

CSP2103-4102: Markup Languages

Lecture 5: Introduction to XSLT

Learning Outcomes

- Why you should transform XML data
- About the nature of transformation
- About XSL, XPath, and XSLT
- About the steps involved in a simple XSLT transformation
- How to perform two XSLT transformation

Working With XSL

- In 1998, the W3C developed the Extensible Style sheet Language, or XSL
- XSL is composed of three parts:
 - XSL-FO (Extensible Style sheet Language – Formatting Objects)
 - XSLT (Extensible Style sheet Language Transformations)
 - XPath

Introducing Xsl-fo, Xslt, And Xpath

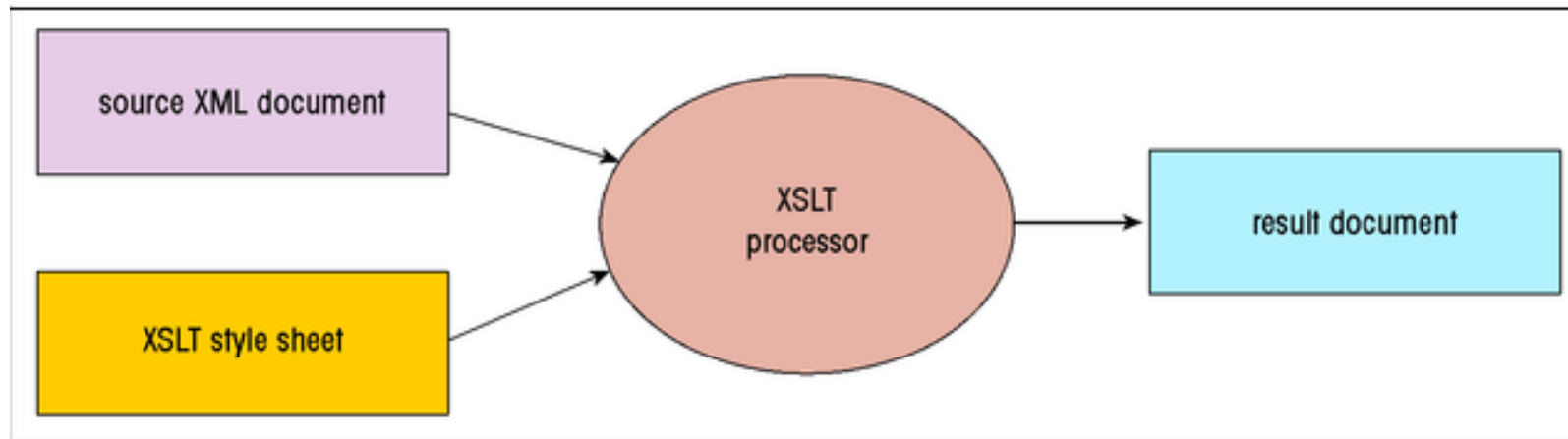
- XSLT is used to transform XML content into another presentation format (typically HTML)
- XPath is used to locate information from an XML document and perform operations and calculations upon that content
- XSL-FO is used to implement page layout and design (such as creating a formatted A4 page)

Introducing Xslt Style Sheets And Processors

- An XSLT style sheet contains instructions for transforming the contents of an XML document into another format
- An XSLT style sheet document is itself an XML document
- An XSLT style sheet document has an extension .xsl (though .xslt is also fine)

Generating A Result Document

- An XSLT style sheet converts a source document of XML content into a result document by using the XSLT processor



Introducing Xslt Style Sheets And Processors

- The transformation can be performed by a server or a client
- In a server-side transformation, the server receives a request from a client, applies the style sheet to the source document, and returns the result document to the client
- In a client-side transformation, a client requests retrieval of both the source document and the style sheet from the server, then performs the transformation, and generates the result document
- For a typical client-side only transformation, the web browser performs the transformation
- In a Windows environment this is can be done in IE 5.5+ or Mozilla Firefox 2.0 +

