XSLT: Transforming XML data to another form

Dr Richi Nayak
School of Information Systems
Research Group: Intelligent Systems

Reference: 1. Text book: XML in a nutshell, Harold & Means
2: Learning XSLT, Fitzerald M

Overview

- Introduction to XSL
- Importance of Transformation
- XSLT Operational Model
- XSLT Stylesheet structures
  - XSLT to create HTML
  - Using XSLT to do XML to XML conversion
- XSLT Templetes
- Using XSLT to sort and count

Introduction to XSLT

- Stylesheets
- Stylesheet language
  - XSL-FO
  - XSLT

Stylesheets

A stylesheet is a file which contains a declarative set of rules for converting an XML document into another document, which can be XML, HTML, XHTML, plain text, pdf ...

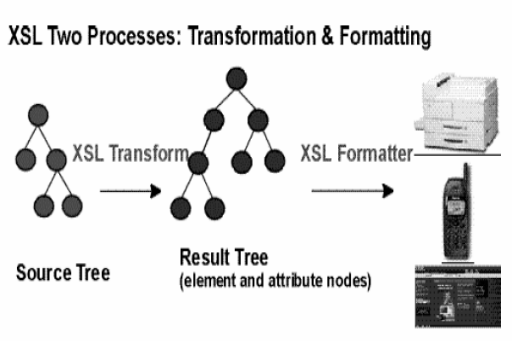
XSL

- eXtensible Stylesheet Language
- A language for expressing stylesheets
- Consists of two parts
  - XSL Transformation (XSLT)
  - XSL Formatting Objects (XSL-FO)
- Uses XPath to access or refer to parts of an XML document.

XSL: eXtensible Stylesheet Language

- XSL Transformation (XSLT)
  - An XML vocabulary for transforming an XML document into another XML document
  - Unlike with a programming language, you don't need to be a programmer to successfully describe how to transform your information.
  - XSLT implements transformation "by example", not just "by program logic", and builds in support for the kinds of transformation typically needed to present information.
- XSL Formatting Objects (XSL-FO)
  - An XML vocabulary to define XML display very high-quality

XSL Stages



7

XSLT Versions

- History
  - XSL Transformations (XSLT) Version 1.0 is a recommendation since 11/99 (first draft dates back to 8/98)
  - Version 2.0 is a recommendation since 01/07.

8

Importance of Transformation

- XSLT Data Transformation
- POP
- MOM

9

Motivation

- XML provides a syntax for semi-structured data formats.
- No one format is likely to enable all possible uses for data
- Transforming XML is useful in two scenarios:
  - Presentation Oriented Publishing (POP)
    - Useful for Browsers and Editors
    - Transforming data to a human viewable forms
      - On the Web as HTML
      - In print using PDF
  - Message Oriented Middleware (MOM)
    - Useful for Machine-to-Machine data exchange
      - Data conversion from one format to another
    - Business-to-Business communication an excellent example
      - Multiple data formats for B2B purchase orders

10

Importance of Transformation

- XSLT is incredibly useful in
  - transforming data into a viewable format in a browser (POP)
  - transforming business data between content models (MOM)

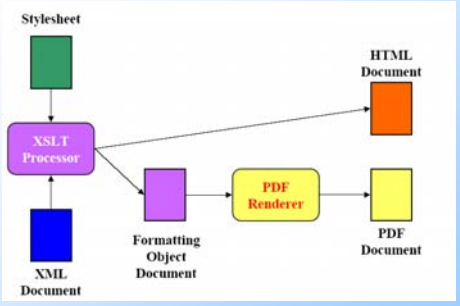
11

XSLT in POP

- XML document separates content from presentation
- Transformations can be used to style (render, present) XML documents
- A common styling technique presents XML in HTML format

12

XSLT in POP



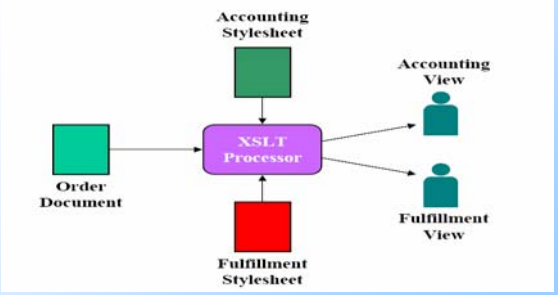
13

XSLT in MOM

- Important for eCommerce, B2B/EDI, and dynamic content generation
  - Different content model
  - Different structural relationship
  - Different vocabularies

