INTERNET & WEB APPLICATION DEVELOPMENT SWE 444

Fall Semester 2008-2009 (081)

Module 4 (III): XSL

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives/Outline

- Objectives
  - Learn to create and use simple XSL style sheets to render XML document data

- Outline
  - What is XSL?
  - Some XSLT Constructs
    - xsl:value-of
    - xsl:for-each
    - xsl:if
    - xsl:choose
    - xsl:sort
    - xsl:text
    - xsl:attribute
  - Templates
  - XSL on the Client
  - XSL on the Server

What is XSL?

- XSL stands for eXtensible Stylesheet Language
  - a standard recommended by the W3C
  - http://www.w3.org/TR/xsl/
- CSS was designed for styling HTML pages, and can be used to style XML pages
- XSL was designed specifically to style XML pages, and is much more sophisticated than CSS
- XSL consists of three languages:
  - XSLT (XSL Transformations) is a language used to transform XML documents into other kinds of documents (most commonly HTML, so they can be displayed)
  - XPath is a language to select parts of an XML document to transform with XSLT
  - XSL-FO (XSL Formatting Objects) is a replacement for CSS
    - The future of XSL-FO as a standard is uncertain, because much of its functionality overlaps with that provided by cascading style sheets (CSS) and the HTML tag set

How does it work?

- The XML *source document* is parsed into an XML source tree
- You use XPath to define *templates* that match parts of the source tree
- You use XSLT to transform the matched part and put the transformed information into the *result tree*
- The result tree is output as a *result document*
- Parts of the source document that are not matched by a template are typically copied unchanged

Simple XPath

- Here's a simple XML document:

```
<?xml version="1.0"?>
<library>
  <book>
    <title>XML</title>
    <author>Gregory Brill</author>
  </book>
  <book>
    <title>Java and XML</title>
    <author>Brett Scott</author>
  </book>
</library >
```

- XPath expressions look a lot like paths in a computer file system
  - `/` means the document itself (but no specific elements)
  - `/library` selects the root element
  - `/library/book` selects *every* book element
  - `//author` selects *every* author element, wherever it occurs

Simple XSLT

- <xsl:for-each select="//book"> loops through every book element, everywhere in the document
- <xsl:value-of select="title"/> chooses the content of the title element at the current location
- <xsl:for-each select="//book">
  <xsl:value-of select="title"/>
  </xsl:for-each>
  chooses the content of the title element for each book in the XML document

Example: Using XSL to Create HTML

- Our goal is to turn this:

```
<?xml version="1.0"?>
<library>
  <book>
    <title>XML</title>
    <author>Gregory Brill</author>
  </book>
  <book>
    <title>Java and XML</title>
    <author>Brett Scott</author>
  </book>
</library >
```

- Into HTML that displays something like *this:*

Book Titles:
- XML
- Java and XML

Book Authors:
- Gregory Brill
- Brett Scott

- Note that we've grouped titles and authors separately

Example: What we need to do

- We need to save our XML into a file (let's call it `books.xml`)
- We need to create a file (`books.xsl`)
  - Describes how to select elements from `books.xml` and embed them into an HTML page
  - We do this by intermixing the HTML and the XSL in the `books.xsl` file
- We need to add a line to our `books.xml` file to tell it to refer to `books.xsl` for formatting information

Example (cont.)

books.xml

```
<?xml version="1.0"?>
<?xml-stylesheet  type="text/xsl" href="books.xsl"?>
<library>
  <book>
    <title>XML</title>
    <author>Gregory Brill</author>
  </book>
  <book>
    <title>Java and XML</title>
    <author>Brett McLaughlin</author>
  </book>
</library >
```

This tells you where to find the XSL file

Example (cont.)

Desired HTML

```
<html>
  <head>
    <title>Book Titles and Authors</title>
  </head>
  <body>
    <h2>Book titles:</h2>
    <ul>
        <li>XML</li>
        <li>Java and XML</li>
    </ul>
    <h2>Book authors:</h2>
    <ul>
        <li>Gregory Brill</li>
        <li>Brett Scott</li>
    </ul>
  </body>
</html>
```

Red text is data extracted from the XML document

Blue text is our HTML template

We don't necessarily know how much data we will have

The .xsl file

- An XSLT document has the .xsl extension
- The XSLT document begins with:
  - `<?xml version="1.0"?>`
  - `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
- Contains one or more templates, such as:
  - `<xsl:template match="/"> ... </xsl:template>`
- And ends with:
  - `</xsl:stylesheet>`
- The template `<xsl:template match="/">` says to select the entire file
  - You can think of this as selecting the root node of the XML tree

