<440: XSLT>

Jianguo Lu
University of Windsor

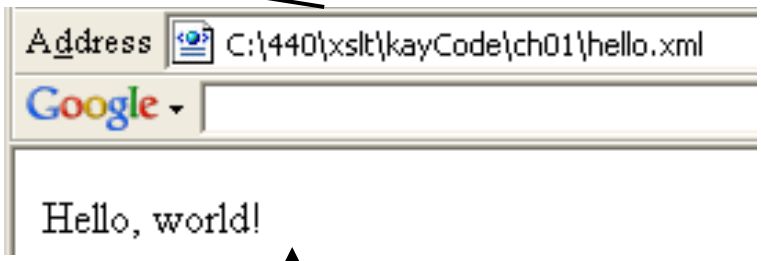
XML based programming

- With the widespread adoption of XML standard, there are languages designed specifically for XML processing

- XML query
  - XQuery
- XML transformation
  - XSLT
  - relies on XPath and XML Schema
- Web Service

Hello world example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<greeting>Hello, world!</greeting>
```




```
<?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>Today's greeting</title>
  </head>
  <body>
    <p><xsl:value-of select="greeting"/></p>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

```
<html>
<head>
<META http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
<title>Today's greeting</title>
</head>
<body>
<p>Hello, world!</p>
</body>
</html>
```

Running XSLT from the client side

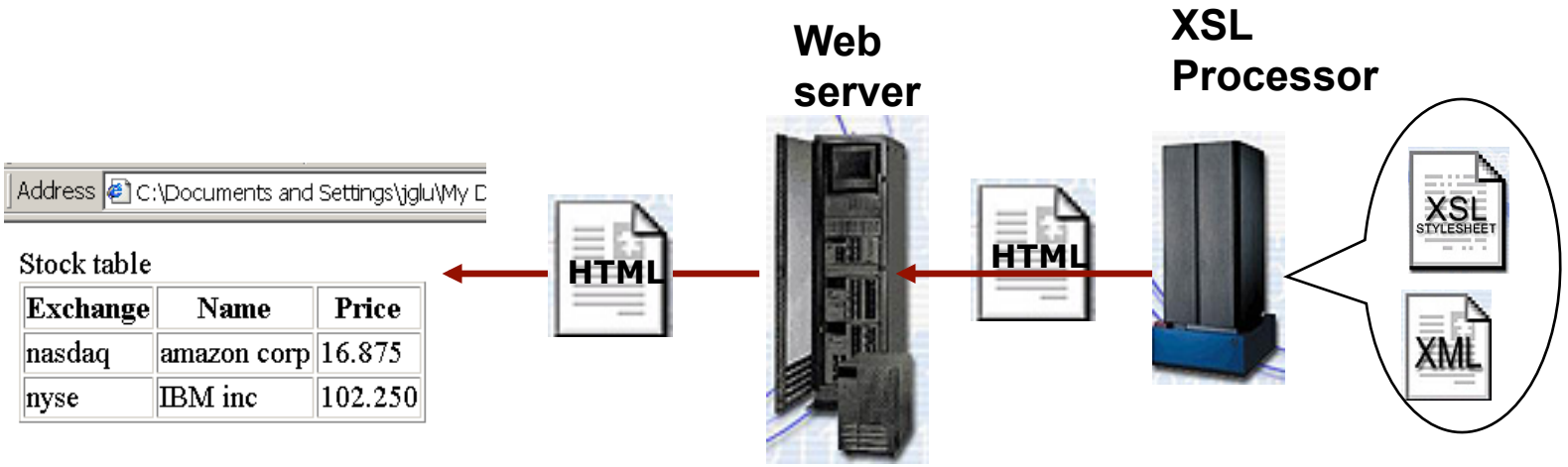
- Browser gets the XML+XSLT, and interprets them inside the browser.

- How to specify the XSL associated with the XML file?
  - <?xml-stylesheet type="text/xsl" href="stock.xsl"?>
- Advantages:
  - Easy to develop and deploy.
- Disadvantages:
  - Not every browser supports XML+XSL;
  - Browsers do not support all XSLT features;
  - Not secure: you only want to show part of the XML data;
  - Not efficient.

Run XSLT from the server side

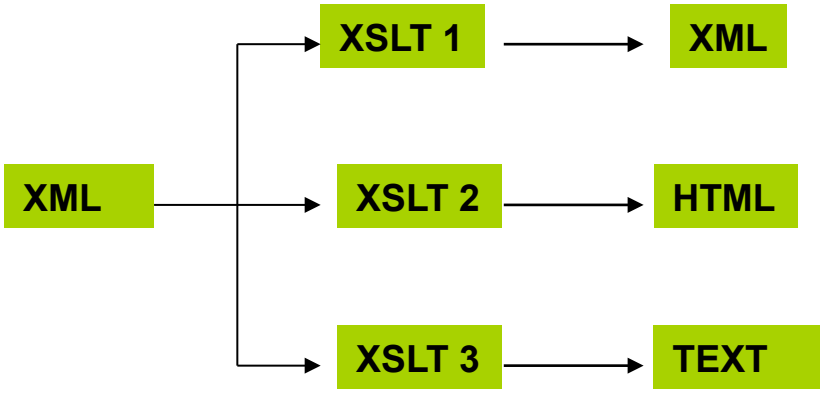
- XSL processor transforms the XML and XSLT to HTML, and the web server send the HTML to the browser.
- Popular tool: xalan
- Download xalan.jar from
  - http://www.apache.org/dyn/closer.cgi/xml/xalan-j
- Run Xalan to transform xml documents:

  java -classpath xalan/bin/xalan.jar org.apache.xalan.xslt.Process -in stock.xml -xsl stock.xsl -out stock.html



