Niagara (VLDB 2003)
- Project milestone 2 due next Thursday

XSLT
- XML-to-XML rule-based transformation language
- An XSLT program is an XML document itself
- Used most frequently as a stylesheet language
- Version 1.0 a W3C recommendation
- Version 2.0 under development together with XPath 2.0

XSLT program
- An XSLT program is an XML document containing
  - Elements in the `<xsl:` namespace
  - Elements in user namespace
- The result of evaluating an XSLT program on an
  input XML document = the XSLT document where each `<xsl:` element
  has been replaced with the result of its evaluation
- Uses XPath as a sub-language

XSLT example
- Find titles of books authored by "Abiteboul"
  ```xml
  <?xml version="1.0"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">
    <xsl:template match="book[author='Abiteboul']">
      <booktitle>
        <xsl:value-of select="title"/>
      </booktitle>
    </xsl:template>
  </xsl:stylesheet>
  ```
  - Not quite; we will see why later

XSLT elements
- Element describing transformation rules
  - `<xsl:template>`
- Elements describing rule execution control
  - `<xsl:apply-templates>`
  - `<xsl:call-template>`
- Elements describing instructions
  - `<xsl:if>`, `<xsl:for-each>`, `<xsl:sort>`, etc.
- Elements generating output
<xsl:template>
<xsl:template match="book[author='Abiteboul']">
  <booktitle>
    <xsl:value-of select="title"/>
  </booktitle>
</xsl:template>

❖ <xsl:template match="match_expr"> is the basic XSLT construct describing a transformation rule
  • match_expr is an XPath-like expression specifying which nodes this rule applies to
  • xsl:value-of select="xpath_expr" within the context of the node matching the template, and converts the result sequence to a string
  • <booktitle> and </booktitle> simply get copied to the output for each node match

Removing the extra output
❖ Add the following template:
  <xsl:template match="text()|@*"/>
❖ This template matches all text and attributes
❖ XPath features
  • text() is a node test that matches any text node
  • @* matches any attribute
  • | means “or” in XPath
❖ Body of the rule is empty, so all text and attributes become empty string
  • This rule effectively filters out things not matched by the other rule

<xsl:copy-of>
❖ Another slightly different example: return (entire) books authored by “Abiteboul”
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="book[author='Abiteboul']">
      <book>
        <xsl:attribute name="title">
          <xsl:value-of select="normalize-space(title)"/>
        </xsl:attribute>
      </book>
    </xsl:template>
  </xsl:stylesheet>
❖ <xsl:copy-of select="xpath_expr"/> copies the entire contents (including tag structures) of the node-set returned by xpath_expr to the output

Template in action
❖ Example XML fragment
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>…</section>…
  </book>
    <title>A First Course in Databases</title>
    <author>Ullman</author>
    <author>Widom</author>
    <publisher>Prentice-Hall</publisher>
    <year>2002</year>
    <section>…</section>…
  </book>
❖ Template applies
  • <booktitle>Foundations of Databases</booktitle>
❖ Template does not apply; default behavior is to process the node recursively and print out all text nodes
❖ Another slightly different example:
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="text()|@*"/>
  </xsl:stylesheet>
❖ <xsl:copy-of select="xpath_expr"/> copies the entire contents (including tag structures) of the node-set returned by xpath_expr to the output

Formatting XML into HTML
❖ Example templates to
  • Render a book title in italics in HTML
  • Render the authors as a comma-separated list
  <xsl:template match="book/title">
    <i><xsl:value-of select="normalize-space(.)"/></i>
  </xsl:template>
  <xsl:template match="book/author[1]">
    <xsl:value-of select="normalize-space(.)"/>
  </xsl:template>
  <xsl:template match="book/author[position()>1]">
    <xsl:text>, </xsl:text>
    <xsl:value-of select="normalize-space(.)"/>
  </xsl:template>
  <xsl:template>
    <xsl:text> allows precise control of white space in output
</xsl:template>
Example: generate a table of contents
- Display books in an HTML unordered list
- For each book, first display its title, and then display its sections in an HTML ordered list
- For each section, first display its title, and then display its subsections in an HTML ordered list

Example continued
- <xsl:template match="book">
  <li>
    <xsl:apply-templates select="title"/>
    <ol><xsl:apply-templates select="section"/></ol>
  </li>
</xsl:template>

One problem remains
- Even if a book or a section has no sections, we will still generate an empty \(<ol></ol>\) element

White space control
- White space is everywhere in XML.
  - A fix using <xsl:if>: replace
    \(<ol><xsl:apply-templates select="section"/></ol>\)
    with \(<ol><xsl:if test="section">\(<ol><xsl:apply-templates select="section"/></ol>\)</xsl:if>

Named templates with parameters
- Define a generic template for rendering a list of things as a comma-separated list
  - Cannot use match because we do not know in advance the things to render
    <xsl:template name="comma-separated-list">
      <xsl:param name="things-to-be-formatted"/>
      <xsl:for-each select="$things-to-be-formatted">
        <xsl:if test="position()>1">, </xsl:if>
        <xsl:value-of select="normalize-space(.)"/>
      </xsl:for-each>
    </xsl:template>
Calling templates & passing parameters

- Use the generic template

```xml
<xsl:template match="book">
  <xsl:text>: </xsl:text>
  <xsl:call-template name="comma-separated-list">
    <xsl:with-param name="things-to-be-formatted" select="author"/>
  </xsl:call-template>
</xsl:template>
```

- `<xsl:with-param>` invokes the named template without changing the context

- `<xsl:call-template>` evaluates `xpath_expr` and passes its result as the value of the parameter `para_name`

XSLT summary

- Used often as a stylesheet language, but can be considered a query language too
  - Very expressive, with full recursion
    - Cannot be replaced by XQuery?
      - Well, XQuery actually support user-defined functions, which can be recursive
  - Easily non-terminating, difficult to optimize
  - Features like dynamic scoping really help in text processing
  - So many features, so little time!

Review

- XML: tree (or graph)-structured data
- DTD: simple schema for XML
  - Well-formed XML: syntactically correct
  - Valid XML: well-formed and conforms to a DTD
- XPath: path expression language for XML
  - An XPath expression selects a list of nodes in an XML document
  - Used in other languages
- XQuery: SQL-like query language for XML
  - FLWOR expression, quantified expression, aggregation, etc.
- XSLT: stylesheet language for XML, in XML
  - Transforms input XML by applying template rules recursively on the structure of input XML

XML API’s

- SAX (Simple API for XML)
  - Started out as a Java API, but now exists for other languages too
  - Streaming input; callbacks for events (start/end of document and elements, chunk of characters, etc.)
- DOM (Document Object Model)
  - Language-neutral API with implementations in Java, C++, etc.
  - Converts input into a main-memory tree; supports tree traversal, construction, and in-place modification
- JAXB (Java Architecure for XML Binding)
  - XML Schema to Java objects