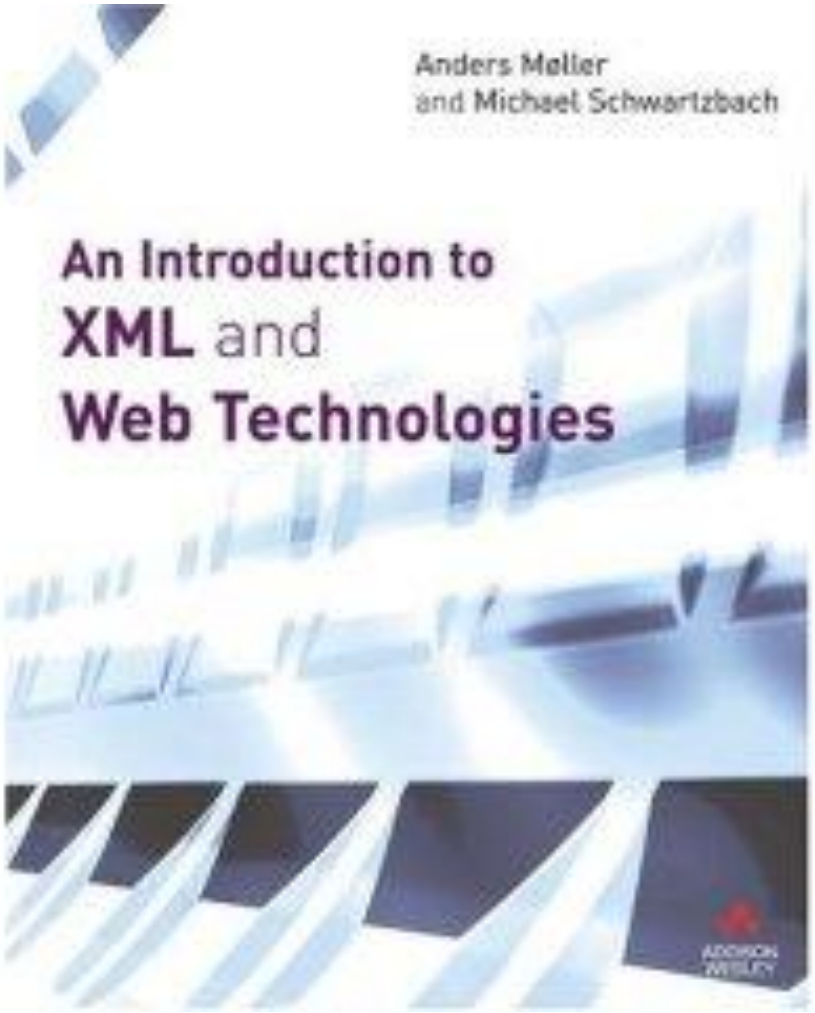
# XSLT (3) – Transforming XML documents

CS 431 – March 12, 2007

Carl Lagoze – Cornell University

Some slides borrowed from….



# Stylesheet Document or Program

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="para">
   <p>
      <xsl:apply-templates/>
   </p>
</xsl:template>
   <xsl:template match="emphasis">
      <p>
         <xsl:apply-templates/>
      </p>
   </xsl:template>
</xsl:stylesheet>
```

# XSL Execution Model

- Templates represent a set of rules
- Rule matching is done within current tree context
- Rules are not executed in order
- Precedence given to more specific rules
- Default behavior
  - Write element value
  - Reevaluate rules in new context resulting from depth-first tree step
  - Default behavior will ALWAYS happen unless overwritten by specific rule

# Template Rules

```
<xsl:template match="...">
   ...
</xsl:template>
```

- Find the template rules that *match* the context node
- Select the *most specific* one
- *Evaluate* the body (a *sequence constructor*)

# Sequence Constructors

- Element and attribute constructors
- Text constructors
- Copying nodes
- Recursive application
- Repetitions
- Conditionals
- Template invocation
- Variables and parameters

# Literal Constructors

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
     <html>
       <head>
         <title>Hello World</title>
       </head>
       <body bgcolor="green">
         <b>Hello World</b>
       </body>
     </html>
  </xsl:template>
</xsl:stylesheet>
```

# Explicit Constructors

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
     <xsl:element name="html">
        <xsl:element name="head">
          <xsl:element name="title">
            Hello World
          </xsl:element>
        </xsl:element>
        <xsl:element name="body">
          <xsl:attribute name="bgcolor" select="'green'"/>
          <xsl:element name="b">
            Hello World
          </xsl:element>
        </xsl:element>
     </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