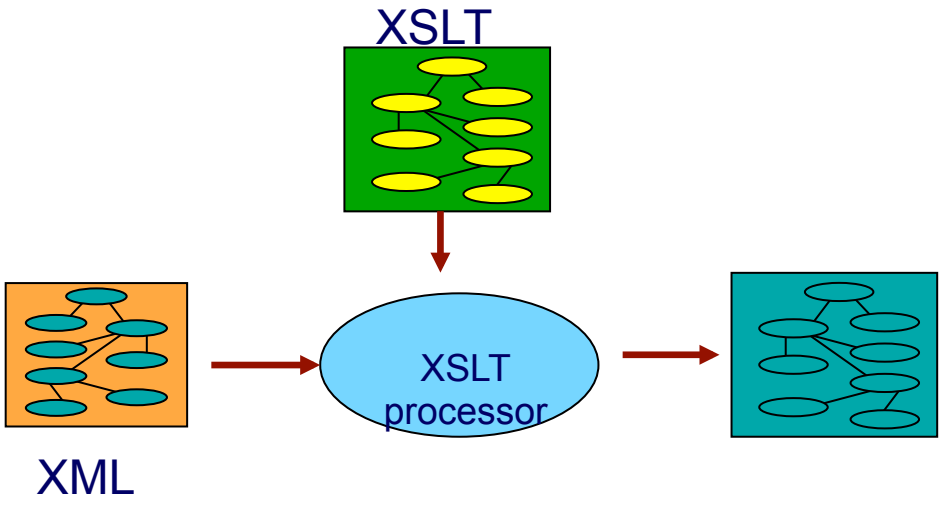
XSLT(XML Stylesheet Language Transformation)

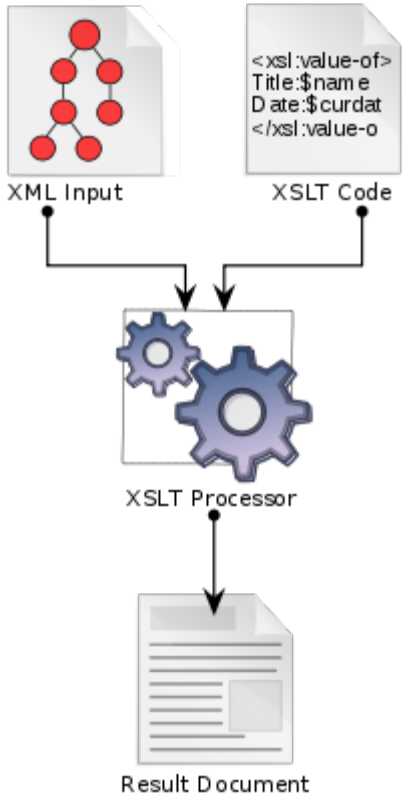
- XSLT Version 1.0 is a W3C Recommendation, 1999
  - http://www.w3.org/Style/XSL/
- XSLT is used to transform XML to other formats.



XSLT basics

- XSLT is an XML document itself
- It is a tree transformation language





- It is a rule-based declarative language
  - XSLT program consists of a sequence of rules.
  - It is a functional programming language.
    - It has no side-effects

Poem example: xml and intended html output

```
<poem>
   <author>Rupert Brooke</author>
   <date>1912</date>
   <title>Song</title>
  <stanza>
      <line>And suddenly the wind comes soft,</line>
      <line>And Spring is here again;</line>
      <line>And the hawthorn quickens with buds of green</line>
      <line>And my heart with buds of pain.</line>
   </stanza>
   <stanza>
      <line>My heart all Winter lay so numb,</line>
      <line>The earth so dead and frore,</line>
      <line>That I never thought the Spring would come again</line>
      <line>Or my heart wake any more.</line>
   </stanza>
   <stanza>
      <line>But Winter's broken and earth has woken,</line>
      <line>And the small birds cry again;</line>
      <line>And the hawthorn hedge puts forth its buds,</line>
      <line>And my heart puts forth its pain.</line>
   </stanza>
</poem>
```

Song

By Rupert Brooke

And suddenly the wind comes soft,
And Spring is here again;
And the hawthorn quickens with buds of green
And my heart with buds of pain.

My heart all Winter lay so numb,
The earth so dead and frore,
That I never thought the Spring would come again
Or my heart wake any more.

But Winter's broken and earth has woken,
And the small birds cry again;
And the hawthorn hedge puts forth its buds,
And my heart puts forth its pain.