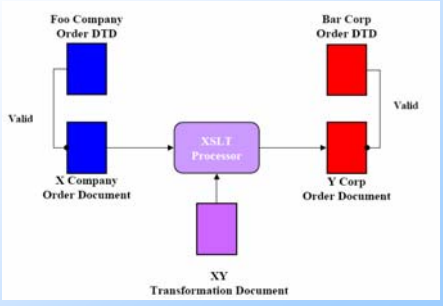
14

XSLT in MOM



15

XSLT – Data Transformation



16

XSLT Operational Model

- XSLT Transformation Process
  - XSLT Processor
  - XSLT Stylesheet or XSLT Program

17

XSLT Transformations: Process

- XSLT style sheet: template rules
  - pattern which specifies which tree it applies to
  - pattern which specifies which tree it should output
- XSLT processor
  - reads XML document and XSLT stylesheet
  - carries out the instructions in the stylesheet
  - outputs a new document

18

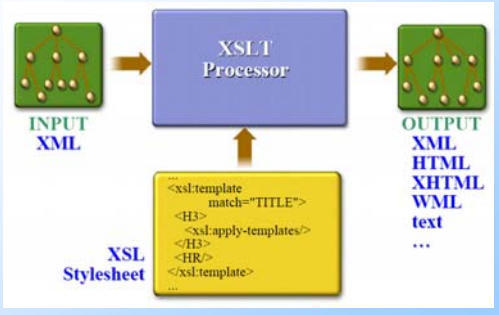
Result

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

<p> A Famous Physicist </p>

<p> A Famous Physicist </p>
```

•Template content contains tags and character data
•They have to be well-formed XML.

19

XSLT Operational Model



20

XSLT Processor

- Piece of software
  - Reads an XSLT stylesheet and input XML document
  - Converts the input document into an output document
  - According to the instruction given in the stylesheet
- Called stylesheet processor sometimes

21

Examples of XSLT Processor

- Built-in within a browser
  - IE 5.5 (not compatible to XSLT standard)
- Built-in within web or application server
  - Apache Cocoon
- Standalone
  - Michael Kay's SAXON
  - Apache.org's Xalan

22

XSLT Stylesheet

- Well-formed XML document
- Contains a number of expressions and elements stating the rules of transformations
- An example:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="try">
      Print this line
</xsl:template>

</xsl:stylesheet>
```

23

XSLT Stylesheet structure

- Namespace
- Root element : stylesheet or transform
- Top-level directories
  - Output methods
- A simple example

24

XSLT Structure

- Stylesheet specification
  - XSLT Namespace
    xmlns:xsl= http://www.w3.org/1999/XSL/Transform
    - 1999 refers to the year in which the URI was allocated by the W3C, not to the version of XSLT
  - Stylesheet root element
    - stylesheet or transform
    - Both are defined in standard XSLT namespace
      - xsl as customary prefix

    <xsl:stylesheet version="1.0">
    or
    <xsl:stylesheet version="2.0">
    or
    <xsl:transform version="1.0">
- Top-level directives

Stylesheet root element

- A transformation is specified by a "stylesheet" or "transform" document:

```
<xsl:transform  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='2.0' >
        <!-- your rules here -->
</xsl:transform>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='2.0' >
        <!-- your rules here -->
</xsl:stylesheet>
```

- Applications may interpret 'stylesheet' differently from 'transform' in terms of:
  - whether they run the transformation
  - what they do with the results.

The Top Level

- The "Top Level" refers to the children of the document element.
- Any element can occur at the top level but it must have a namespace

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='2.0'
    xmlns:my="http://cde.berkeley.edu/my/other/stuff" >
    <!-- your rules here -->
    <my:other-stuff type="random-crap"/>
</xsl:transform>
```

  - This is illegal:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version='2.0'>
  <!-- your rules here -->
  <mr-no-namespace-name/>
</xsl:stylesheet>
```

  - Typically, you'll use elements from the XSLT namespace.

