XSLT

Introduction to Databases
CompSci 316 Fall 2014
Announcements (Thu., Oct. 30)

• Homework #3 due next Thursday
• Project milestone #2 due the following Thursday
• Play/Java example app posted
XSLT

- XML-to-XML rule-based transformation language
  - Used most frequently as a stylesheet language
  - An XSLT program is an XML document itself

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

• Result of evaluating an XSLT program on an input XML document = the XSLT document where each <xsl: element is 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>`, etc.

- Elements generating output
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
• `<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 matched
Template in action

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

• 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>
</book>

    <title>A First Course in Databases</title>
    <author>Ullman</author>
    <author>Widom</author>
    <publisher>Prentice-Hall</publisher>
    <year>2002</year>
</book>

Template applies

<booktitle>
    Foundations of Databases
</booktitle>

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

A First Course in Databases
Ullman
Widom
Prentice-Hall
2002
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
\texttt{<xsl:element> and <xsl:attribute>}

- Again, find titles of books authored by “Abiteboul,” but make the output look like \texttt{<BOOK title="booktitle"/>}

\begin{verbatim}
... ...
\<xsl:template match="book[author='Abiteboul']">
  \<BOOK title="{normalize-space(title)}"/>
\</xsl:template>  \{expr\} evaluates expr and replaces itself with the output string
... ...
\end{verbatim}

- A more general method

\begin{verbatim}
... ...
\<xsl:template match="book[author='Abiteboul']">
  \<xsl:element name="{upper-case(name())}">
    \<xsl:attribute name="title">
      \<xsl:value-of select="normalize-space(title)"/>
    \</xsl:attribute>
  \</xsl:element>
\</xsl:template>

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

\<xsl:element> creates an element of the given name in the output
\end{verbatim}
• Another slightly different example: return (entire) 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="text()|@*"/>
 <xsl:template match="book[author='Abiteboul']">
   <xsl:copy-of select="."/>
 </xsl:template>
</xsl:stylesheet>
```

• `<xsl:copy-of select="xpathExpr"/>
 copies the entire contents (including tag structures) of the node-set returned by `xpathExpr` 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

```xml
<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 more precise control of white space in output
<xsl:apply-templates>

• 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

<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></xsl:apply-templates>
  </li>
</xsl:template>

(Continue on next slide)

<xsl:apply-templates select="xpath_expr"/>

applies templates recursively to the node-set returned by xpath_expr
Example continued

• One problem remains
  • Even if a book or a section has no sections, we will still generate an empty `<ol></ol>` element
<xsl:if>

- A fix using <xsl:if>: replace
  <ol><xsl:apply-templates select="section"/></ol>
with
  <xsl:if test="section">
    <ol><xsl:apply-templates select="section"/></ol>
  </xsl:if>

- The body of <xsl:if test="xpath_cond"> is processed only if xpathCond evaluates to true
Output control

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

- Specifies that output
  - Will be HTML
  - Will be indented to make reading easier

- Other possible method values include "text", "xml"
  - For XML output, if you want to suppress "<?xml ...?>" at the beginning of the output, set attribute `omit-xml-declaration="yes"`
White space control

- White space is everywhere in XML

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

  - 
  - “…” goes into a text node (assuming no DTD)
  - “…” “Foundations of Databases…” goes in 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 one, specify

  ```xml
  <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:for-each>
    ... ...
  </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

```xml
<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 with parameters

• Use the named template

```xml
<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>
  <br/>
</xsl:template>
```

• `<xsl:with-param name="para_name" select="xpath_expr">` evaluates `xpath_expr` and passes its result as the value of the parameter `para_name`.

• `<xsl:call-template>` invokes the named template without changing the context.
Other useful features

• `<xsl:text>&#10;</xsl:text>` inserts a newline in the output
• `<xsl:message>` for debugging
  • `<xsl:message terminate="yes">` exits the program
• `<xsl:variable>` defines a (constant) variable
• `<xsl:function>` defines a function
• `<xsl:key>` defines a key that can be used for lookups
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
• XML Schema: a more sophisticated schema for XML
• 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