1912

Poem example: XSL

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

<xsl:template match="poem">
  <html>
  <head>
    <title><xsl:value-of select="title"/> </title>
  </head>
  <body>
    <xsl:apply-templates select="title"/>
    <xsl:apply-templates select="author"/>
    <xsl:apply-templates select="stanza"/>
    <xsl:apply-templates select="date"/>
  </body>
  </html>
</xsl:template>
```

```
<xsl:template match="title">
  <div align="center">
    <h1><xsl:value-of select="."/> </h1>
  </div>
</xsl:template>

<xsl:template match="author">
  <div align="center">
    <h2>By <xsl:value-of select="."/> </h2>
  </div>
</xsl:template>

<xsl:template match="date">
  <p><i><xsl:value-of select="."/> </i></p>
</xsl:template>

<xsl:template match="stanza">
 <p><xsl:apply-templates select="line"/> </p>
</xsl:template>

<xsl:template match="line">
  <xsl:if test="position() mod 2 = 0">    </
     xsl:if>
  <xsl:value-of select="."/> <br/>
</xsl:template>
</xsl:stylesheet>
```

Generate the online test questions

```
<test>
<Q a="E">
<question>
Which of the following is not an OO programming language?
</question>
<item>  Java
</item><item> C++
      </item><item> Smalltalk
      </item><item> Simula 67
      </item><item> None of the above.
</item></Q>


<Q a="A"><question>Which of the following language is not a
      declarative language?
</question><item>
 Java
</item><item> Prolog
</item><item> SQL
</item><item> Scheme
</item><item> None of the above.

</item></Q>
```

- Which of the following is not an OO programming language?

  A. Java
  B. C++
  C. Smalltalk
  D. Simula 67
  E. None of the above.

- Which of the following language is not a declarative language?

  A. Java
  B. Prolog
  C. SQL
  D. Scheme
  E. None of the above.

- Which of the following is a script language?

  A. Java
  B. C#
  C. PHP
  D. Prolog
  E. Scheme
  F. None of the above.

Rule-based

- XSLT consists of a sequence of template rules;
- Each describes how a particular element type (or other constructs) can be processed;
- Rules are not arranged in any particular order (theoretically)
  - They don't have to match the order of input or output;
  - Programmers don't know (and don't care) the execution ordering of the rules;
  - That is why it is called a declarative language;
  - Programmers specify what output should produce when particular pattern occurs in input.

```
<xsl:template match="title">
  <div align="center">
    <h1><xsl:value-of select="."/
    > </h1>
  </div>
</xsl:template>

<xsl:template match="author">
  <div align="center">
    <h2>By <xsl:value-of
    select="."/> </h2>
  </div>
</xsl:template>

<xsl:template match="date">
  <p><i><xsl:value-of select="."/
    > </i></p>
</xsl:template>

… …
```

No side effect

- If functions have side effects, it is important to call them the right number of times, in a correct order, and in a correct context.
- Why XSLT does not want to have side effects:
  - Requirement from the rule-based language
  - Incremental development
  - Incremental rendering
    - Browser display partially downloaded xml file

XSLT process model

```
<source>
<employee>
   <firstName>Joe</firstName>
   <surName>Smith</surName>
</employee>

<employee>
   <firstName>Andrew</firstName>
   <surName>Wang</surName>
   <supervisor>
    <employee>
     <firstName>Steve</firstName>
     <surName>Miller</surName>
    </employee>
   </supervisor>
</employee>
</source>
```

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform"
        version="1.0">

 <xsl:template match="employee">
Hello, Found you!
 </xsl:template>

 </xsl:stylesheet>
```

- Output:

Hello, Found you!

Hello, Found you!

- Questions:
  - Why only two matches?
  - What are the two matches?

Built-in template

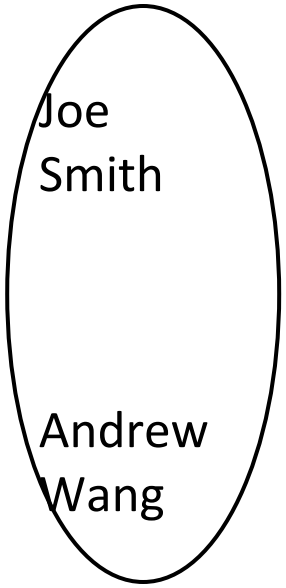
- Select the employee who is a supervisor (incorrect)

```
<xsl:template match="employee/supervisor/
    employee">
Hello, found you!
 </xsl:template>
```

- XSLT starts with the root of the document.
- If there is no template rule defined, built-in template is invoked.
  - For element, the built-in template processes the children of the current element.
  - For text, the built-in template copy the text to the result tree.

Outout:

```
<?xml version="1.0"
    encoding="UTF-8"
    ?>
```



Hello, found you!