The Top Level

- Top-level directives
  - Output (text, html, xml)
    <xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>
  - Whitespace (can strip or preserve from specified elements)
    <xsl:strip-space elements="sale name"/>
  - Import/include other XSLT programs
    <xsl:include href=*uri-reference*/>
    - (as if defined here)

    <xsl:import href=*uri-reference*/>
    - (importer has priority)
  - List of *templates*

Output Methods

- Various types of *output methods*
  - html
    - default when first tag emitted is <html>
  - Xml
    - default
  - text
  - xhtml

```
<!DOCTYPE xsl:stylesheet>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0">
   <xsl:output method="html"/>
   <xsl:template match="…">
      …
   </xsl:template>
</xsl:stylesheet>
```


XML-stylesheet Processing Instruction

- Included as part of XML document
- Tells XML-ware browser where to find associated stylesheet

```
<?xml version="1.0"?>
<?xml-stylesheet
   type="text/xml"
   href="http://www.oreilly.com/styles/people.xsl"?>
   <people>
   ....
   </people>
```

Example 1: Minimal but Complete XSLT Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

31

XML Example Document

```
<?xml version="1.0"?>

    <people>
        <person born="1912" died="1954">
                <name>
                        <first_name>Alan</first_name>
                        <last_name>Turing</last_name>
                </name>
                <profession>computer scientist</profession>
                <profession>mathematician</profession>
                <profession>cryptographer</profession>
        </person>
        <person born="1918" died="1988">
                <name>
                        <first_name>Richard</first_name>
                        <middle_initial>M</middle_initial>
                        <last_name>Feynman</last_name>
                </name>
                <profession>physicist</profession>
                <hobby>Playing the bongoes</hobby>
        </person>
        .....
</people>
```

32

Example 1: Result of XSLT Processing

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


   Alan
   Turing

computer scientist
mathematician
cryptographer


   Richard
   M
   Feynman

physicist
Playing the bongoes
```

33

Result of XSLT Processing

- Applying empty stylesheet to any XML document
  - Elements are traversed sequentially
  - Content of each element is put in output
    - Attributes are NOT traversed
  - Default behavior
- Without any specific templates
  - XSLT processor falls back to default behavior
- Need for templates

34

XSLT Templates (coverage)

- Basics
- Template Match Patterns
- Template Matching
- Simple Examples 2 to 5
- Text processing
- xsl:element
- xsl:attribute
- xsl:comment
- xsl:value-of
- xsl:copy-of
- xsl:copy
- Activation of templates

35

XSLT Templates ..cont

- xsl:apply-templates
- Mulitple modes
- Filter
- xsl:for-each
- Selection – xsl:if and xsl:choose
- Sorting
- Counting
- Variables and parameters

36

XSL Templates: Basics

- Templates are "pictures" of the result.
  - Templates define how to output XML fragments
  - *xsl:template* element form
- They are associated with the input document by matching patterns.
  - A *match* attribute is used to associate the template with an XML element
  - The value of the match attribute is a set (super/subset) of an XPath expression.

    <xsl:template match="/">
- A template is always instantiated with respect to a *current node* and a *current node list*.
  - both are specified by the match
- Elements in the XSLT namespace are replaced by their actions.
- Literal elements are copied to the output.

37

Template Match Pattern

- Similar to XPath expressions, but
  - allows top-level '/' signs
  - only 'child' and 'attribute' axes allowed
  - allows '//' for descendants
  - allows id(name) to start the expression
  - predicates are not restricted.
  - can have multiple patterns separated by '|' to match several patterns.
  - Example:

    contents/p|
    slide[@show='true']|
    @hrefa|
    graphic[ancestors::section]

38

Literal Elements

- A literal element is a non-XSL element.
- It generates a copy of itself in the output.
- The children may be generated by subsequent templates.
- For example:

```
<xsl:template match="people">
 <html>
   <head> <title>Famous People Document</title>
     <style type='text/css'>...</style>
   </head>
   <body><xsl:apply-templates/></body>
 </html>
 </xsl:template>
```

39

Template Matching: Process

- The stylesheet processor goes through the XML document one element at a time, finds the first template which that element matches, and carries out the instructions in that template.
- If the element does not match any template in the stylesheet, then the default behaviour is for the processing to pass through to the children of this element without carrying of any instructions.
- When the processing reaches an element which has only text children, the result of processing these children is to print out the text.

40

Example 2: Simple XSLT Stylesheet

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

- Simplest form of Xpath pattern is a Name of a single element.

41

Result: Example 2

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

42

Example 3: Simple XSLT Stylesheet

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

        <xsl:template match="people">
                Two Famous Physicists
        </xsl:template>
</xsl:stylesheet>
```

- Literal data characters - text copied from the stylesheet into the output document