The .xsl file outline

```
<?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> ... </html>


      </xsl:template>

</xsl:stylesheet>
```

Selecting Titles and Authors

Notice the `xsl:for-each` loop

```
<h2>Book titles:</h2>
   <ul>
      <xsl:for-each  select="//book">
         <li>
            <xsl:value-of  select="title"/>
         </li>
      </xsl:for-each>
   </ul>
<h2>Book authors:</h2>
   ...same thing, replacing title with author
```

- Notice that XSL can rearrange the data; the HTML result can present information in a different order than the XML

Example: All of books.xml

```
<?xml version="1.0"?>
<?xml-stylesheet  type="text/xsl"
href="books.xsl"?>
<library>
  <book>
    <title>XML</title>
    <author>Gregory Brill</author>
  </book>
  <book>
    <title>Java and XML</title>
    <author>Brett Scott</author>
  </book>
</library >
```

Note: if you do View Source, *this* is what you will see, not the resultant HTML

Example: All of books.xsl

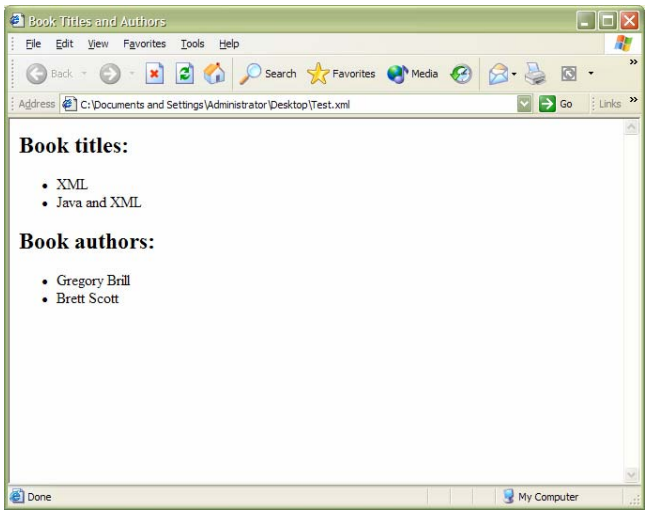
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/
                          XSL/Transform">
<xsl:template match="/">
<html>
  <head>
    <title>Book Titles and Authors</title>
  </head>
  <body>
    <h2>Book titles:</h2>
    <ul>
      <xsl:for-each select="//book">
        <li>
          <xsl:value-of select="title"/>
        </li>
      </xsl:for-each>
    </ul>
```