Text node XML Tree

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/
    1999/XSL/Transform" version="1.0">

 <xsl:template match="surName">
  sur name is <xsl:value-of select="." />
</xsl:template>
</xsl:stylesheet>
```

- Output

Joe

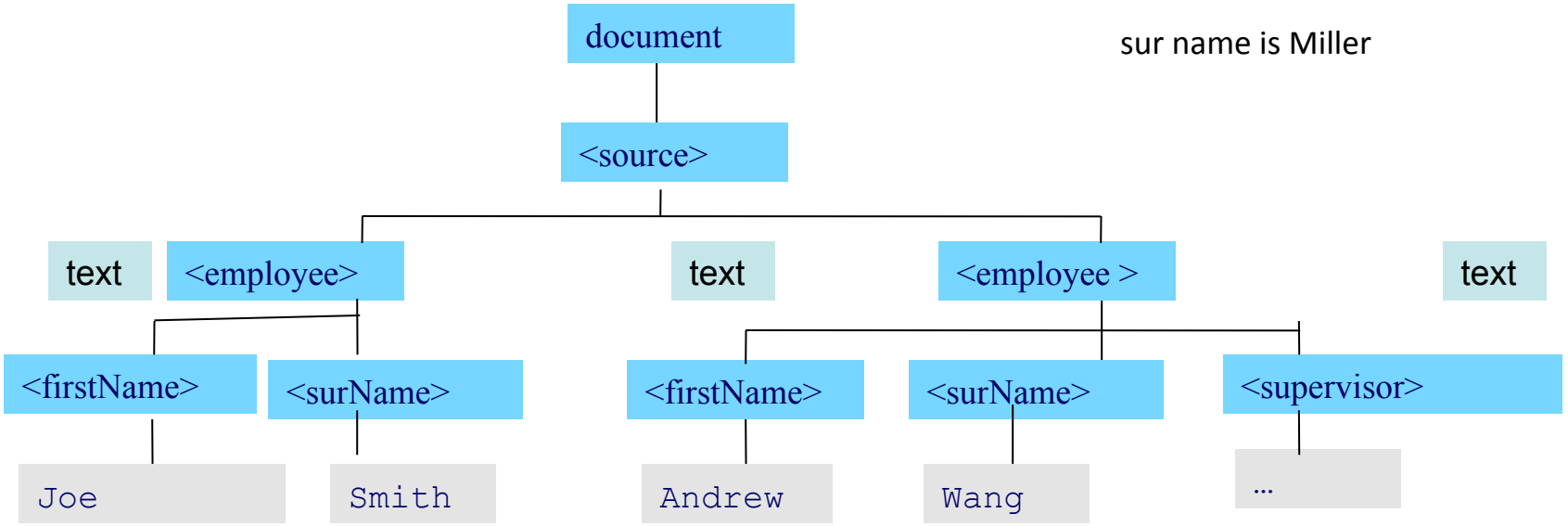
sur name is Smith

Andrew

sur name is Wang

Steve

sur name is Miller



Override built-in template

- Pick employees who are supervisors (still not correct)

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

<xsl:template match="employee/supervisor/employee">
Hello, found you!
  </xsl:template>
</xsl:stylesheet>
```

Output:

nothing.

Override built-in template

- When find outer employees, apply rules

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

  <xsl:template match="employee/supervisor/
     employee">
Hello, found you!
  </xsl:template>
</xsl:stylesheet>
```

Output:

Joe

Smith

Andrew

Wang

Hello, found you!

More specific about what to do

```
<xsl:template match="employee">
  <xsl:apply-templates select="supervisor/>
</xsl:template>

  <xsl:template match="employee/supervisor/
     employee">
Hello, found you!
  </xsl:template>
</xsl:stylesheet>
```

Output:

Hello, found you!

XML process model: why built-in rules

```
<?xml version='1.0'?>
<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">
 <i><xsl:apply-templates/></i>
</xsl:template>
 </xsl:stylesheet>
```

- Output copies most part of the input
- It would be cumbersome to write each of those built-in rules explicitly

```
<?xml version='1.0'?> <para>
    This is a <emphasis> test
    </emphasis></para> .
```

- Output

```
<p> This is a <i> test </i></p>.
```

The input

```
<source>
<employee>
    <firstName>Joe</firstName>
    <surName>Smith</surName>
</employee>

<employee>
    <firstName>Andrew</firstName>
    <surName>Wang</surName>
    <supervisor>
      <employee>
        <firstName>Steve</firstName>
        <surName>Miller</surName>
      </employee>
    </supervisor>
</employee>
</source>
```

The order of rules

```
<xsl:template match="employee">
  <tr> <xsl:apply-templates /> </tr>
</xsl:template>

<xsl:template match="surName">
 <td> <xsl:value-of select="."/> </td>
</xsl:template>

<xsl:template match="firstName">
 <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

- Input data decides the rules to be applied.
- It works well if the output has the same structure and sequence as the input;
- All we did is performing simple editing of values as we go along.
- What if we want sur name printed first?

Result:

```
<tr>
   <td>Joe</td>
   <td>Smith</td>
</tr>

<tr>
   <td>Andrew</td>
   <td>Wang</td>

    <tr>
     <td>Steve</td>
     <td>Miller</td>
    </tr>

</tr>
```

Controlling which rule to apply first

- Be more precise about which node to process, rather than just saying process all children of the current node
- Be more precise about how to process them, rather than just saying choose the best-fit template rule

```
<xsl:template match="employee">
  <tr> <xsl:apply-templates select="surName" />
    <xsl:apply-templates select="firstName" />
  </tr>
</xsl:template>

<xsl:template match="surName | firstName">
 <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

- Instead of selecting all child elements and finding the appropriate template rule for each one, this one explicitly selects <surName> and <firstName> elements
- Other child elements are not processed.
  - <supervisor> is not processed.
- Output