Creating An Xslt Style Sheet

- To create an XSLT style sheet, the general structure:

```
<?xml version =“1.0”>
```

```
<xsl:stylesheet version = 1.0
```

```
  xmlns:xsl =“http://www.w3.org/1999/XSL/Transform”>
```

Content of the style sheet

```
</xsl:stylesheet>
```

- The <xsl:stylesheet> tag can be substituted for the <xsl:transform> tag

Using Xpath To Reference A Node

- XPath provides the syntax to refer to the various nodes in an XML node tree
- The syntax is used by the operating system to specify file pathnames
- The location of a node can be expressed in either absolute or relative terms
- XPath also does data extraction

Relative Paths

- With a relative path, the location of the node is indicated relative to a specific node in the tree called the context node

PATH	DESCRIPTION
.	Refers to the context node
..	Refers to the parent of the context node
<i>child</i>	Refers to the child of the context node with the node name <i>child</i>
<i>child1/child2</i>	Refers to the <i>child2</i> node, a child of the <i>child1</i> node beneath the context node
<i>../sibling</i>	Refers to a sibling of the context node
<i>//name</i>	Refers to a descendant of the context node with the name <i>name</i>

Using Xpath To Reference A Node

- For absolute path, XPath begins with the root node, identified by a forward slash and proceeds down the levels of the node tree
- An absolute path: /child1/child2/child3/...
- To reference an element without regard to its location in the node tree, use a double forward slash with the name of the descendant node
- A relative path : //descendant

Referencing Groups Of Elements

- XPath allows you to refer to groups of nodes by using the wildcard character (*)
- To select all of the nodes in the node tree, you can use the path:
 - `//*`
- The (*) symbol matches any node, and the (//)symbol matches any level of the node tree
- **Example:** `/portfolio/stock/*`

Referencing Attribute Nodes

- XPath uses different notation to refer to attribute nodes
- The syntax for attribute node is:

@attribute

- where *attribute* is the name of the attribute
- For example, to reference the *artist* attribute of *album*

```
<music>
  <album artist="Metallica" title="Black">
    <tracks>
      .....
    </tracks>
  </album>
</music>
```

- Would be - `/music/album/@artist`

Working With Text Nodes

- The text contained in an element node is treated as a text node
- The syntax for selecting a text node is:
`@text()`
- To match all text nodes in the document, use:
`//text()`

Creating The Root Template

- A template is a collection of elements that define how a particular section of the source document should be transformed in the result document
- The root template sets up the initial code for the result document

Creating A Root Template

- To create a root template, the syntax is:

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

XSLT and Literal Result Elements

```
</xsl:template>
```

- This allows the XSLT processor to access all the elements and attributes in the XML document
- Typically, the best approach to XSLT development is to not place all code in the root template, but rather call a large number of named templates, each of which performs a specific task

Creating a Template

- To create a template, the syntax is:

```
<xsl:template match="node" name="whatever">
```

XSLT and Literal Result Elements

```
</xsl:template>
```

- where node is either the name of a node from the source document's node tree, or an XPath expression that points node in the tree
- The beauty of templates is that they can be named, and called in any order (almost like a procedure in normal programming languages)

Creating The Root Template

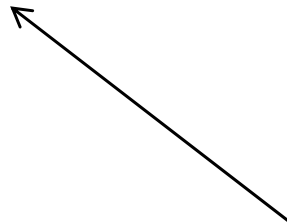
- A template contains two types of content: XSLT elements and literal result elements
 - XSLT elements are those elements that are part of the XSLT namespace and are used to send commands to the XSLT processor
 - A literal result element is text sent to the result document, but not acted upon by the XSLT processor