```
    <h2>Book authors:</h2>
      <ul>
        <xsl:for-each
               select="//book">
          <li>
            <xsl:value-of
                  select="author"/>
          </li>
        </xsl:for-each>
      </ul>
    </body>
  </html>
  </xsl:template>
  </xsl:stylesheet>
```


How to use it

- In a modern browser, such as Netscape 6, IE 6, or Mozilla 1.0, open the XML file
  - Older browsers will ignore the XSL and just show the XML contents as continuous text
- You can use a program such as Xalan, MSXML, or Saxon to create the HTML as a file
  - This can be done on the server side, so that all the client-side browser sees is plain HTML
  - The server can create the HTML dynamically from the information currently in XML

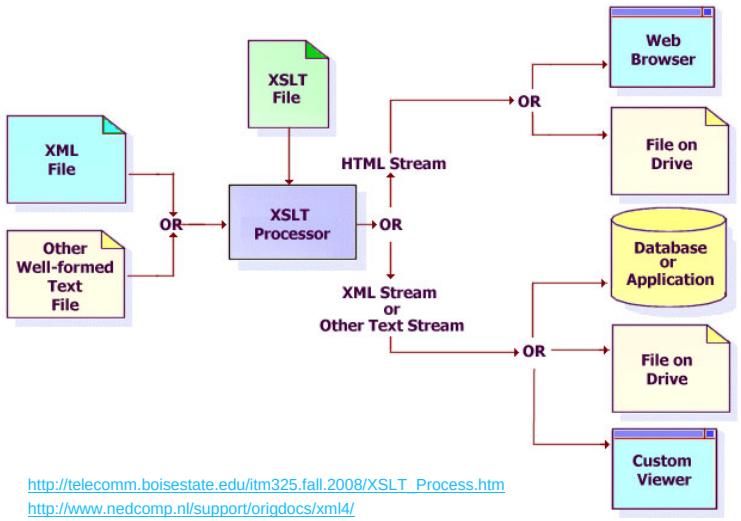
The result (in IE)



XSLT

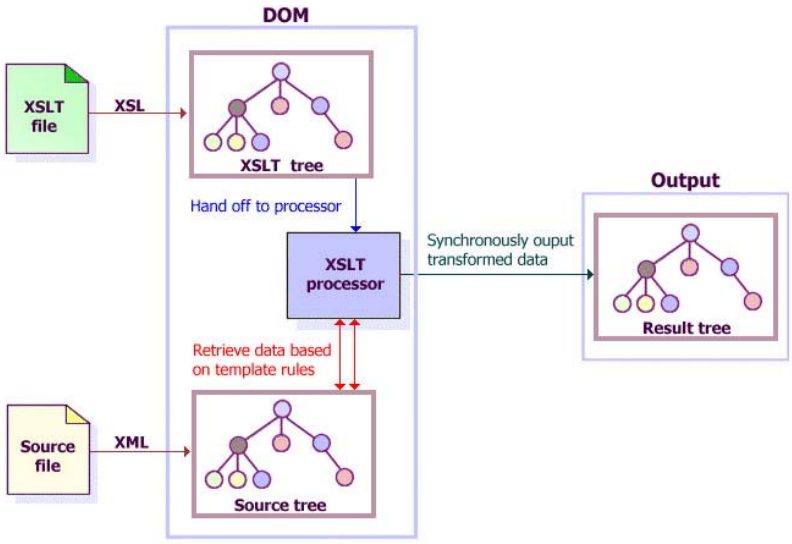
- XSLT stands for eXtensible Stylesheet Language Transformations
- XSLT is used to transform XML documents into other kinds of documents--usually, but not necessarily, XHTML
- XSLT uses two input files:
  - The XML document containing the actual data
  - The XSL document containing both the "framework" in which to insert the data, and XSLT commands to do so

Understanding the XSLT Process



http://telecomm.boisestate.edu/itm325.fall.2008/XSLT_Process.htm
http://www.nedcomp.nl/support/origdocs/xml4/

The XSLT Processor



Where XSLT can be used

- A server can use XSLT to change XML files into HTML files before sending them to the client
- A modern browser can use XSLT to change XML into HTML on the client side
  - This is what we will mostly be doing here
- Most users seldom update their browsers
  - If you want "everyone" to see your pages, do any XSL processing on the server side
  - Otherwise, think about what best fits your situation

xsl:value-of

- `<xsl:value-of select="XPath expression"/>`

selects the contents of an element and adds it to the output stream

- The select attribute is required
- Notice that `xsl:value-of` is not a container, hence it needs to end with a slash

xsl:for-each

- `xsl:for-each` is a kind of loop statement
- The syntax is

```
<xsl:for-each select="XPath expression">
        Text to insert and rules to apply
</xsl:for-each>
```

- Example:
  - to select every book (`//book`) and make an unordered list (`<ul>`) of their titles (`title`), use:

```
<ul>
  <xsl:for-each select="//book">
     <li> <xsl:value-of select="title"/> </li>
  </xsl:for-each>
</ul>
```


Filtering Output

- You can filter (restrict) output by adding a criterion to the select attribute's value:
- Example: select book titles by Brett Scott

```
<ul>
  <xsl:for-each select="//book">
     <li>
      <xsl:value-of
       select="title[../author='Brett Scott']"/>
     </li>
  </xsl:for-each>
</ul>
```

Filter Details

- Here is the filter we just used:

```
<xsl:value-of
    select="title[../author='Brett Scott']"/>
```

- `author` is a sibling of `title`, so from `title` we have to go up to its parent, `book`, then back down to `author`
- This filter requires a quote within a quote, so we need both single quotes and double quotes
- Legal filter operators are:

  `=` `!=` `<` `>`
  - Numbers should be quoted

But it doesn't work right!

- Here's what we did:

```
<xsl:for-each select="//book">
    <li>
      <xsl:value-of
        select="title[../author='Brett Scott']"/>
    </li>
 </xsl:for-each>
```

- This will output `<li>` and `</li>` for every book, so we will get empty bullets for authors other than Brett Scott
- There is no obvious way to solve this with just `xsl:value-of`

xsl:if

- Allows us to include content if a given condition (in the test attribute) is true
- Example

```
<xsl:for-each  select="//book">
     <xsl:if  test="author='Brett Scott'">
       <li>
         <xsl:value-of  select="title"/>
       </li>
     </xsl:if>
  </xsl:for-each>
```

- This does work correctly!

xsl:choose

- The `xsl:choose ... xsl:when ... xsl:otherwise` construct is XML's equivalent of Java's switch ... case ... default statement
- xsl:choose is often used within an xsl:for-each loop
- The syntax is:

```
<xsl:choose>
     <xsl:when test="some condition">
           ... some code ...
     </xsl:when>
     <xsl:otherwise>
           ... some code ...
     </xsl:otherwise>
</xsl:choose>
```

xsl:sort

- You can place an `xsl:sort` inside an `xsl:for-each`
- The attribute of the sort tells what field to sort on
- Example:

```
<ul>
    <xsl:for-each select="//book">
      <xsl:sort select="author"/>
        <li> <xsl:value-of select="title"/> by
                <xsl:value-of select="author">
            </li>
     </xsl:for-each>
</ul>
```

  - This example creates a list of titles and authors, sorted by author

xsl:text

- Used inside templates to indicate that its contents should be output as text
  - Its contents are pure text, not elements, and white space is not collapsed
- `<xsl:text>...</xsl:text>` helps deal with two common problems:
  - XSL isn't very careful with whitespace in the document
    - This doesn't matter much for HTML, which collapses all whitespace anyway
    - `<xsl:text>` gives you much better control over whitespace; it acts like the `<pre>` element in HTML
  - Since XML defines only five entities, you cannot readily put other entities (such as ` `) in your XSL
    - These are & (&), < (<), > (>), " ("), ' (')
    - Others can be inserted using their decimal or hexadecimal number forms
    - You may use the following secret formula for entities:

```
<xsl:text disable-output-escaping="yes">&amp;nbsp;</xsl:text>
```

- A "yes" value means special characters like "<" should be output as it. "no" indicates that "<" should be output as "<". Default is "no"

Creating Tags from XML Data

- Suppose the XML contains

```
<name>Dr. Scott's Home Page</name>
<url>http://www.kfupm.edu/~scott</url>
```

- And you want to turn this into

```
<a href="http://www.kfupm.edu/~scott">
Dr. Scott's Home Page</a>
```

- We need additional tools to do this
  - It doesn't even help if the XML directly contains `<a href="http://www.kfupm.edu/~scott">Dr. Scott's Home Page</a>` -- we still can't move it to the output
  - The same problem occurs with images in the XML
- A reason for the above is that attribute fields may not contain reserved characters like < and > in XML

Creating Tags - solution 1

- Suppose the XML contains

```
<name>Dr. Scott's Home Page</name>
<url>http://www.kfupm.edu/~scott</url>
```

- `<xsl:attribute name="...">` adds the named attribute to the enclosing tag
- The value of the attribute is the content of this tag
- Example:

```
<a>
    <xsl:attribute name="href">
        <xsl:value-of select="url"/>
    </xsl:attribute>
    <xsl:value-of select="name"/>
</a>
```

- Result:

```
<a href="http://www.kfupm.edu/~scott">
Dr. Scott's Home Page</a>
```

Creating Tags - solution 2

- Suppose the XML contains
```
<name>Dr. Scott's Home Page</name>
<url>http://www.kfupm.edu/~scott</url>
```
- An attribute value template (AVT) consists of braces { } inside the attribute value
- The content of the braces is replaced by its value
- Example:
  - ```
    <a href="{url}">
       <xsl:value-of select="name"/>
    </a>
    ```
- Result:
```
<a href="http://www.kfupm.edu/~scott">
 Dr. Scott's Home Page</a>
```


Modularization

- Breaking up a complex program into simpler parts (is an important programming tool)
  - In programming languages modularization is often done with functions or methods
  - In XSL we can do something similar with `xsl:apply-templates`
- For example, suppose we have a DTD for `book` with parts `titlePage`, `tableOfContents`, `chapter`, and `index`
  - We can create separate templates for each of these parts
- Template rules are used to control what output is created from what input
```

…Modularization

- A template rule is represented by an `<xsl:template>` element
- The `<xsl:template>` element has
  - A `match` attribute that contains an XPath pattern identifying the input it matches
  - A template that is instantiated and output when the pattern is matched
- Template skeleton:

```
<xsl:template match="person">
     A Person
</xsl:template>
```

- The above says that every time a `<person>` element is seen, the stylesheet processor should emit the text "A Person"

Book example

```
<xsl:template match="/">
     <html> <body>
     <xsl:apply-templates/>
     </body> </html>
</xsl:template>