```
<tr><td>Smith</td><td>Joe</td></tr>

<tr><td>Wang</td><td>Andrew</td></tr>
```

Resolve conflict

```
<xsl:template match="employee">
  doing this
 </xsl:template>

  <xsl:template match="source/
   employee">
  doing that
  </xsl:template>
```

In this case there are two rules matching with employee elements.

Which rule should be used?

- Output

  doing that

  doing that

- Priority rules
  - "source/employee" gets higher priority than "employee";
  - "employee" gets higher priority than "*";
  - * matches everything. We will see that later.

Another priority example

```
<xsl:template match="doc">
 <html><head><title>A Document</title></head>
 <body><xsl:apply-templates/></body></html>
</xsl:template>

<xsl:template match="para">
 <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="emphasis">
 <i><xsl:apply-templates/></i>
</xsl:template>

<xsl:template match="emphasis/emphasis">
 <b><xsl:apply-templates/></b>
</xsl:template>
</xsl:stylesheet>
```

- Why nested emphasis is not using the rule adding <i> ?

```
<doc>
<para>This is a <emphasis>test</emphasis>.
  <emphasis>Nested      <emphasis>emphasis</
    emphasis>
  </emphasis>.
</para>
</doc>
```

```
<html>
<head>
<title>A Document</title>
</head>
<body>
<p>This is a <i>test</i>.
<i>Nested <b>emphasis</b></i>.</p>
</body>
</html>
```

This is a *test*. *Nested* ***emphasis***.

Pattern matching vs. <for-each>

- Pattern matching style of processing is the most characteristic feature of XSLT;
- Good for
  - applications where each type of node in the document is handled independently;
  - processing an element that may contain children of different types in an unpredictable sequence.
  - documents that are likely to evolve over time.
    - As new child elements are added, template rules to process them can be added, without changing the logic for the parent element.
- There are alternatives
  - <xsl:for-each> is closer to procedure programming
  - Perform explicit processing for each node

<for-each> and XPath

- Select sequence of items using XPath expression, performs same processing for each item in the sequence.

```
<xsl:template match="source">
 <xsl:for-each select="employee">
        <tr> <xsl:apply-templates select="surName"/></tr>
 </xsl:for-each>
</xsl:template>

<xsl:template match="surName">
 <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

- "employee" is an XPath.
- "source", "surName", "." are also xpaths.
- It selects nodes relative to the context node (the node currently being processed).
- Output

```
<tr><td>Smith</td></tr><tr><td>Wang</td></tr>
```

XPath

- XPath select nodes relative to its context

```
<xsl:template match="source">
  <xsl:for-each select="employee/supervisor/employee">
        <tr> <xsl:apply-templates select="surName"/></tr>
  </xsl:for-each>
</xsl:template>

<xsl:template match="surName">
  <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

- Output
  - <tr><td>Miller</td></tr>

Selecting more than one case

```
<xsl:template match="source">
  <xsl:for-each select="employee/supervisor/employee | employee">
    <tr> <xsl:apply-templates select="surName"/>
          <xsl:apply-templates select="firstName"/>
     </tr>
   </xsl:for-each>
</xsl:template>

<xsl:template match="surName | firstName">
  <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

- Output

```
<tr><td>Smith</td><td>Joe</td></tr>
<tr><td>Wang</td><td>Andrew</td></tr>
<tr><td>Miller</td><td>Steve</td></tr>
```

- Can we have more concise notation than “employee/supervisor/employee | employee" ?
- In general, simply writing the element names are not enough. What is the syntax (or the XPath language)?

XPath language

- XPath is a sublanguage in XSLT.
- For example, previous example can be simplified as follows, where "//employee" is an XPath expression, meaning any employee element, no matter where it occurs.

```
<xsl:template match="source">
  <xsl:for-each select= "//employee">
   <tr> <xsl:apply-templates select="surName"/>
   <xsl:apply-templates select="firstName"/>
   </tr>
</xsl:for-each>
</xsl:template>

<xsl:template match="surName | firstName">
  <td> <xsl:value-of select="."/> </td>
</xsl:template>
```

XPath

- One critical capability of a stylesheet language is to locate source elements to be styled.
- XPath has a *string-based* syntax (it is no longer in XML format)
- It describes "location paths" between parts of a document or documents
- XPath defines a library of standard functions
  - Such as arithmetic expressions.
- XPath is a major element in many xml based languages, such as XSLT and XML query languages
- XPath is a W3C Standard
- One inspiration for XPath was the common "path/file" file system syntax

What is XPath

- Like traditional file paths

  public_html/440/xslt.ppt

- XPath uses path expressions to identify nodes in an XML document.