Creating The Root Template Example

```
<?xml version='1.0' ?>
<xsl:stylesheet version='1.0' xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <head>
  <title>Stock Information</title>
  <link href="stock.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
  <h1 class="title">Hardin Financial</h1>
  <h2 class="title">Stock Information</h2>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```



Notice that within the root template, we can just write normal xhtml if that is the document type we are creating

Specifying The Output Method

- By default, the XSLT processor will render the result document as an XML file
- To control how the processor formats the source document, you can specify the output method using the `<xsl:output/>` element

Attributes of the <Xsl:output/> Element

ATTRIBUTE	DESCRIPTION
method	Defines the output format using one of the following values: "xml," "html," or "text"
version	Specifies the version of the output
encoding	Specifies the character encoding
omit-xml-declaration	Specifies whether to omit an xml declaration in the first line of the result document ("yes") or to include it ("no")
standalone	Specifies whether a standalone attribute should be included in the output and sets its value ("yes" or "no")
doctype-public	Sets the URI for the public identifier in the <!DOCTYPE> declaration
doctype-system	Sets the system identifier in the <!DOCTYPE> declaration
cdata-section-elements	A list of element names whose content should be output in CDATA sections
indent	Specifies whether the output should be indented to better display its structure. Note that indentations are automatically added to HTML files
media-type	Sets the MIME type of output

Transforming A Document

- A browser with a built-in XSLT processor allows you to view the result document
- Alternatively, you can use XML Spy to create the result document as a separate file, and then view that file in your browser
- Most XSLT processors provide the capability to create the result document as a separate file

Viewing The Result Document In A Browser

- Internet Explorer 6.0 contains built-in XSLT processor
- You can view the results of the transformation by opening the result document in the browser
- When an XML document is parsed to XHTML using XSLT the transformation is typically dynamic, in that HTML tags and structure are sent to the browser
- The browser treats these commands as though there were coming from a physical HTML document
- XSLT can also be used to output to a physical .html file rather than just output HTML dynamically

Creating An Html File In Xml Spy

- One advantage of creating a separate HTML file is that it can be viewed in any Web browser, not just one that is XML / XSLT compliant
- You have to regenerate the HTML file every time you make a change to the source document, or the style sheet
- The XSLT processor adds one extra line to the document that provides additional information to the browser about the content of the document and its encoding

Inserting A Node Value

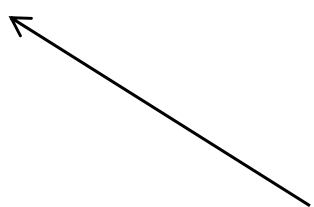
- To insert a node's value into the result document, the syntax is:

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

- where XPath Expression is an expression that identifies the node from the source document's node tree
- If the node contains child elements in addition to text content, the text in those child nodes appears as well

Inserting A Node Value Example

```
<xsl:template match="/">
<html>
<head>
<title>stock information</title>
<link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="datetime"><b>Last Updated: </b>
  <xsl:value-of select="portfolio/date" /> at
  <xsl:value-of select="portfolio/time" />
</div>
<h1 class="title">Hardin Financial</h1>
<h2 class="title">Stock Information</h2>
</body>
</html>
</xsl:template>
```



In this example we are seeing the integration of xslt and html

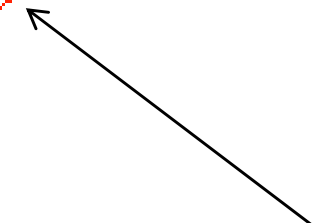
Processing A Batch Of Nodes

- To process a batch of nodes, the syntax is:
 <xsl:for-each select="XPath Expression" />
 XSLT and Literal Elements
 </xsl:for-each>
- where XPath Expression is an expression that defines the group of nodes to which the XSLT and literal result elements are applied
- This structure will loop through a set of elements
- A for-each loop can contain other for-each loops for multi-level XML element structures

Processing A Batch Of Nodes

Example

```
<xsl:template match="/">
<html>
<head>
<title>Stock Information</title>
<link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="datetime"><b>Last Updated: </b>
  <xsl:value-of select="portfolio/date" /> at
  <xsl:value-of select="portfolio/time" />
</div>
<h1 class="title">Hardin Financial</h1>
<h2 class="title">stock information</h2>
<xsl:for-each select="portfolio/stock">
<h3 class="name">
  <xsl:value-of select="name" />
</h3>
</xsl:for-each>
</body>
</html>
</xsl:template>
```



Within the for-each loop normal html can be written as well as xslt referencing xml elements

Working With Templates

- To apply a template in the result document, use the XSLT element

`<xsl:apply-templates select="XPath Expression" />`

- where XPath Expression indicates the node template to be applied

Creating Template Example

