Introduction to Extensible Stylesheet Language: XSL

Lecture 2: XSLT

20 March 2013

Roman Bleier, TCD
bleierr@tcd.ie

What is XSL?

Extensible Stylesheet Language (XSL)

"A family of recommendations for defining XML document transformation and presentation..."

W3C recommendation: http://www.w3.org/Style/XSL/

Three Parts:

- **XSLT**: transformation language
- **XSL-FO**: formatting language
- **XPath** is used to address the element in the source document.

XSL Transform and Formatter

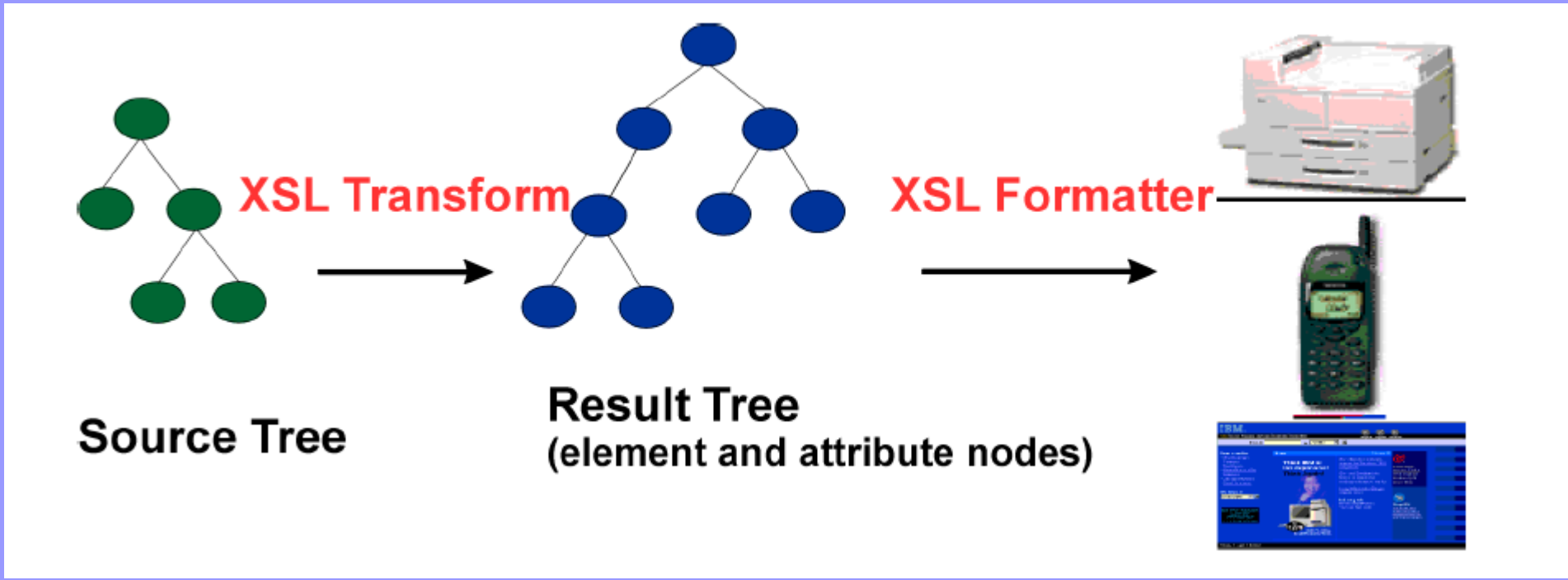


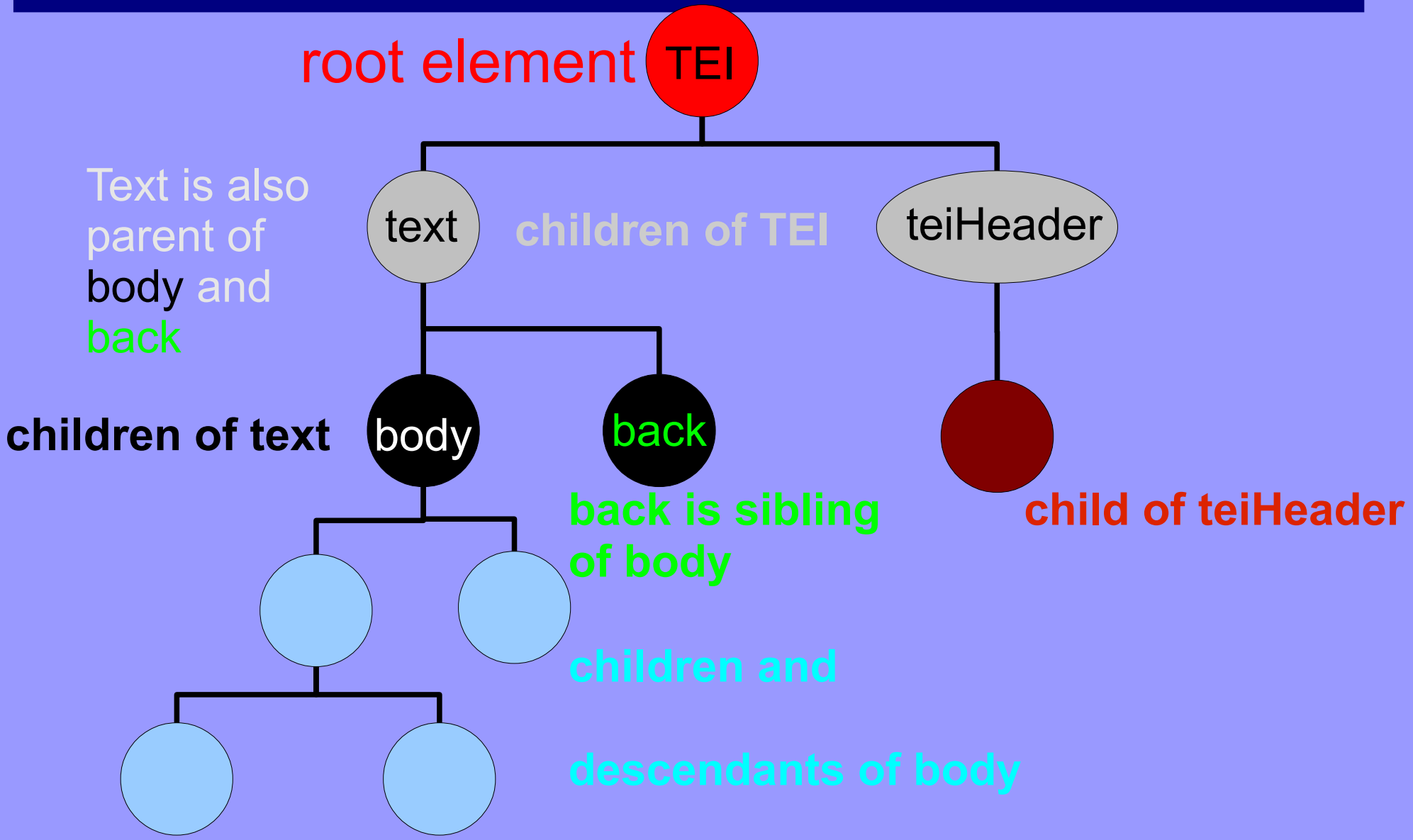
Image from the w3c recommendations: http://www.w3.org/TR/xsl/

Recap XPath - Week 1

- Path language for hierarchical addressing of nodes and axes in XML document
- Data model provides a tree representation of the XML document
- **Nodes:** *element, attribute, text, comment nodes etc.*
- **Axes:** *self-, child-, parent-, attribute-, following-axis etc.*
- Some examples:

  http://www.tei-c.org/Talks/OUCS/2006-02/talk-xpath.pdf

XPath - XML tree



XPath - Axes

- ancestor::
- ancestor-or-self::
- parent::
- preceding::
- preceding-sibling::
- self::
- following-sibling::
- following::
- child::
- descendant-or-self::
- descendant::

- attribute::
- namespace::

XPath Functions

- **Shorthand Notation, Wildecard, Predicates:**

  . (dot) stand for the current node

  @ is the same as attribute::

  // is used to select nodes on any hierarchy

- **Wildecard: ***
- **Predicates: [ ]**
- **Functions:**

  not() position() last() contains() etc.