43

Result: Example 3

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

Two Famous Physicists
```

- people element is replaced by contents of template

44

Example 4: Simple XSLT Stylesheet

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

        <xsl:template match="person">
                A Famous Physicist
        </xsl:template>
</xsl:stylesheet>
```

45

Result: Example 4

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

A Famous Physicist

A Famous Physicist
```

- person element is replaced by contents of template

46

Example 5: Simple XSLT Stylesheet

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

        <xsl:template match="person">
                <p>A Famous Physicist  </p>
        </xsl:template>
</xsl:stylesheet>
```

- Literal result elements - elements copied from stylesheet to output document

47

Result: Example 5

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

<p>A Famous Physicist </p>

<p>A Famous Physicist </p>
```

- person element is replaced by contents of template

48

Outputting Text

- Any non-whitespace inside a template is automatically copied to the output.
- Whitespace that contains a non-whitespace character is copied as well:

```
<xsl:template match="foo"> some text </xsl:template>
or
 <xsl:template match="foo"> some
        text
</xsl:template>
```

- Whitespace between elements is "stripped" and not copied:

```
<xsl:template match="foo">  <p>some text</p>  </xsl:template>
 or
<xsl:template match="foo">
        <p>some text</p>
</xsl:template>
```

generates the following without a leading or trailing carriage return:

```
<p>some text</p>
```

49

Preserving Whitespace

- The element 'xsl:text' preserves text and whitespace.
- For example, to preserve the whitespace in the previous example:

```
<xsl:template match="foo"><xsl:text>
  </xsl:text><p>some text</p><xsl:text>
  </xsl:text></xsl:template>
```

- This is often used to add whitespace between non-literal elements.

50

Comments processing

- Comments and processing instructions are ignored.
- They aren't copied to the output.
- For example:

```
<xsl:template match="foo">
 <!-- The next element is significant -->
   <spam type='fried'/>
</xsl:template>
```

generates:

```
<spam type='fried'/>
```

51

Understanding Actions

- Templates specify actions.
- A literal element or text is really an action to create a copy.
- There are many other kinds of actions specified by elements in the XSLT namespace:
  - apply-templates, call-template, apply-imports, for-each, value-of, copy-of, number, choose, if, text, copy, variable, message, fallback, processing-instruction, comment, element, attribute.
- XSLT is extensible: a processor can add to these.

52

Creating Elements "Manually"

- Elements can also be created by xsl:element.
- This is used when the element name or namespace is created based on a expression.
- An example:

```
<xsl:element name="top"> <a/><b/><c/> </xsl:element>
```

constructs:

```
<top> <a/><b/><c/> </top>
```

- The children of 'xsl:element' are the children of the newly created element.
- You can use path expressions in the name:

```
<xsl:element name="{@born}"/>
```

or

```
<xsl:element name="{people/person}"/>
```

53

Creating Attributes "Manually"

- Attributes can also be created by xsl:attribute.
- They must be created before children are added to the element.
- You can use them on literal elements:

```
<section> <xsl:attribute name="id">sect1</xsl:attribute> </section>
```

- Or on xsl:element constructions

```
<xsl:element name="section">
      <xsl:attribute name="id">sect1</xsl:attribute>
</xsl:element>
```

- The children of xsl:attribute must be text nodes that represent the value of the attribute.
- You can use expressions in the name:

```
<xsl:attribute name="{child/@ref}"/>
```

54

Attribute Value Templates

- XPath expressions can be used to "insert" content into attribute values.
- Attribute value templates are delimited by curly braces: {…}
- Double curly braces are used if you want a curly brace in the attribute value.
- The expression result becomes the attribute value.

55

Attribute Value Templates - Example

- For example

for the content:

<image-data base-uri="http://mydomain.com"
src="picture.jpg"/>

would generate:

56

Creating Comments and Processing Instructions "Manually"

- Comments are created by:

```
<xsl:comment>
    your comment text here
</xsl:comment>
```

- Processing Instructions are created by

```
<xsl:processing-instruction name="target">
    your PI text here
</xsl:processing-instruction>
```

57

Getting Values - *xsl:value-of*

- Extracts the value of an element or an attribute and writes it to output
  - Output the text content of the element after all the tags have been removed and entity references are resolved
- *select* attribute containing XPath expression identifies an element or an attribute
  - It could be a node set, in which case, the string value of first node is taken

58

Example 6

- Extract names of all the people

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

  <xsl:template match="person">
   <p>
        <xsl:value-of select="name" />
   </p>
  </xsl:template>

</xsl:stylesheet>
```