<xsl:template match="tableOfContents">
     <h1>Table of Contents</h1>
     <xsl:apply-templates select="chapterNumber"/>
     <xsl:apply-templates select="chapterName"/>
     <xsl:apply-templates select="pageNumber"/>
</xsl:template>
```

- Etc.

xsl:apply-templates

- The <xsl:apply-templates> element applies a template rule to the current element or to the current element's child nodes
- If we add a select attribute, it applies the template rule only to the child that matches
- If we have multiple <xsl:apply-templates> elements with select attributes, the child nodes are processed in the same order as the <xsl:apply-templates> elements

When templates are ignored

- Templates aren't used unless they are applied
  - Exception: Processing always starts with select="/"
  - If it didn't, nothing would ever happen
- If your templates are ignored, you probably forgot to apply them
- If you apply a template to an element that has child elements, templates are not automatically applied to those child elements

Applying templates to children

```
<book>
    <title>XML</title>
    <author>Gregory Brill</author>
</book>


<xsl:template match="/">
   <html> <head></head> <body>
      <b><xsl:value-of select="/book/title"/></b>
      <xsl:apply-templates select="/book/author"/>
   </body> </html>
</xsl:template>

<xsl:template match="/book/author">
   by <i><xsl:value-of select="."/></i>
</xsl:template>
```


Calling named templates

- You can name a template, then call it, similar to the way you would call a method in Java
- The named template:

```
<xsl:template name="myTemplateName">
        ...body of template...
</xsl:template>
```

- A call to the template:

```
<xsl:call-template name="myTemplateName"/>
```

- Or:

```
<xsl:call-template name="myTemplateName">
        ...parameters...
</xsl:call-template>
```

Templates with parameters

- Parameters, if present, are included in the content of `xsl:template`, but are the only content of `xsl:call-template`
- Example call:

```
<xsl:call-template name="doOneType">
  <xsl:with-param name="header" select="'Lectures'"/>
  <xsl:with-param name="nodes" select="//lecture"/>
</xsl:call-template>
```

- Example template:

```
<xsl:template name="doOneType">
    <xsl:param name="header"/>
    <xsl:param name="nodes"/>
    ...body of template...
</xsl:template>
```

- Parameters are matched up by name, not by position

XSL - On the Client

- If your browser supports XML, XSL can be used to transform the document to XHTML in your browser
- A JavaScript Solution
  - Even if this works fine, it is not always desirable to include a style sheet reference in an XML file (i.e. it will not work in a non XSL aware browser.)
  - A more versatile solution would be to use a JavaScript to do the XML to XHTML transformation
- By using JavaScript, we can:
  - do browser-specific testing
  - use different style sheets according to browser and user needs
- XSL transformation on the client side is bound to be a major part of the browsers work tasks in the future, as we will see a growth in the specialized browser market (Braille, aural browsers, Web printers, handheld devices, etc.)

Transforming XML to XHTML in Your Browser

```
<html>
<body>
<script type="text/javascript">

// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("books.xml")

// Load XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("books.xsl")

// Transform
document.write(xml.transformNode(xsl))

</script>
</body>
</html>
```


XSL - On the Server

- Since not all browsers support XML and XSL, one solution is to transform the XML to XHTML on the server
- To make XML data available to all kinds of browsers, we have to transform the XML document on the SERVER and send it as pure XHTML to the BROWSER
- That's another beauty of XSL! One of the design goals for XSL was to make it possible to transform data from one format to another on a server, returning readable data to all kinds of future browsers

Thoughts on XSL

- XSL is a programming language--and not a particularly simple one
  - Expect to spend considerable time debugging your XSL
- These slides have been an introduction to XSL and XSLT--there's a lot more of it we haven't covered
- As with any programming, it's a good idea to start simple and build it up incrementally: "Write a little, test a little"
  - This is especially a good idea for XSLT, because you don't get a lot of feedback about what went wrong
- Try jEdit with the XML plugin
  - write (or change) a line or two, check for syntax errors, then jump to IE and reload the XML file

Q & A

?

References

- Some useful links with examples and other resources:
  - *Internet and World Wide Web How to Program*, 4/e, H. M. Deitel, P. J. Deitel, and A. B. Goldberg, Pearson Education Inc., 2008.
  - W3C http://www.w3.org/TR/xsl/
  - W3School XSL Tutorial http://www.w3schools.com/xsl/default.asp
  - XSLT Tutorial http://www.zvon.org/xxl/XSLTutorial/Output/index.html
  - XSL & XSLT **http://www.dcs.bbk.ac.uk/~mick/academic/xml/dip/xsl.shtml**
  - MSXML 4.0 SDK
  - http://www.topxml.com
  - http://www.xml.org
  - http://www.xml.com