XML and Databases

Lecture 11
XSLT – Stylesheets and Transforms

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2010

Outline

1. eXtensible Stylesheet Language Transformations (XSLT)
2. Templates: (match pattern → action)
3. Default Templates
4. Conflict Resolution
5. Modes and Variables

XML

→ General data format
→ Includes no formatting information (other than linebreaks etc in text nodes)

Data Model (XPath)
→ Seven types of nodes (doc, elem, text, attr, ns, com, pi)

XML, typical usage scenario

<Product>
  <product_id> d101 </product_id>
  <name> Sony discman </name>
  <currency> AUD </currency>
  <price> 169.00 </price>
  <gst> 10% </gst>
</Product>

... 

1. eXtensible Stylesheet Language

"stylesheet" = recipe how to display your XML data

  e.g. “display titles of books in bold, large font”
  "display authors of books in italic, medium size"
  etc.

→ Choose an output format (e.g. XHTML, HTML, text, PDF, etc.)
→ Transform the XML:
  - add formatting information
  - rearrange data
  - sorting
  - delete certain parts, copy others
  etc.

→ Store data once, in most general format
  (free tech writers from bothering with layout issues)

→ Reuse fragments of the data (same contents; looks different depending on the context)

E.g. different output formats (size, device)
  style tailored to reader’s preference
  style tailored to adhere w. corporate/web site identity
Example Transformations

- XML to HTML — for old browsers
- XML to LaTeX — for TeX layout
- XML to SVG — graphs, charts, trees
- XML to tab-delimited — for db/stat packages
- XML to plain-text — occasionally useful
- XML to FO — XSL formatting objects (mostly used to generate PDF)

This slide and some examples in the end are taken from David G. Durand’s “Introduction to XSLT” http://www.cs.brown.edu/~dgd/xslt-tutorial.ppt

1. eXtensible Stylesheet Language

XSLT:
- XSLT takes
  - a “source” XML document
  - a transform (XSLT program)
- XSLT applies templates to matched nodes
  - may delete or include the rest
  - may process in document or tree or any order
- XSLT generates
  a “result” XML or text document

XSL  -- developed by W3C , quite old!
XSL Transformations (XSLT)
Version 1.0
W3C Recommendation 16 November 1999

→ XSL Transformations are written in XML themselves.
Element name prefix “xsl:” indicates an XSLT specific command
→ functional programming style
→ pattern matching as basic construct

XSL Stylesheets

- An XSL stylesheet defines a set of templates (“tree patterns and actions”).
- Each template...
  - matches specific elements in the XML doc tree, and then
  - constructs the contribution that the elements make to the transformed tree.
- XSL is an application of XML itself:
  - Each XML stylesheet is an XML document,
  - elements with a name prefix “xsl:” are part of the XSLT language,
  - non-“xsl:” elements are used to construct the transformed tree.

More correctly: elements in the namespace http://www.w3.org/1999/XSL/_transforms. For details on namespaces, see http://www.w3.org/TR/REC-xml-names.

Example: Transform text markup into HTML style paragraph and emphasis tags:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output omit-xml-declaration="yes" indent="yes"/>
  
  <xsl:template match="p">
    <p>
      <xsl:apply-templates/>
    </p>
  </xsl:template>

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

  <xsl:output method="xml" encoding="UTF-8"/>
</xsl:stylesheet>
```

N.B. Note how XSLT acts like a tree transformer in this simple example.
What goes in a template?

- Literal XML to output
- "Pull" references to other content
- Instructions to generate more output
  - Setting and using variables
  - Invoking other templates like macros
  - Manually constructed XML output
  - Conditional instructions (if, choose, etc.)
  - Auto-numbering hacks

Default templates

Each XSLT stylesheet contains two default templates which:

1. copy Text and Attr (attribute) nodes into the result tree:
   `<xsl:template match="text()"/>
   `<xsl:value-of select="self::model()"/>
   </xsl:template>`

2. recursively drive the matching process, starting from the
document root:
   `<xsl:template match="/">  Generates an output root node.
   `<xsl:apply-templates/>
   `<xsl:value-of select="e"/>  copies those nodes into the result tree
   that are reachable by the XPath expression e (context node is the
   matched node).

The default templates may be overridden.

Overiding default XSLT templates

What would be the effect of applying the following XSLT stylesheet?

```xml
<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>
<!-- a comment -->
<para>This is a <emphasis>test</emphasis>.</para>
```

More XSLT defaults

XSLT contains the following additional default template. Explain its effect.

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

Overiding default XSLT templates

What would be the effect of applying the following XSLT stylesheet?

```xml
<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>
<!-- a comment -->
<para>This is a <emphasis>test</emphasis>.</para>
```

More XSLT defaults

XSLT contains the following additional default template. Explain its effect.