59

Result: Example 6

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

  <p>
    Alan
    Turing
    </p>

  <p>
    Richard
    M
    Feyman
    </p>
```

60

Copying Nodes - *xsl:copy-of*

- Copies the specific node to output
  - The elements with their tag names are copied.
  - All the nodes' children, attributes, namespaces and descendants are copied as well.
- *select* attribute containing XPath expression identifies an element.

61

Example 7

- Extract names of all the people

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

  <xsl:template match="person">
      <xsl:copy-of select="name" />
  </xsl:template>

</xsl:stylesheet>
```

62

Result: Example 7

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

  <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
  </name>

  <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
  </name>
```

63

Copying a Node

- Sometimes you might want to copy a node to the output.
- xsl:copy will copy the matching node to the output.
- It only applies to the current node and not its children or attributes.
- Example:

```
<xsl:template match="credit-card">
    <xsl:copy>XXXX-XXXX-XXXX-XXXX</xsl:copy>
</xsl:template>
```

would create:

```
<credit-card>XXXX-XXXX-XXXX-XXXX</credit-card>
```

64

The Identity Transform

- You can specify the identity transformation with xsl:copy:

```
<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>
```

- This matches and copies all nodes to the output.

65

Activation of templates

- XSLT processor reads (traverses) the input XML document sequentially from top to bottom
  - Templates are activated in the order they match elements encountered
    - Template for a parent will be activated before the children
- At a node
  - No template matches - do nothing
  - A single template matches - evaluate body
  - Multiple templates match - only apply one

66

Multiple templates

- Multiple templates match - only apply one
  - Give preference to local templates over imported templates
  - Give preference to *more specific* matches, for example
    <xsl:template match="*">1</xsl:template>
    <xsl:template match="name">2</xsl:template>
    The first template matches *any* node, the second only *name* nodes.
    The second has preference for *name* nodes.
  - Give preference to higher priority templates
    - Single names (e.g. "person") have priority 0.
    - Wildcards (e.g. *, @*) have priority -0.25
    - Node tests for other nodes (e.g. comment(), node(), etc.) have priority -0.5
    - Otherwise, the priority is 0.5

Multiple templates - Priorities

- For example:
  - para → 0
  - h:* → -0.25
  - * → -0.25
  - node() → -0.5
  - contents/para → 0.5
  - contents/* → 0.5
- You can adjust the priority to get what you want with a 'priority' attribute.
  Example:

<xsl:template match="h:*" priority="1"> ... </xsl:template>

Change in order: Activation of templates

- To continue traversal (recursive descent)
  <xsl:apply-templates *select="XPath expression"*/>
- *xsl:apply-templates* lets you make your choice of processing order explicit.
- The order of the traversal can be changed by apply-templates
  - It can specify which element or elements should be processed next
  - It can specify an element or elements should be processed in the middle of processing another element
  - It can prevent particular elements from being processed

xsl:apply-templates

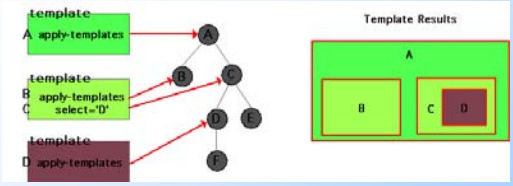
<xsl:apply-templates *select="XPath expression"*/>

- *select* attribute containing a XPath expression tells the XSLT processor which nodes to process in the input tree
  - The *apply-templates* with no *select* attribute means all elements relative to the current element (context node) should be matched

<xsl:apply-templates/>

- By default, the descendants are selected:

Template Results



The Results of applying templates to an instance.

xsl:apply-templates

- Let the output be:
  - Last name then first name
  - Only name not profession nor hobby

```
<?xml version="1.0" encoding="utf-8"?>
Turing
Alan

Feyman
Richard
```

xsl:apply-templates Example:8

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

<xsl:template match="name">
    <xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/>
</xsl:template>

 <!-- Something is missing here -->
</xsl:stylesheet>
```


Result: Example 8

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

    Turing
      Alan


  computer scientist
  mathematician
  cryptographer

    Feyman
      Richard


  physicist
  Playing the bongoes
```


xsl:apply-templates Example:9

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

    <xsl:template match="name">
          <xsl:value-of select="last_name"/>,
          <xsl:value-of select="first_name"/>
    </xsl:template>

    <!-- Apply templates only to name children -->
    <xsl:template match="person">
          <xsl:apply-templates select="name"/>
    </xsl:template>
</xsl:stylesheet>
```

- Order of templates does not matter.
- Only the order of elements in the input document matters.

