XSLT: Using XML to transform other XML files

Introduction to databases
CSC343 Fall 2011
Ryan Johnson

Thanks to Manos Papagelis, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides

What is XSLT?

• An XML-based style sheet language for XML documents
  – Tells web browser how to display data
  – Similar to CSS for HTML, but with absolute control over input
• Achieves similar results to XQuery
  – Pro: supported by all non-ancient web browsers
  – Con: XML syntax is [really] verbose

XSLT processing model

• The XSLT processing model involves:
  – one (or more) XML source documents
  – one (or more) XSLT style sheet modules
  – the XSLT template processing engine
  – one (or more) result documents

• XSLT is most often used to
  – convert data between different XML schemas
  – to convert XML data into web pages (HTML)

XSLT vs XQUERY

• XSLT capabilities overlap with XQuery
  – same data model, type system, function library
  – both include XPath 2.0 as a sublanguage
• Rooted in different traditions and communities
  – XSLT: primarily conceived as a stylesheet language
  – XQuery: primarily conceived as a database query language
• When to use:
  – XSLT: stronger in its handling of narrative documents with more flexible structure
  – XQuery: stronger in its data handling (e.g. when performing relational joins)
Example 1: Transform. XML to HTML

The XML Document

<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  ...
  <cd>
    <title>Unchain my heart</title>
    <artist>Joe Cocker</artist>
    <country>USA</country>
    <company>EMI</company>
    <price>8.20</price>
    <year>1987</year>
  </cd>
</catalog>

The XSLT for the XML

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body></html>
  </xsl:template>
</xsl:stylesheet>

The HTML Result Document

My CD Collection

<table>
<thead>
<tr>
<th>Title</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empire Burlesque</td>
<td>Bob Dylan</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Unchain my heart</td>
<td>Joe Cocker</td>
</tr>
</tbody>
</table>

Example 2: Transform. XML to XML

The XML Document

<?xml version="1.0"?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
Example 2: Transform. XML to XML

```xml
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>
  <xsl:output method='xml' indent='yes'/>
  <xsl:template match='persons'>
    <root>
      <xsl:apply-templates select='person'/>
    </root>
  </xsl:template>
  <xsl:template match='person'>
    <name username='{@username}'>
      <xsl:value-of select='name'/>
    </name>
  </xsl:template>
</xsl:stylesheet>
```

The XSLT for the XML

The HTML Result Document

```
<root>
  <name username='JS1'>John</name>
  <name username='MI1'>Morka</name>
</root>
```

XSLT in a nutshell

- All XSL commands are xml (**xsl**: namespace)
- Everything works by templates
  - Output generated by instantiating templates
  - Templates can call other templates (recursion allowed)
  - No output unless some template matches document root
  - Overloading rules for resolving 2+ matching templates
- Builds on material we already know
  - XPath used to select nodes for various uses
  - Embedded expressions use {} syntax from XQuery

Control flow in XSL

- **<xsl:for-each select='...'...</xsl:for-each>**
  - Used to select every XML element of a specified node-set
  - Instantiate the template once for each item
  - Inside the template, context node (.) refers to “current” item
- **<xsl:if test='...'...</xsl:if>**
  - Used to put a conditional test against the content of the XML file
  - Instantiate the template inside if the condition is true
  - No ’else’ (use a second **xsl:if** or **xsl:choose** instead)
- **<xsl:choose>...</xsl:choose>**
  - used along with **<xsl:when>** and **<xsl:otherwise>** to express multiple conditional tests
  - **<xsl:when test='...'...</xsl:when>** (same usage as **xsl:if**)
  - **<xsl:otherwise>...</xsl:otherwise>** (no-match template)
  - Instantiate only one template (first match wins)
Sorting elements

- Elements can be sorted inside for-each loops
- `<xsl:sort select='...' order='...' />`
  - select should identify the element to sort on
  - order can be “ascending” or “descending”
- Example:

```xml
<xsl:for-each select='//book'>
  <xsl:sort select='author/surname' />
  <xsl:sort select='author/given-name' />
  ...
</xsl:for-each>
```

Declaring variables

- `<xsl:variable name='...' select='...'/>
  - Store the result of an XPath query in a variable
- `<xsl:variable name='...'>...</xsl:variable>
  - Instantiate a template and store its result in a variable
- Notes
  - Default value (if none specified) is an empty string
  - Cannot be overwritten or updated once set
  - Sometimes valid as input to templates (more later)

Declaring templates

- Idea: A template contains rules to apply when a specified node is matched. Results are inserted into the output
- `<xsl:template match='...' mode='...'>`
  - Template instantiated for every element satisfying the match’s XPath expression
  - Optional mode attribute allows to select from 2+ templates
- `<xsl:template name='...'>`
  - Template instantiated explicitly
  - OK to name a match template

Applying vs. calling templates

- `<xsl:apply-templates select='...' mode='...'/>`
  - Applies a template to the current element or to the its child nodes
  - Format a node (default: .) using "best" matching template
  - Complicated priority rules, but more-specific and declared-later templates usually chosen
  - When in doubt disambiguate using optional mode attribute
- `<xsl:call-template name='...'/>`
  - Instantiate the named template
  - WARNING: context node unchanged by call!
  - Call a template which uses "." => possible infinite loop
- `<xsl:parameter> and <xsl:with-param>`
  - Templates can declare xsl:parameter (acts just like xsl:variable)
  - Same effect if template is called normally
  - Template call can have xsl:with-param child elements
  - Overrides value which xsl:parameter would have assigned
Applying templates to variables