# Computed Attributes Values (1/2)

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
     <xsl:element name="html">
        <xsl:element name="head">
          <xsl:element name="title">
            Hello World
          </xsl:element>
        </xsl:element>
        <xsl:element name="body">
          <xsl:attribute name="bgcolor" select="//@bgcolor"/>
          <xsl:element name="b">
            Hello World
          </xsl:element>
        </xsl:element>
     </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

# Computed Attribute Values (2/2)

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
     <html>
       <head>
         <title>Hello World</title>
       </head>
       <body bgcolor="{//@bgcolor}">
         <b>Hello World</b>
       </body>
     </html>
  </xsl:template>
</xsl:stylesheet>
```

# Text Constructors

- Literal text becomes character data in the output
- Whitespace control requires a constructor:

```
<xsl:text>2+2 = </xsl:text><xsl:value-of select="2+2"/>
```

- The value of an XPath expression:

```
<xsl:value-of select=".//@unit"/>
```

# Copying Nodes

- The `copy-of` element creates *deep* copies
- The `copy` element creates *shallow* copies
- Give top-most HTML lists square bullets:

```
<xsl:template match="ol|ul">
   <xsl:copy>
      <xsl:attribute name="style"
                     select="'list-style-type: square;'"/>
      <xsl:copy-of select="*"/>
   </xsl:copy>
</xsl:template>
```

# Recursive Application

- The `apply-templates` element
  - finds some nodes using the `select` attribute
  - applies the entire stylesheet to those nodes
  - concatenates the resulting sequences
- The default `select` value is `child::node()`

## Names, Modes, Priorities

- Templates may have several attributes

- `select:` used to change traversal on recursion (remember default is depth-first)
- `mode`: used to restrict the candidate templates
- `name`: used to call templates like function
- `priority`: used to determine specificity

# Explicit node selection

- http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/students.xsl
- http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/students.xml
-

# Mode setting

- <xsl:apply-templates mode="this">
- <xsl:template match="foo" mode="this">
- <xsl:template match="foo" mode="that">

- http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/modes.xsl

# Namespaces in XSLT

- The XSL document MUST know about the namespaces of elements that it references (via XPATH expressions) in the instance document
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/baseNS.xml
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/elementsNS.xsl
- Watch out for the default namespace!!
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/baseNoNS.xml
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/elementsNoNS.xsl

# Branching and repetitions programming

- for-each
- conditionals

# For-each programming example

- XML base file
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/foreach.xml
- XSLT file
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/foreach.xsl

# Conditionals

- If
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/if.xsl
- Choose
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/choose.xsl
  -

# Call-template programming example

- XML base file
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/call.xml
- XSLT file
  - http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/call.xsl

# Variables

- Scoped in normal fashion
  - Global
  - Within tree nesting level
- No static typing – take type of setting
  - string, number, boolean, node-set (set of nodes, sub-tree)

# Variables

- Initialization
  - `<xsl:variable name="age" select="25"/>`
  - Distinguish between string literal and xpath
    - `<xsl:variable name="city" select="'ithaca'"/>`
      - set variable to string "ithaca"
    - `<xsl:variable name="city" select="ithaca"/>`
      - set variable to result of xpath expression "ithaca"

# Variables

- Initialization
  - Construct temporary tree
    - <xsl:variable name="temptree">
      <foo><bar></bar></foo>
      </xsl:variable>
  - Usage – precede by '$'
    - <xsl:value-of select="$city"/>

## Variables (assignment)

- No assignment after initialization
- Think functional programming model (LISP, ML, Scheme)
- Use conditional initialization (<xsl:choose>)
- Use recursion rather than iteration for repetitive tasks

# Variables example

- http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/variables.xsl
- http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xslt/fib.xsl

# Inputs and outputs

- Inputs - Default is single input document
  - `Document('URL')` function returns root node of document at URL
- Outputs – Default is single XML document
  - `<xsl:output method={"xml" | "html" | "text"}/>` changes output format
  - `<xsl:document href="URL">`
    - Anywhere in xsl file changes the output destination.

# Associating an XML document with a transform

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://www.cs.cornell.edu/Courses/cs502/2002SP/Demos/xslt/simple.xsl"?>
<para>
  this is
  <emphasis>
    big
  </emphasis>
  text
</para>
```