```
<xsl:template match="/">
<html>
<head>
<title>stock information</title>
<link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="datetime"><b>Last Updated: </b>
  <xsl:apply-templates select="portfolio/date" /> at
  <xsl:value-of select="portfolio/time" />
</div>
<h1 class="title">Hardin Financial</h1>
<h2 class="title">Stock Information</h2>
<xsl:apply-templates select="portfolio/stock/name" />
</body>
</html>
</xsl:template>

<xsl:template match="name">
<h3 class="name">
  <xsl:value-of select="." />
</h3>
</xsl:template>

</xsl:stylesheet>
```



In this case the template is called at the place it is needed within the root template

Using The Built-in Templates

- Each node has its own built-in template.
- The built-in template for element nodes matches the document root and all elements in the node tree
- The built-in template for text nodes matches all text nodes and causes their values to appear in the result document
- For example, you can add the stock template to the style sheet

Creating The Stock Template Example

```
<xsl:template match="/">
<html>
<head>
<title>stock information</title>
<link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="datetime"><b>Last Updated: </b>
  <xsl:apply-templates select="portfolio/date" /> at
  <xsl:value-of select="portfolio/time" />
</div>
<h1 class="title">Hardin Financial</h1>
<h2 class="title">Stock Information</h2>
<xsl:apply-templates select="portfolio/stock" />
</body>
</html>
</xsl:template>

<xsl:template match="stock">
<div class="stock_info">
  <xsl:apply-templates select="name" />
  <p><xsl:value-of select="description" /></p>
</div>
</xsl:template>
```


Named Templates

- Templates can be assigned names
- Templates can be called by those names

```
<xsl:call-template name="templatename">  
  
  <!-- Content:xsl:with-param* -->  
  
</xsl:call-template>
```

W3schools.com

- The named templates can be called at any time, and can also have parameters passed to them, such as search or search criteria

Sorting Nodes

- By default, nodes are processed in document order, by their appearance in the document
- To specify a different order, XSLT provides the `<xsl:sort>` element
- This element can be used with either the `<xsl:apply-templates>` or the `<xsl:for-each>` element
- This is why the structure of an XML document is important, for if you wish to sort data by say *surname*, you need `<firstname>` and `<surname>` elements rather than single `<name>` element

Sorting Nodes

- The `<xsl:sort>` element contains several attributes to control how the XSLT process sorts the nodes in the source document
 - The `select` attribute determines the criteria under which the context node is sorted
 - The `data-type` attribute indicates the type of data (numeral or text)
 - The `order` attribute indicates the direction of the sorting (ascending or descending)
- ```
<xsl:apply-templates select="day".
 <xsl:sort data-type="number" order="descending" />
</xsl:apply-templates>
```

# Creating Conditional Nodes

- XSLT supports two kinds of conditional elements:
  - `<xsl:if>`
  - `<xsl:choose>`
- To apply a format only if a particular condition is met , use the `<xsl:if>` element
- To test for multiple conditions and display different outcomes, use the `<xsl:choose>` element (*similar to a elseif structure in procedural languages*)

# Creating Conditional Nodes Example

```
<xsl:template match="today">
<table class="today">
<tr>
 <th class="today">Current</th>
 <th class="today">Open</th>
 <th class="today">High</th>
 <th class="today">Low</th>
 <th class="today">Volume</th>
</tr>
<tr>
 <td class="number">
 <xsl:choose>
 <xsl:when test="@current < @open">