- Only select-type variables have XML content
  - Otherwise, contents treated as text
  - Limitation of both XSLT 1.0 and 2.0 (fixed by EXSLT, next)

- Example:
  ```xml
  <xsl:variable name='foo' select='//foo'/>
  <xsl:variable name='bar'>
    <xsl:apply-templates select='$foo'/>
  </xsl:variable>
  <xsl:variable name='baz'>
    <xsl:apply-templates select='$bar'/>
  </xsl:variable>
  ```

EXSLT

- The EXSLT spec addresses weaknesses of XSLT
  - Several “modules” each with its own namespace
  - Supported by modern XSLT processors (IE9, FF3+, etc.)

- Have to ask for it by adding namespaces

- Especially handy modules:
  - xmlns:exsl="http://exslt.org/common"
    - Provides `exsl:node-set`, a function to turn text into XML
  - xmlns:str="http://exslt.org/strings"
    - Provides `str:split`, which works just like python’s `split()`

Outputting things

- `<xsl:text>...</xsl:text>`
  - Adds text (including whitespace!) to output
  - Optional yes/no attribute: `disable-output-escaping`
    - In theory, good for outputting ‘&’, ‘<’ etc.
    - WARNING: optional part of spec, often ignored

- `<xsl:value-of select='...'/>`
  - Converts the selected node(s) to text and outputs it

- `<xsl:copy-of select='...'/>`
  - Embeds the selected node(s) directly into the output
  - e.g. good for emitting snippets of pre-constructed HTML

Global output controls

- `<xsl:output method='...' indent='...'/>`
  - Defines the format of the output document
  - Methods include xml (default), html, and text
  - Defaults to html method if parser detects HTML output
  - Enabling indentation makes output prettier
  - Bad idea if whitespace matters for presentation
  - Lots of other options (look them up)
Whitespace handling

- Default: any text node with non-space characters preserves all space
- Override #1: `<xsl:strip-space elements='...
  - Specifies that the elements named (separated by spaces) should normalize whitespace before output
  - A corresponding preserve-space command exists also
  - Same conflict resolution as used for template matching
- Override #2: `<... xml:space='preserve' ...>
  - Little-known xml attribute that tells XML processors whether whitespace matters
  - Inherited by child elements (xml:space='default' cancels)
  - Watch out for elements forbidding text (e.g. `xsl:call-template`)

Keys

- Like the ID/IDREF from DTD but better
  - No need for the DTD (often ignored even when present)
  - Multiple (named) ID domains allowed
- `<xsl:key name='...' match='...' use='...' />
  - Declares a key (name) for all nodes which match, as well as how to find (use) the value of the key
- `{key('...', ...)` retrieves the value of a key
- Example:
  - Find all links which point to an image in the same file
    `<xsl:key name='imglinks' match='//a' use='@href'/>
    `{key('imglinks', '//img/@src')}

Sample XSLT skeleton

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match='/'
    name='show-tree'>
    <html>
      <body>
        <xsl:call-template name='show-tree'>
          <xsl:with-param name='e' select='*'/>
          <xsl:with-param name='i' select='0'/>
        </xsl:call-template>
        <xsl:template match='*'>
          <xsl:value-of select='name(.)'/>
        </xsl:template>
        <xsl:template match='nct_id'>
          <span style='color:red'>
            <xsl:value-of select='self::text()'/>
          </span>
        </xsl:template>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Sample: root node template

```
<xsl:template match='/*'>
  <html>
    <body>
      <xsl:call-template name='show-tree'>
        <xsl:with-param name='e' select='*'/>
        <xsl:with-param name='i' select='0'/>
      </xsl:call-template>
      <xsl:template match='*'>
        <xsl:value-of select='name(.)'/>
      </xsl:template>
      <xsl:template match='nct_id'>
        <span style='color:red'>
          <xsl:value-of select='self::text()'/>
        </span>
      </xsl:template>
    </body>
  </html>
</xsl:template>
```
Sample: recursive template

```xml
<xsl:template name='show-tree'>
  <xsl:param name='e'/>
  <xsl:param name='i'/>
  <div style='padding-left:{$i}px'>
    <xsl:apply-templates select='name($e)' />
    <xsl:if test='$e/@*'>
      (<xsl:for-each select='$e/@*'>
        <xsl:value-of select='name(.)'/>,
      </xsl:for-each>,
    </xsl:if>
    <xsl:for-each select='$e/*'>
      <xsl:call-template name='show-tree'>
        <xsl:with-param name='e' select='.'/>
        <xsl:with-param name='i' select='30'/>
      </xsl:call-template>
    </xsl:for-each>
  </div>
</xsl:template>
```

Sample XSLT + clinical_study XML

**HTML Output**

```html
<html><body>
  <div style="margin-left:0px;">
    clinical_study (rank , )
    required_header
    download_date
  </div>
  ... 
</body></html>
```

**Rendered HTML**

```html
  <div style="margin-left:0px;">
    clinical_study (rank , )
    required_header
    download_date
  </div>
  ... 
```