Result: Example 9

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

    Turing
      Alan

    Feyman
      Richard
```


Example 10

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

    <xsl:template match="people">
          <html>
                <head><title>Famous Scientists</title></head>
                <body> <xsl:apply-templates select="person"/> </body>
          </html>
    </xsl:template>


    <xsl:template match="person">
          <xsl:apply-templates select="name"/>
    </xsl:template>


    <xsl:template match="name">
          <p><xsl:value-of select="last_name"/>,
          <xsl:value-of select="first_name"/></p>
    </xsl:template>

</xsl:stylesheet>
```


Result: Example 10

```
<html>
<head>
<title>Famous Scientists</title>
</head>

<body>


 <p>Turing,
    Alan</p>


 <p>Feynman,
    Richard</p>


</body>
</html>
```

Example 11: Handling Attributes

```
<?xml version="1.0"?>
    <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="people">
        <html>
          <head><title>Famous Scientists</title></head>
          <body>
            <dl>
                <xsl:apply-templates/>
            </dl>
          </body>
        </html>
    </xsl:template>
    <xsl:template match="person">
          <dt><xsl:apply-templates select="name"/></dt>
          <dd><ul>
           <li>Born: <xsl:apply-templates select="@born"/></li>
           <li>Died: <xsl:apply-templates select="@died"/></li>
          </ul></dd>
    </xsl:template>
</xsl:stylesheet>
```


Attributes

- Default rule does not apply
  - *apply-templates* has to be present in order to output values of attributes

Result: Example 11

```
<html>
    <head>
        <title>Famous Scientists</title>
    </head>
    <body>
        <dl>
          <dt>
            Alan
            Turing
          </dt>
          <dd>
            <ul>
                <li>Born: 1912</li>
                <li>Died: 1954</li>
            </ul>
          </dd>
          <dt>
            Richard
            M
            Feynman
          </dt>
          <dd>
            <ul>
                <li>Born: 1918</li>
                <li>Died: 1988</li>
            </ul>
          </dd>
        </dl>
    </body>
</html>
```


Multiple Modes

- Same input content needs to appear multiple times in the output document formatted according to different template
  - Titles of chapters
    - Table of contents
    - In the chapters themselves
- mode attribute
  - xsl:template
  - xsl:apply-templates

Example 12 with mode attribute

```
<?xml version="1.0"?>
    <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="people">
        <html>
          <head><title>Famous Scientists</title></head>
          <body>
                <ul> <xsl:apply-templates select="person"
    mode="toc"/></ul>
                <xsl:apply-templates select="person"/>
          </body>
        </html>
    </xsl:template>
    <!-- Table of Contents Mode Templates -->
    <xsl:template match="person" mode="toc">
          <xsl:apply-templates select="name" mode="toc"/>
    </xsl:template>
    <xsl:template match="name" mode="toc">
          <li><xsl:value-of select="last_name"/>,
          <xsl:value-of select="first_name"/></li>
    </xsl:template>
    <!-- Normal Mode Templates -->
    <xsl:template match="person">
          <p><xsl:apply-templates/></p>
    </xsl:template>
</xsl:stylesheet>
```


Result: Example 12

```
<html>
    <head>
        <title>Famous Scientists</title>
    </head>
    <body>
     <ul>
          <li>Turing,
              Alan</li>
          <li>Feynman,
              Richard</li>
    </ul>
    <p>
              Alan
              Turing

          computer scientist
          mathematician
          cryptographer
    </p>
    <p>
              Richard
              M
              Feynman


          physicist
          Playing the bongoes
    </p>

    </body>
</html>
```

Filtering

- So far we either process all the elements relative to a node or one element
- We need a way to filter out elements as well
- This is done with an XPath control structure

85

Example 13: Filtering

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

   <xsl:template match="people">
        <xsl:apply-templates select="*[not(self::hobby)]" />
   </xsl:template>
</xsl:stylesheet>
```

The *self* keyword is needed to inform the XSLT processor that the node following is a child of the current one

86

Result: Example 13

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

      Alan
      Turing
   computer scientistmathematiciancryptographer

      Richard
      M
      Feynman
    physicist
```