  source/employee

- Absolute path
  - /source/employee
- Relative path
  - employee

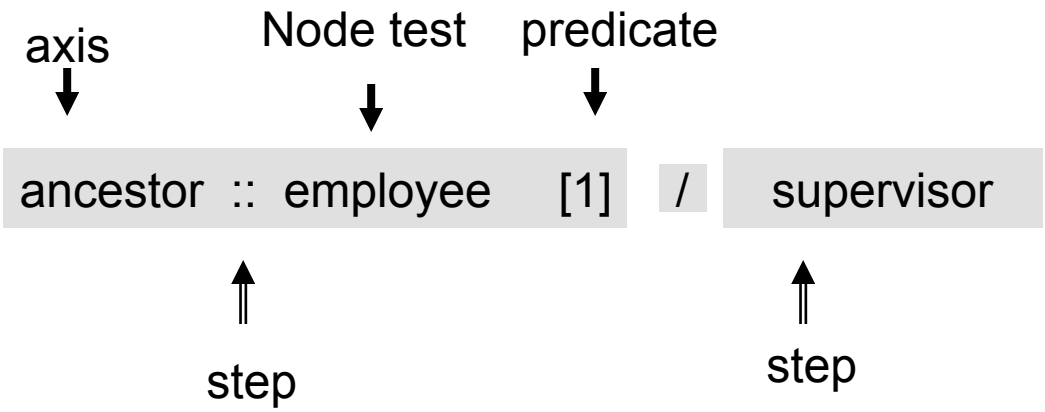
XPath pattern syntax

A path consists of a series of *steps*, separated by slashes

- E.g., "`employee/supervisor`" has two steps

– A step can be an *axis* specifier, followed by a node *test*, and a *predicate*

- "`employee`" in our previous example is a node test.
- "`employee[1]`" is an node test followed by a predicate
- "`ancestor::employee`" contains an axis followed by a node test



Predicate in XPath

```
<xsl:template match="source">
  <xsl:for-each select="employee[1]">
    <xsl:value-of select="firstName"/>
  </xsl:for-each>
</xsl:template>
```

- "[1]" is a predicate, matching the first element
- Output
  - Joe

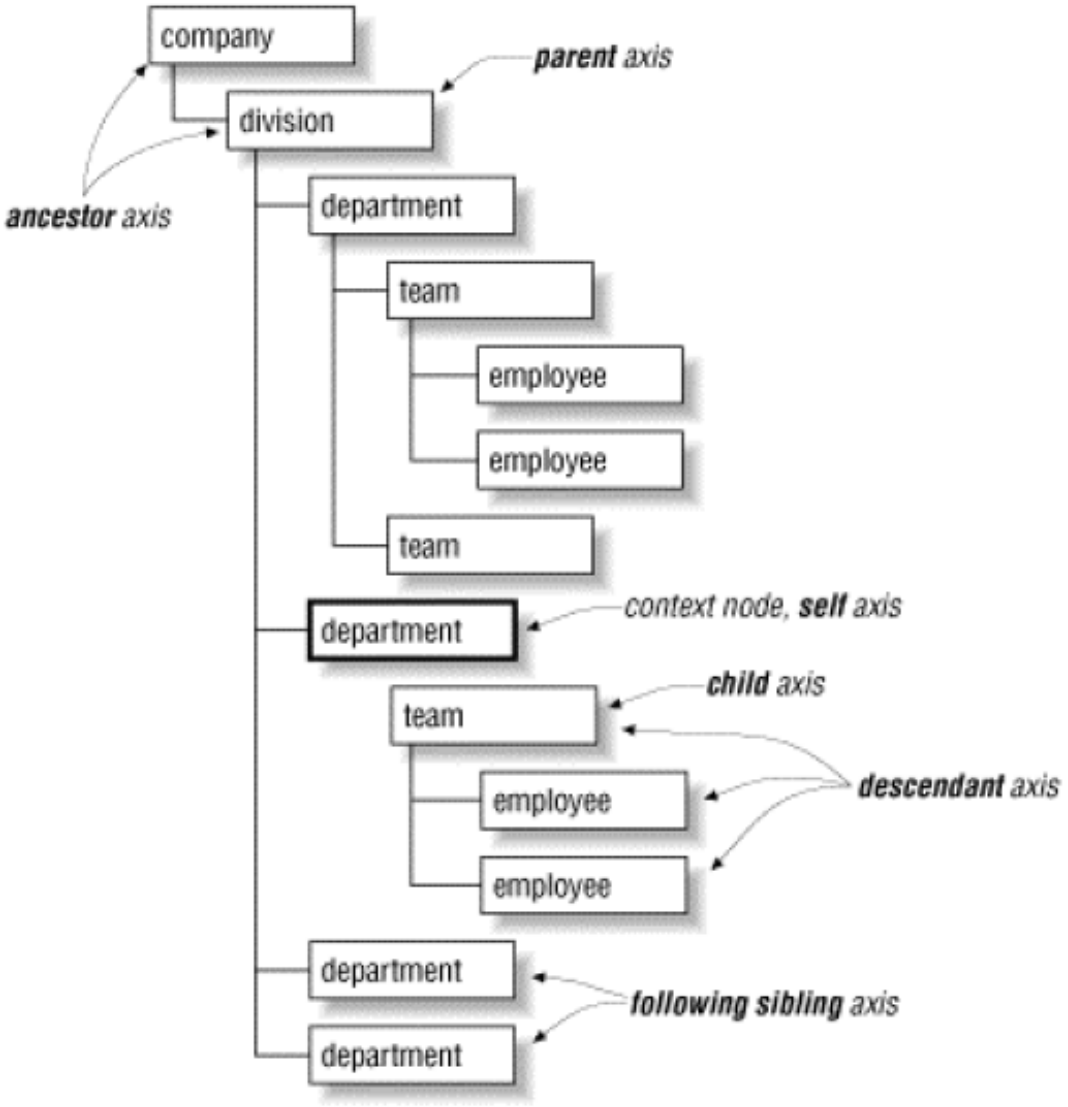
- Predicate can involve xslt library functions

```
<xsl:for-each select="employee[position()=last()]">
```

