Announcements (October 25)
- Homework #3 due in 1½ weeks
  - Start early!
- Project milestone #2 due in 2 weeks

XSLT

- XML-to-XML rule-based transformation language
  - Used most frequently as a stylesheet language
  - An XSLT program is an XML document itself
  - Current version is 2.0; W3C recommendation since January 2007

XSLT processor

Input XML

Output XML

Actually, output does not need to be in XML in general

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
- Basic ideas
  - Templates specify how to transform matching input nodes
  - Structural recursion applies templates to input trees recursively
  - Uses XPath as a sub-language

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>`
- Elements generating output
  - `<xsl:value-of>`
  - `<xsl:attribute>`
  - `<xsl:copy-of>`
  - `<xsl:text>`
- Not quite; we will see why later

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>
```
<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"/> evaluates 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
  </xsl:template>

  <xsl:template match="text()|@*"/>

  <xsl:template match="book[author='Abiteboul']">
    <booktitle>
      <xsl:value-of select="normalize-space(title)"/>
    </booktitle>
  </xsl:template>

  <xsl:attribute name="attr">
    adds an attributed named attr with value body to the parent element in the output
  </xsl:template>

  <xsl:copy-of select="/"/> copies the entire contents (including tag structures) of the node-set returned by xpath_expr to the output
</xsl:template>

Template in action

<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>
  …
</book>

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

Formatting XML into HTML

Another slightly different example: return (entire) books authored by “Abiteboul”

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:text> allows precise control of white space in output

Example XML fragment

Template applies

Template does not apply; default behavior is to process the node recursively and print out all text nodes

A First Course in Databases
Ullman
Widom
Prentice-Hall
2002

Adding an attribute

<book>
  <xsl:attribute name="attr">
    body
  </xsl:attribute>
</book>
<xsl:apply-templates>
  <xsl:template match="title">
    <xsl:value-of select="normalize-space(.)"/>
  </xsl:template>
  <xsl:template match="section">
    <li>
      <xsl:apply-templates select="title"/>
      <ol><xsl:apply-templates select="section"></ol>
    </li>
  </xsl:template>
</xsl:apply-templates>

Example continued

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

Output control

<xsl:output method="html" indent="yes"/>

Specify that output

- Will be HTML
- Will be indented to make reading easier
- Other possible method values include "text", "xml"

For XML output method, set
omit-xml-declaration="yes"
to suppress "<?xml ...?>" at the beginning of the output

White space control

- White space is everywhere in XML

  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
  </book>

- "..." goes into a text node (assuming no DTD)
- "..." goes into another text node

- Specify <xsl:strip-space elements="*"/> to remove text nodes (under any element) containing only white space
- To strip leading and trailing white space and replace any sequence of white space characters by a single space, specify <xsl:template match="text()">
  <xsl:value-of select="normalize-space()"/>
</xsl:template>

<xsl:for-each>
  <xsl:for-each select="xpath_expr">
    body
  </xsl:for-each>
  <xsl:template match="book">
    <xsl:for-each select="author">
      <xsl:if test="position()>1">, </xsl:if>
      <xsl:value-of select="normalize-space(.)"/>
    </xsl:if>
  </xsl:template>
</xsl:for-each>
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
  ```xslt
  <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
  ```xslt
  <xsl:template match="book">
    <xsl:value-of select="normalize-space(title)"/>
    <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>
  ```

XSLT summary

- Used often as a stylesheet language, but can be considered a query language too
  - Grouping in XSLT 2.0 (`<xsl:for-each-group>`)
  - Very expressive, with full recursion
    - Cannot be replaced by XQuery?
      - Well, XQuery supports user-defined functions, which can be recursive
  - Easily non-terminating, difficult to optimize
    - Cannot replace XQuery
- 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