87

Example 14: xsl:for-each

```
<?xml version="1.0"?>
   <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Tr
   ansform">
   <xsl:template match="people">
        <xsl:for-each select="person">
              <xsl:value-of select="name" />
              <xsl:value-of select="@born" />
        </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

88

Result: Example 14

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

      Alan
      Turing
   1912

      Richard
      M
      Feynman
    1918
```

89

Selection

- <xsl:if test="...">
  ...
  </xsl:if>

- <xsl:choose>
  <xsl:when test="..."> ... </xsl:when>
  <xsl:when test="..."> ... </xsl:when>
  ...
  <xsl:otherwise> ... </xsl:otherwise>
  </xsl:choose>

90

Example 15: xsl:if

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

        <xsl:template match="people">
                <xsl:for-each select="person">
                        <xsl:value-of select="name" />
                        <xsl:if test="@born='1912'">
                                Died in
                                <xsl:value-of select="@died" />
                        </xsl:if>
                </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```


Result: Example 15

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


        Alan
        Turing

                Died in
         1954
        Richard
        M
        Feynman
```


Example 16: xsl:choose

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="people">
                <xsl:for-each select="person">
                        <xsl:value-of select="name" />
                        <xsl:choose>
                                <xsl:when test="@born='1912'">
                                    Died in <xsl:value-of select="@died" />
                                </xsl:when>
                                <xsl:otherwise>
                                    Did not die in 1912
                                </xsl:otherwise>
                        </xsl:choose>
                </xsl:for-each>
        </xsl:template>
```


Result: Example 16

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


        Alan
        Turing

                Died in
         1954
        Richard
        M
        Feynman

                Did not die in 1912
```


Sorting

- XSLT can sort the documents by element contents
  - by alphabetical or numerical order, according to attribute values, text contents, in ascending or descending order, and more ...
- Sorting can only be done in the following constructs:
  - <xsl:apply-templates…/>
  - <xsl:for-each …/>
- The construct to use is:

  <xsl:sort select=*selection*></xsl:sort>

  or

  <xsl:sort
  select = *string-expression (sort key)*
  lang = *nmtoken*
  data-type = "text" | "number"
  order = "ascending" | "descending"
  case-order = "upper-first" | "lower-first" />

Example 17: xsl:sort (“Ascending”)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="people">
        <xsl:apply-templates>
                <xsl:sort select="name" />
        </xsl:apply-templates>
   </xsl:template>
</xsl:stylesheet>
```

Result: Example 17 (Sorting)

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

        Alan
        Turing

    computer scientist
    mathematician
    cryptographer

        Richard
        M
        Feyman

    physicist
    Playing the bongoes
```


Example 18: xsl:sort ("Descending")

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

   <xsl:template match="people">

        <xsl:apply-templates>
              <xsl:sort select="name" order="descending" />
        </xsl:apply-templates>

   </xsl:template>

</xsl:stylesheet>
```


Result: Example 18 (Sorting)

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

        Richard
        M
        Feyman

    physicist
    Playing the bongoes

        Alan
        Turing

    computer scientist
    mathematician
    cryptographer
```


Example 19: xsl:copy with xsl:sort ("Descending")

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"

   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" />

 <xsl:template match="*">
  <xsl:copy>
   <xsl:apply-templates>
    <xsl:sort select="name" order="descending" />
   </xsl:apply-templates>
  </xsl:copy>
 </xsl:template>

</xsl:stylesheet>
```


Result: Example 19

```
<?xml version="1.0" encoding="UTF-8"?><people><person>
  <name>
   <first_name>Richard</first_name>
   <middle_initial>M</middle_initial>
   <last_name>Feynman</last_name>
  </name>
  <profession>physicist</profession>
  <hobby>Playing the bongoes</hobby>
 </person><person>
  <name>
   <first_name>Alan</first_name>
   <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
 </person>

</people>
```


Counting

- XSLT can also be used to count or number elements

```
<xsl:number
    level = "single" | "multiple" | "any"
    count = pattern
    from = pattern
    value = number-expression
    format = { string }
    lang = { nmtoken }
    letter-value = { "alphabetic" | "traditional" }
    grouping-separator = { char }grouping-size = { number }
/>
```

- Can implement complex hierarchical numbering schemes

Example 20: count

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

<xsl:template match="person">
  <xsl:for-each select="name">
    <p>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>
    </p>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```