- **Operators:**

  = != - + * or and | *etc.*

XPath Examples

Selecting nodes based on names and position:

- //l[last()][@n]
- //body/div//*[1][not(self::l or self::lg)]
- //l[position()=1 or position()=last()]

Selecting nodes based on text, data or number of child nodes they contain:

- //l[text()]
- //l[last()][@n=2 or @n=3]
- //*[self::editor][preceding-sibling::author or following-sibling::author][contains(../title[1],"Patrick")]
- //lg[count(*)>=4]

Recap XSLT – Week 1

Set of elements to describe rules for transforming XML content (XML to XML)

i.e. TEI to XHTML

XSLT is a XML language:

well-formed, xsl: namespace, <xsl:stylesheet> is the root element

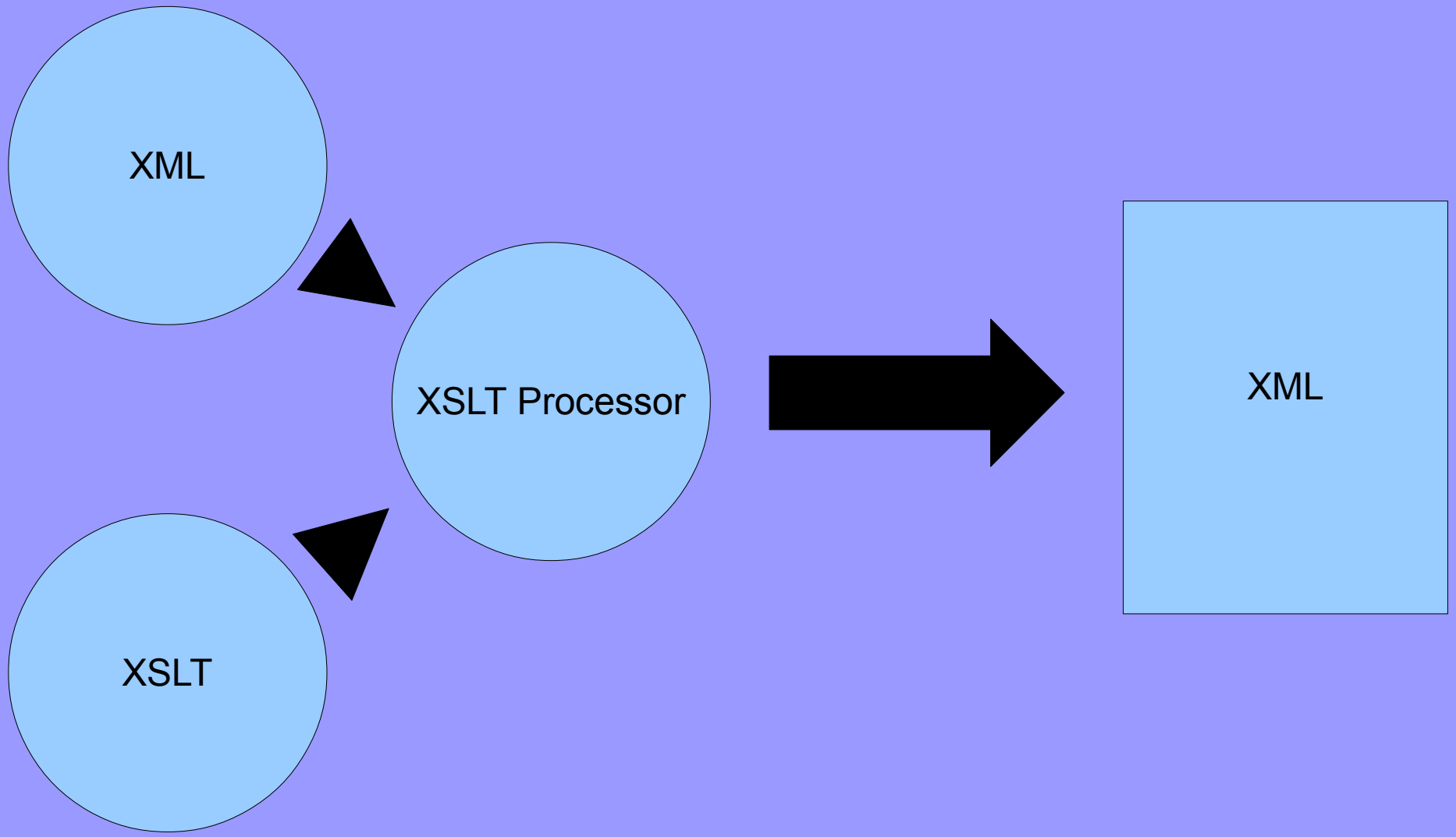
The file ending is .xsl

XPath expressions are used to select elements for processing

XSLT processor (i.e. Saxon)

Processing: Server-side, Client-side (web browser) or third software (i.e. Oxygen)

How does a Transformation work?