 </xsl:when>
 <xsl:when test="@current > @open">

 </xsl:when>
 <xsl:otherwise>

 </xsl:otherwise>
 </xsl:choose>
 <xsl:value-of select="@current" />
 </td>
 <td class="number"><xsl:value-of select="@open" /></td>
 <td class="number"><xsl:value-of select="@high" /></td>
 <td class="number"><xsl:value-of select="@low" /></td>
 <td class="number"><xsl:value-of select="@vol" /></td>
</tr>
</table>
</xsl:template>
```

# Using Comparison Operators And Functions

OPERATOR	DESCRIPTION	EXAMPLE
=	Tests whether two values are equal to each other	@symbol = "AA"
!=	Tests whether two values are unequal	@symbol != "AA"
&lt;	Tests whether one value is less than another	day &lt; 5
&lt;=	Tests whether one value is less than or equal to another	day &lt;= 5
>	Tests whether one value is greater than another	day > 1
>=	Tests whether one value is greater than or equal to another	day >= 1

# Working With Predicates

- Predicates are XPath expressions that test for a condition and create subsets of nodes that fulfill that condition
- The predicate can also indicate the position of the node in the node tree
- To select a specific position in the source document, use the position() function combined with any XPath expression
- Predicates can also be comprised of conditions or calculations, such as Age>17

# Adding Predicates To The Root Template Example

```
<xsl:template match="/">
<html>
<head>
<title>Stock Information</title>
<link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="datetime">Last Updated:
 <xsl:apply-templates select="portfolio/date" /> at
 <xsl:value-of select="portfolio/time" />
</div>
<h1 class="title">Hardin Financial</h1>
<h2 class="title">Stock Information</h2>
<h2 class="category">Industrials</h2>
<xsl:apply-templates select="portfolio/stock[category='Industrials']">
 <xsl:sort select="name" />
</xsl:apply-templates>

<h2 class="category">Utilities</h2>
<xsl:apply-templates select="portfolio/stock[category='Utilities']">
 <xsl:sort select="name" />
</xsl:apply-templates>

<h2 class="category">Transportation</h2>
<xsl:apply-templates select="portfolio/stock[category='Transportation']">
 <xsl:sort select="name" />
</xsl:apply-templates>
</body>
</html>
</xsl:template>
```



# Creating Elements And Attributes

- To create an element, XSLT uses the `<xsl:element>` tag
- The namespace attribute assigns a name to the element
- The namespace attribute provides a namespace
- The use-attribute provides a list of attribute-sets
- Essentially you can build up elements and their attributes from scratch

# Creating An Element

- To create the <a> element in the result document, use the <xsl:element> tag

```
<xsl:template match="name">
 <xsl:element name="a">
 <h3 class="name">
 <xsl:value-of select="." />
 (<xsl:value-of select="@symbol" />)
 </h3>
 </xsl:element>
</xsl:template>
```

# Creating An Attribute

- Attributes are created in XSLT by using the `<xsl:attribute>` element
- The name attribute specifies the name of the attribute
- The namespace attribute indicates the namespace
- You can create inline images in the result document by using the attribute tag

# Creating An Attribute

- To add the href attribute to the <a> tag, use the <xsl:attribute> element for building a hyperlink

```
<xsl:template match="name">
 <xsl:element name="a">
 <xsl:attribute name="href">
 <xsl:value-of select=" ../link" />
 </xsl:attribute>
 <h3 class="name">
 <xsl:value-of select="." />
 (<xsl:value-of select="@symbol" />)
 </h3>
 </xsl:element>
</xsl:template>
```

# Creating Comments And Processing Instructions

- The `<xsl:comment>` element creates the comment
- You can create a processing instruction by using the `<xsl:processing-instruction>` element
- If you want to add a processing instruction to attach the result document to the style.css sheet, use the following code:

```
<xsl:processing-instruction name="xml-style sheet"
 href="styles.css" type="text/css" />
```

# Summary

- Extensible Style sheet Language, or XSL, is composed of three parts: XSL-FO, XSLT, and XPath
- XPath language is used to reference a node
- Templates are used to format sections of the XML document and transform XML data into a variety of formats, using rules and conditions

# Summary

- Nodes can be sorted in either alphabetical or numerical order
- Comparison elements allow changing the contents of the result document based on the values of the nodes in the source document
- Predicates are used to create subsets of the source document's node tree
- You can insert new elements and attributes in the transformed document
- These technologies effectively give the developer server-side processing capability on the client-side (almost)