- Output
  - Andrew
- <xsl:for-each select="employee[position() mod 2 = 0]">
  - match even nodes
  - Recall the poem example uses similar predicate

Axes

- Ancestor
- Parent
- Child
- Descendant
- Sibling
- Self



Axis in XPath

```
<xsl:template match="supervisor/employee">
    ancestor node of <xsl:value-of select="firstName"/> is
    <xsl:value-of select="ancestor::employee/firstName"/>
 </xsl:template>
<xsl:template match="employee">
  <xsl:apply-templates select="supervisor/employee"/>
</xsl:template>
```

- ancestor:: is the axis specifier. It matches with ancestor employee elements
- Output:
  - ancestor node of Steve is Andrew

XPath examples

- “para”
  - Matches all <para> children in the current context
- “para/emphasis”
  - Matches all <emphasis> elements that have a parent of <para>
- “emphasis | strong”
  - Matches <emphasis> or elements
- “/”
  - Matches the root of the document
- “para//emphasis”
  - Matches all <emphasis> elements that have an ancestor of <para>
- “section/para[1]”
  - Matches the first <para> child of all the <section> children in the current context
- “//title”
  - Matches all <title> elements anywhere in the document
- “.//title”
  - Matches all <title> elements that are descendants of the current context

Access attributes

```
<library location="Bremen">
    <author name="Henry Wise">
      <book title="Artificial Intelligence"/>
      <book title="Modern Web Services"/>
      <book title="Theory of Computation"/>
    </author>
    <author name="William Smart">
      <book title="Artificial Intelligence"/>
    </author>
    <author name="Cynthia Singleton">
      <book title="The Semantic Web"/>
      <book title="Browser Technology Revised"/>
    </author>
</library>
```

/library

/library/author

//author

/library/@location

//
book[@title="Artificial Intelligence"]

XPath examples

- “section/*/note”
  - Matches <note> elements that have <section> grandparents.
- “stockquote[@symbol]”
  - Matches <stockquote> elements that have a "symbol" attribute
- “stockquote[@symbol=appl]”
  - Matches <stockquote> elements that have a "symbol" attribute with the value ”appl“
- Exercise: Select all the stock with price greater than 2
  - stockquote[@price > 2] ?
  - Stockquote[@price > 2]

Other language constructs

- If statement

```
<xsl:template match="line">
  <xsl:if test="position() mod 2 = 0">    </xsl:if>
  <xsl:value-of select="."/><br/>
</xsl:template>
```

- Template definition and call
  - Sample input:

```
<?xml version="1.0" ?>
<stocks>
 <stock exchange="nasdaq">
   <name>amazon corp  </name>
   <symbol>amzn</symbol>
   <price>16</price>
  </stock>
  <stock exchange="nyse">
    <name>IBM inc</name>
    <price>102</price>
  </stock>
</stocks>
```

stock.xml

Template definition and call

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

<xsl:template match="/">
<companyList>
<xsl:for-each select="/*/stock">
 <company> <xsl:value-of select="name"/>
 <stockPrice currency="CAD">
  <xsl:call-template name="priceConversion">
      <xsl:with-param name="amt" select="price"/></xsl:call-template>
 </stockPrice>
 </company>
</xsl:for-each>
</companyList>
</xsl:template>

<xsl:template name="priceConversion">
 <xsl:param name="amt" />
 <canadianPrice> <xsl:value-of select="$amt * 1.5"/></canadianPrice>
</xsl:template>

</xsl:stylesheet>
```

```xml
<?xml version="1.0" ?>
<stocks>
 <stock exchange="nasdaq">
   <name>amazon corp  </name>
   <symbol>amzn</symbol>
   <price>16</price>
  </stock>
  <stock exchange="nyse">
    <name>IBM inc</name>
    <price>102</price>
  </stock>
</stocks>
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
- <companyList>
 - <company>
     amazon corp
   - <stockPrice currency="CAD">
       <canadianPrice>24</canadianPrice>
     </stockPrice>
   </company>
 - <company>
     IBM inc
   - <stockPrice currency="CAD">
       <canadianPrice>153</canadianPrice>
     </stockPrice>
   </company>
</companyList>
```

XSLT rule: <xsl:template>

```
<xsl:template match="stock">
<company>
<value>
   <xsl:value-of select="price*1.5"/> CAD
</value>
<name>
   <xsl:value-of select="name"/>
</name>
<company>
</xsl:template>
```

xslt template for <stock>

```
<?xml version="1.0" ?>
<stocks>
 <stock exchange="nasdaq">
   <name>amazon corp  </
   name>
   <symbol>amzn</symbol>
   <price>16</price>
  </stock>
  <stock exchange="nyse">
    <name>IBM inc</name>
    <price>102</price>
  </stock>
</stocks>
```

stock.xml