XSLT Tree Transformation

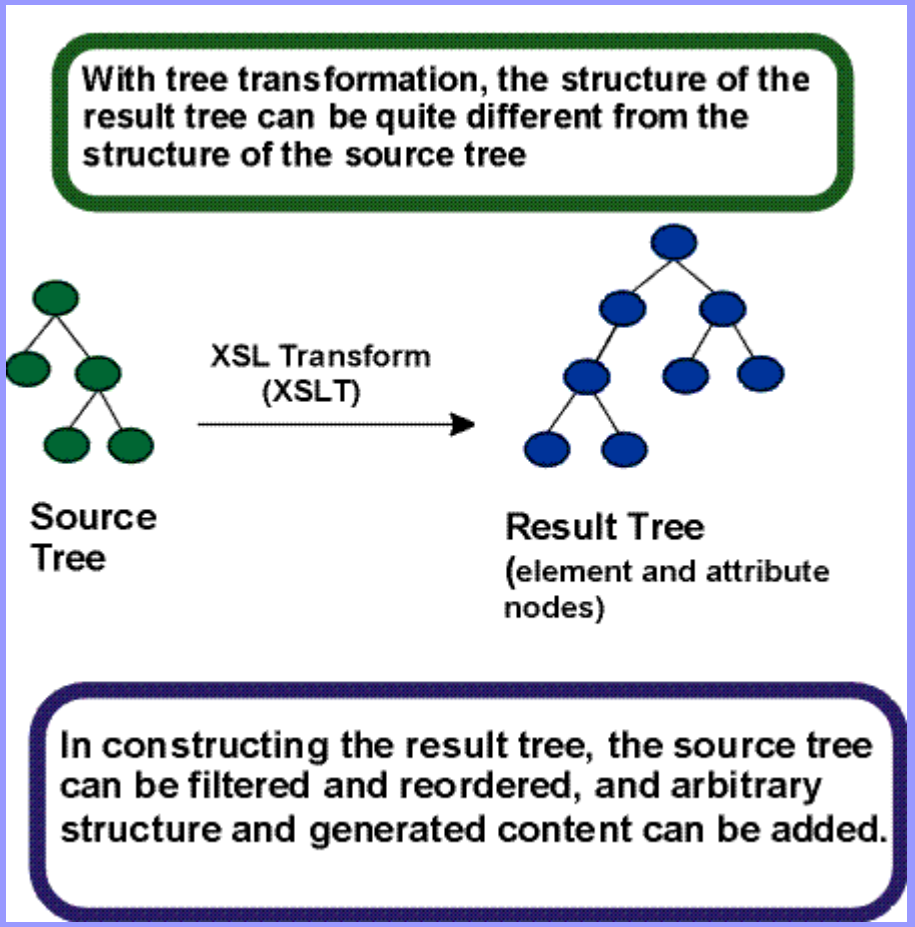


Image from the w3c recommendations: http://www.w3.org/TR/xsl/

Stylesheet and Template

Root-element: **xsl:stylesheet** or **xsl:transform**

Rules for transformation: **xsl:template**

Invoking the build-in templates by using **apply-templates** without any **@select**:

```
<xsl:stylesheet xmlns:xsl="...">
   <xsl:template match="/">
      <xsl:apply-templates></xsl:apply-templates>
   </xsl:template>
</xsl:stylesheet>
```

RESULT: only the text nodes get output

Apply-templates

Default behaviour – it applies templates to the child nodes

<xsl:apply-templates></xsl:apply-templates>

This can be changed by selecting nodes via the @select:

<xsl:apply-templates select="tei:lg"> </xsl:apply-templates>

=>will only happen if a child element of the current element is tei:lg

<xsl:apply-templates select="//tei:lg/tei:l"> </xsl:apply-templates>

=>looks for tei:l that is a child of tei:lg on any hierarchy (//) below current element

Continue XSLT Exercise 1

- Use Confessio_Ferguson.xml
- Create a new XSLT file, name it and safe it in the same folder as Confessio_Ferguson.xml
  - First invoke the build-in templates as shown before
  - Observer how the XML content is displayed in the Browser
- Transform the XML into a valid HTML web page:
  - Display only the body content
  - The title (tei:body/tei:div/tei:head) should become <h2>
  - The lines should be displayed as paragraphs: tei:l should become HTML <p>
  - Display the line number @n at the beginning of the line
  - Name the created HTML file **text.html**

Push vs Pull in XSLT

Also single- and multi-template approach

Single-template and pull-method:

- 'pull' nodes by directly addressing them, only template for root:
  - no use of xsl:apply-template
  - xsl:value-of and xsl:for-each used to retrieve nodes

Multiple-templates and push-method:

- the source tree nodes get pushed through the stylesheet:
  - xsl:apply-template is used to iterate over nodes
  - a series of xsl:template is used to specify the treatment of nodes

- http://www.xml.com/pub/a/2005/07/06/tr.html

  http://www.ibm.com/developerworks/library/x-xdpshpul.html

Output with value-of

- Value-of is used to generate text-nodes from source tree, variables, parameters
- @select (optional)
- **Examples:**

```
<xsl:value-of select="."/>
<xsl:value-of select="$variable"/>
<xsl:value-of select="lg"/>
<xsl:value-of select="text()"/>
<xsl:value-of select="@type"/>
```

- **It is NOT applying templates for descendant elements !**

value-of and apply-templates

```
<head>
The <hi xml:lang="lat">Confessio</hi> of Saint Patrick
</head>
```

```
<xsl:template match="tei:body//tei:head">
        <p><xsl:apply-templates></xsl:apply-templates></p>
        <p><xsl:value-of select="."/></p>
</xsl:template>
<xsl:template match="tei:hi">
<strong><xsl:value-of select="."/></strong>
</xsl:template>
```

uses template for tei:hi

xsl:for-each

'processes each item in a sequence of items'

The following produces a list of all books with the author and title element displayed:

```
<ul>
  <xsl:for-each select="tei:TEI//tei:back//tei:bibl[@type='Book']">
    <li>
      <xsl:apply-templates select="tei:author"></xsl:apply-templates>,
      <xsl:value-of select="tei:title"/>
    </li>
  </xsl:for-each>
</ul>
```

What happens if there are two tei:author or tei:title elements in a tei:bibl?

xsl:sort

For sorting of data

@select, @data-type, @order etc.

Used with xsl:for-each

```
<xsl:for-each select="tei:TEI//tei:back//tei:bibl">

<xsl:sort select="tei:author/tei:name/tei:surname" data-type="text"></xsl:sort>

<xsl:sort select="tei:date/@when | tei:date/@from" data-type="number"></xsl:sort>

<!--statements-->

</xsl:for-each>
```

XSLT Exercise 2 - pull-method

- Source XML: Confessio_Ferguson.xml
- Create a second XSLT file in the same folder.
- The element <back> contains a bibliography
- Write a new XSLT that will transform the XML into a valid HTML web page:
  - Create a table: columns for the author(s)/ editor(s), title(s) and year(s) of publication. Sort the table by year of publication.
  - The title (tei:back/tei:div/tei:head) should become <h2>
  - **Use the pull-method** as described in the slide before to "pull" data out of the XML and fill your HTML table.
  - Name the created HTML file biblio.html

Variables and Parameters

Variables:

To store values

No overwriting of variables ($x = $x+1) 

Required: @name; optional: @select

```
<xsl:variable name="var_name">VALUE</xsl:variable>
<xsl:variable name="var_name" select="@type"></xsl:variable>
```

```
<div class="{$var_name}">
<xsl:value-of select="$var_name"/>
```