```xml
<xsl:template match="processing-instruction()">comment()
```
**Question**

Which default rules are overridden through this match expression?

**Question**

Which default rules are overridden through this match expression?

**Question**

Which default rules are overridden through this match expression?

**7.6.1 Generating Text with xsl:value-of**

The xsl:value-of element is instantiated to create a text node in the result tree. The required select attribute is an expression; this expression is evaluated and the resulting object is converted to a string as if by a call to the string function. The string specifies the string-value of the created text node. If the string is empty, no text node will be created. The created text node will be merged with any adjacent text nodes. The xsl:copy-of element can be used to copy a node-set over to the result tree without converting it to a string. See [11.3 Using Values of Variables and Parameters with xsl:copy-of].

**Running the empty stylesheet**

```
Running the empty stylesheet
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
</xsl:stylesheet>
```

On this document
```
<?xml version="1.0"?>
<para size="5pt">This is a <emphasis>test</emphasis>.</para>
```

```
smaneth@comqu:~/xslt$ xsltproc empty.xslt 1.xml
<?xml version="1.0"?>
This is a test.
```

Not XML, of course... Root node is generated by default root-rule.
The string-values of a PI or comment nodes are their "contents".

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>
<!-- a comment -->
<a h="99">
<para>This is a <emphasis u="m" f="a"><u>t</u>est</emphasis>.</para>
</a>

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=...>
<xsl:template match="node()">
<xsl:value-of select="self::node()"/>
</xsl:template>
</xsl:stylesheet>

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>
<!-- a comment -->
<para at1="val1">This is a <emphasis>test</emphasis>.</para>

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?><!-- a comment -->
<para at1="val1">This is a <emphasis>test</emphasis>.</para>

White-space deleted.

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?><!-- a comment -->
<para at1="val1">This is a <emphasis>test</emphasis>.</para>

Do not override default @-rule.

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>

<?xml version="1.0"?>
<page number="1"><p at1="val1">This is a <em>test</em>.</p></page>

Do not override default @-rule.

<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>

<?xml version="1.0"?>
<page number="1"><p at1="val1">This is a <i>test</i>.</p></page>

No attribute nodes are ever translated...
⇒ They all disappear in the output.
We can use XSLT to generate formatted answers to queries. E.g., of a simple XPath expression:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
    <report>
      <title>Expense Report Summary</title>
      <amount><xsl:value-of select="expense-report/total"/></amount>
    </report>
  </xsl:template>
</xsl:stylesheet>
```

What happens if this template is applied?

Thus: XSLT can be used as a query language for XML!

Of course, instead of translating XML to HTML, we can also translate to XML again.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<report>
  <title>Expense Report Summary</title>
  <amount><xsl:value-of select="expense-report/total"/></amount>
</report>
```

What happens if this template is applied??

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<report>
  <title>Expense Report Summary</title>
  <amount><xsl:value-of select="expense-report/total"/></amount>
</report>
```

Remember from XPath: an empty node sequence is interpreted as false, a non-empty sequence as true.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
    <report>
      <title>Expense Report Summary</title>
      <amount><xsl:value-of select="expense-report/total"/></amount>
    </report>
  </xsl:template>
</xsl:stylesheet>
```
Question

How do you remove book/author/first elements, but keep everything else in the XML?
Question

Assume the XML input has precisely one element node with name AA and one element node with name BB (independent).

Can you give an XSL transform which exchanges the first subtree of AA with the last subtree of BB?
Variables & Parameters
A variable-binding element can specify the value of the variable in three alternative ways.

1. If the variable-binding element has a select attribute, then the value of the attribute must be an expression and the value of the variable is the object that results from evaluating the expression. In this case, the content must be empty.

2. If the variable-binding element does not have a select attribute and has non-empty content (i.e., the variable-binding element has one or more child nodes), then the content of the variable-binding element specifies the value. The content is a template, which is instantiated to give the value of the variable. The value is a result tree fragment equivalent to a node-set containing just a single root node having as children the sequence of nodes produced by instantiating the template.

3. If the variable-binding element has empty content and does not have a select attribute, then the value of the variable is an empty string.

This example declares a global variable para-font-size, which it references in an attribute value template.

```xml
<xsl:variable name="para-font-size">12pt</xsl:variable>
<xsl:template match="para">
  <fo:block font-size="{$para-font-size}">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Numbering example

```xml
<xsl:template select="list">
  <xsl:attribute name="type="unordered">1</xsl:attribute>
</xsl:template>
```

<xsl:library name="xsl:sequence">
  <xsl:variable name="x" select="''"/>

  <xsl:variable name="n" select="2"/>
  <xsl:value-of select="item[$n]"/>

  <xsl:variable name="n">2</xsl:variable>
  <xsl:value-of select="item[position()=$n]"/>

  <xsl:number level="multiple" count="pattern" from="pattern" value="number-expr" format="s" lang="lg" letter-value="alphabetic|traditional" grouping-separator="char" grouping-size="number" />
</xsl:library>
What will be the result?