```
<company>
<value>
 get the value of <price>* 1.5, i.e. 24 CAD
</value>
<name>
   get the value of <name>, i.e amazon
</name>
</company>
```

Part of the output

Transforming XML to HTML

```
<stocks>
   <stock exchange="nasdaq">
      <name>amazon corp</name>
      <symbol>amzn</symbol>
      <price>16</price>
   </stock>
   <stock exchange="nyse">
      <name>IBM inc</name>
      <price>102 </price>
   </stock>
</stocks>
```

stock.xsl

```
<xsl:template match="/">
<html><body>
Stock table <TABLE border="1">
<TR> <TH> Exchange </TH> <TH>Name</TH> <TH>Price</TH> </TR>
<xsl:apply-templates select="stocks/stock" />
</TABLE></body></html>
</xsl:template>

<xsl:template match="stock" >
  <tr>
  <TD><xsl:value-of select="@exchange"/>  </TD>
  <TD><xsl:value-of select="name"/>  </TD>
  <TD><font color="red">  <xsl:value-of select= "price * 1.5 "/> </font> CAD </TD>
  </tr>
</xsl:template>
```

toHTML.xsl

XSL Output.html

Stock table

Exchange	Name	Price
nasdaq	amazon corp	24 CAD
nyse	IBM inc	153 CAD

Assignment

```
<html>
<script>
   function addDisplay (details) {
      document.write(details);
   }
   </script>
<body>
<table border="1">
<th>Course No.</th><th>Course Title</th><th>Prof</th><th> Year
      </th><th> Days</th><th>Time</th><th>Location</th>
<tr>
<td>E-169c</td><td> <a href="JavaScript:addDisplay('This course
      presents an overview of the immune system and focuses on
      providing the student with a solid background in modern
      molecular and cellular immunology. The course covers the
      molecules of the immune system, including antibodies, T-cell
      receptors, and major histocompatibility locus-encoded
      proteins and cytokines; the genes encoding these molecules;
      the cells of the immune system and their interactions; and the
      biological functions of the immune system and its regulation.
      Prerequisite: background in biology, biochemistry, and
      genetics.');">Immunology I</a>
```

```
- <courses>
  - <course acad_year="2004" term_id="1" crn="10018">
      <course_group>BIOL</course_group>
      <course_num>E-169c</course_num>
      <title>Immunology I</title>
    - <meeting>
        <meeting_days>W</meeting_days>
        <meeting_begin>1935</meeting_begin>
        <meeting_end>2135</meeting_end>
        <location>Science Center Hall C</location>
      </meeting>
    - <course_head>
      - <person person_id="Pe1be2ae94357a0cb43164abef98ec438">
          <person_name>Jeffrey Lyczak</person_name>
          <person_lname>Lyczak</person_lname>
          <person_fname>Jeffrey</person_fname>
          <person_title>PhD, Manager of Microbiology, Nucryst Pharma
        </person>
      </course_head>
      <description>This course presents an overview of the immune sys
        student with a solid background in modern molecular and cellula
        the molecules of the immune system, including antibodies, T-cel
        histocompatibility locus-encoded proteins and cytokines; the ge
```

Course No.	Course Title	Prof
E-169c	Immunology I	Jeffrey Lyczak
E-2a	Organic Chemistry I	Craig Masse , Gregory Ph Tochtrop
E-1 Section 1	Elementary Modern Chinese I	Raymond D. Lum
E-30	Elementary Modern Chinese III	Min Wan
E-113	Introduction to C, Unix/Linux, and CGI	Bruce Molay

Xsl:attribute

- Add an attribute name and its vaule
- If you want

   …
- You may want to write

  <a href="<xsl:value_of select="description">"> …
- But that is not a valid xml document
- We need to give a value to an attribute

```
<xsl:attribute name="href">
    <xsl:value_of select="description">
</xsl:attribute>
```

Add attribute to an element

- Task: pass XSLT values as JavaScript parameter
  - When clicking on the hyper link such as Fundamentals of Grammar, the course description will popup
- Popup a window

```
function addDisplay (details) {
      document.write(details);
}
```

- Pass values in XSLT to JavaScript

```
<a href="JavaScript:addDisplay( 'someTextExtractedUsingXSLT')"> Fundamentals of
    Grammar </a>
➔
<a href="JavaScript:addDisplay( <xsl:value_of select="description">> Fundamentals of
    Grammar </a>
➔
<a> <xsl:attribute name="href">
        JavaScript:addDisplay(' <xsl:value-of select="description"/> ');  </xsl:attribute>
      Fundamentals of Grammar </a>
```

Summary

- XSLT is an XML transformation language
  - The input is an XML document, the output can be in any format
  - XSLT itself is in XML format
- XSLT is very different from other programming languages
  - Rule based (templates, …)
  - Processing model (scan from the root of the input)
- It also has the conventional programming language constructs
  - If, for-each, variable, procedure declaration and invocation …
- It relies on XPath
  - XPath itself is not in XML format

- Questions?

- "If you give someone a program, you will frustrate them for a day; if you teach them how to program, you will frustrate them for a lifetime."
  - Anonymous