Global and Local Scope

- Variables and Parameters:
- Global Scope – defined as child of the stylesheet element:
  - <xsl:stylesheet xmlns:xsl="...">
  - <xsl:param name="email">default@tcd.ie</xsl:param>
  - …
  - </xsl:stylesheet>
- Local Scope – within templates:

```
<xsl:template name="footer">
<xsl:variable name="email">myemail@tcd.ie</xsl:variable>
```

Global and Local Scope

Local variable shadowing a global variable:

```
<xsl:param name="email">default@tcd.ie</xsl:param>
```

Global Var

```
<xsl:template name="footer">
```

Local Var

```
<xsl:variable name="def-email" select="$email"></xsl:variable>
<xsl:variable name="email">myemail@tcd.ie</xsl:variable>
<p>
  Contact Details:
   <strong><xsl:value-of select="$email"></xsl:value-of></strong><br/>
  Default Email: <strong><xsl:value-of select="$def-email"/></strong>
</p>
</xsl:template>
```

Variables and Parameters

Parameters:

Stylesheet, template and function parameters

Parameters may be passed on by template or function calls

Required: @name; optional: @select

```
<xsl:param name="par_name">VALUE</xsl:param>
<xsl:param name="par_name" select="$val"></xsl:param>
```

```
<div class="{$par_name}">
<xsl:value-of select="$par_name"/>
```

Parameter and Call-template

Call-template statement passes on parameters:

```
<xsl:call-template name="footer">
    <xsl:with-param name="email">myemail@tcd.ie</xsl:with-param>
</xsl:call-template>
```

```
<xsl:template name="footer">
    <xsl:param name="email">default@tcd.ie</xsl:param>
    <p>
        Contact Details: <xsl:value-of select="$email"></xsl:value-of>
    </p>
</xsl:template>
```

Conditional Processing

- **xsl:if** [no xsl:else or xsl:else-if, use xsl:choose]
- **xsl:choose \ xsl:when - xsl:otherwise**
- **@test** tests for a condition to be true or false

```
<xsl:if test="@type='title'">
    <xsl:text>There is a title!</xsl:text>
</xsl:if>
```

```
<xsl:choose>
        <xsl:when test="@title">...</xsl:when>
        <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

Looping

- For-each element

```
<xsl:for-each select="tei:l">
    <li><xsl:value-of select="."/></li>
</xsl:for-each>
```

- Template element

```
<xsl:template match="tei:l">
    <li><xsl:value-of select="."/></li>
</xsl:template>
```

- Recursion
  - IBM devWorks: Loop with recursion in XSLT

Grouping and XSL:Function

- Grouping: based on content or position

  <xsl:for-each-group select="" group-by="@type">

  http://www.xml.com/lpt/a/1314

- Functions: http://www.xml.com/lpt/a/1278

  Namespace: xmlns:mfs="functionNamespace"

  Function call: mfs:myFunction($par1, $par2)

  Function declaration:

  - <xsl:function name="mfs:myFunction">

    <xsl:param name="par1"></xsl:param>

    <xsl:param name="par2"></xsl:param>

    <!-- Function action goes here-->

  </xsl:function>

More XSLT Elements

- xsl:text, xsl:comment, xsl:number
- xsl:element, xsl:attribute and xsl:attribute-set
- xsl:preserve-space and xsl:strip-space

```
<xsl:preserve-space elements="tei:l tei:title" />
```

- xsl:copy (copy of the current node only) and xsl:copy-of (deep copy of the current node)
- xsl:output <xsl:output method="html"/>
- xsl:import (lower precedence) and xsl:include (same precedence)

XSLT Development Tips

For better readability of your XSLT:

- Use comments: <!-- comment goes here -->
- Code additions should be documented: i.e. with a comment <!-- roman's templates-->
- Organise your code in templates
- Global Variables/ Parameters should be in one place
- If your file gets too big use multiple files:

```
<xsl:import href="filename.xsl"/>
```

XSLT Exercise 3

- Source XML: Confessio_Ferguson.xml
- You created already two web pages: text.html and biblio.html
- Rewrite the XSLT for the biblio.html
- This time use a multiple-templates approach:
  - The final product should be a unordered list <ul> of bibliographic entries
  - tei:head should be again displayed as <h2>
  - Create templates for all child elements of tei:bibl:
    - Multiple authors/editors should be separated by comma
    - Titles should be in italics and separated by a colon name : title
    - The date should be bold: 1923