Result: Example 20

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

 <p>1.
    Alan
    Turing
  </p>

 <p>2.
    Richard
    M
    Feynman
  </p>
```


XSLT Template Examples

- Match the root, spit out a canned document

```
<xsl:template match="/">
   <html>
      <body>That's all folks.</body>
   </html>
 </xsl:template>
```

- Change <foo/> elements to <goop>foo</goop> elements.

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


XSL Variables and Parameters

- <xsl:variable name="..." select="..."/>
  - value of select is the value of the variable
  - if no select, then value is the tree contained within the element.
- <xsl:param name="..." select="..."/>
  - value of select if the default value of the parameter, if no other value is specified on call
  - can have top-level params, in which case the way it gets its initial value is implementation defined.
  - can declare inside templates as well

Named templates ...(1)

- <xsl:apply-templates select="..."/>
  - changes the current node and the current node list
- <xsl:call-template name="...">
  <xsl:with-param name="..." select="..."/>
  </xsl:call-template>
  - does not change the current node or the current node list

Named templates ...(2)

- **Matching Templates**

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

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

- **Named Templates**

```
<xsl:template name="foo">
    <xsl:param name="bar" select="'baz'"/>
                ...
</xsl:template>

<xsl:call-template name="foo">
    <xsl:with-param name="bar" select="'not bar'"/>
      ...
</xsl:template>
```

Summary of Query Languages

- Path regular expressions
  - XSLT - yes
  - XQuery - yes
  - XPath - yes
- XML restructuring on output
  - XSLT - yes
  - XQuery - yes
  - XPath - no
- Aggregates
  - XSLT - no
  - XQuery - yes
  - XPath - no
- Text-order preserving
  - XSLT - yes
  - XQuery - yes
  - XPath - yes

109

Summary of Query Languages

- DTD or Schema used
  - XSLT - no
  - XQuery - no
  - XPath - no
- Attribute vs. Elements (node types)
  - XSLT - yes
  - XQuery - no
  - XPath - yes
- Intended use
  - XSLT - Rendering XML content in different styles
  - XQuery - Database sharing in XML, querying of federation
  - XPath - Language for pinpointing, selecting XML text

110

XSLT vs. XQuery

- Querying is useful when you do more than transformation
- Examples
  - Interpreting certain elements as database queries
  - Inserting the query results into output document
  - Asking users questions in the middle of transformation

111

XSLT vs. XQuery

- XQuery and XSLT are both *domain-specific languages* for combing and transforming XML data from multiple sources.
- They are vastly different in design, partly for historical reasons.
- XQuery is designed from scratch, XSLT is an intellectual descendant of CSS.
- Technically, they emulate each other.

112

Summary: XSLT 2.0

- Nodes
  - root node
  - Element nodes
  - Attribute nodes
  - Text nodes
  - Comment nodes
  - ...
- Elements
  - xsl:stylesheet
  - xsl:apply-templates, xsl:call-template,
  - xsl:template, xsl:with-param
  - xsl:for-each
  - xsl:value-of

113

Summary: XSLT 2.0

- XSLT directives are evaluated and replaced by result
  - Scalar expressions - result in a single value
    - Value-of gets the value of a node
      <xsl:value-of select="*XPath Expression*"/>
    - Copy-of makes a copy of a node
      <xsl:copy-of select="*XPath Expression*"/>
  - Loops - loop is evaluated for each matching node

    <xsl:foreach select="*XPath List Expression*">
    ...body...
    </xsl:foreach>
  - Conditionals

    <xsl:if test="*comparison*">...body...</xsl:if>

114

Summary: XSLT 2.0

- **Applying Templates**
  - **<xsl:apply-templates select="..." />**
- **Built-In Templates**

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

<xsl:template match="text() | @*">
    <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="comment() | processing-instruction()" />
```

- **Values of Nodes**
  - **<xsl:value-of select="..." />**

References

- **The W3c Official Web site that includes the complete reference of XSLT:**
  http://www.w3.org/TR/xslt
- **A simple easy to follow XSLT notes: http://www.cse.ohio-state.edu/~gurari/course/cis788/cis788se12.html#cis788su91.html**
- **XSL FAQ: An excellent FAQ resource**
  http://www.dpawson.co.uk/xsl/
- Text book: XML in a nutshell, Harold & Means
- Text book: Learning XSLT, Fitzerald M