

XML and Databases

Lecture 11
XSLT – Stylesheets and Transforms

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2008

Outline

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

2

XML

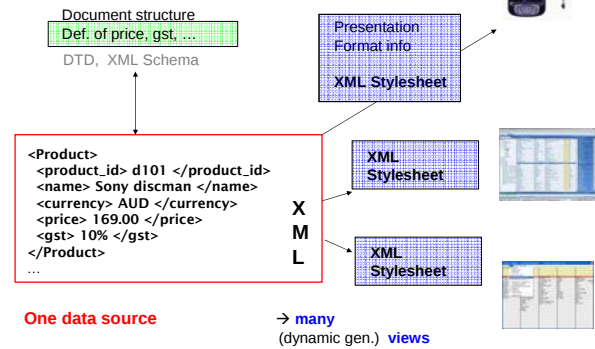
- General data format
- Includes no formatting information (other than linebreaks etc in text nodes)

Data Model (XPath)

- Seven types of nodes (doc, elem, text, attr, ns, com, pi)

3

XML, typical usage scenario



4

XML

- General data format
- Includes no formatting information

Usage Idea

- Store data *once*, in general most 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

5

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 etc

6

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>

7

1. eXtensible Stylesheet Language

- XSLT takes
 - A "source" XML document
 - A transform (XSLT program)
- XSLT applies templates to found nodes
 - (may delete or include A "result" XML or text document the rest)
 - (may process in document or tree or any order)
- XSLT generates

8

Google AJAXSLT

→ to play around a bit with XSLT, you can use Google AJAXSLT

<http://goog-ajaxslt.sourceforge.net>

Google AJAXSLT

AJAXSLT is an implementation of XSL-T in JavaScript, intended for use in fat web pages, which are nowadays referred to as AJAX applications. Because XSL-T uses XPath, it is also an implementation of XPath that can be used independently of XSL-T.

[code.google.com](#)
[Project Page](#)
[Downloads](#)
[News](#)

Overview

XSL-T stands for [XSL Transformations](#). XSL stands for [eXtensible Stylesheet Language](#).

XSL-T is a language for transforming XML documents from one language to another. An XSL-T style sheet would be used, for instance, to convert the unformatted content from an XML document into the a fully-formatted HTML document.

AJAXSLT takes this process one step forward, by implementing XSL-T in JavaScript and having it run in your browser. Thus, your web browser can fetch XML documents directly from the server, and perform the format conversion locally, thus saving time and bandwidth.

AJAXSLT is distributed under the terms of the [BSD License](#).

For downloads, news, and other information, visit our [Project Page](#)

Download

For downloads, visit our [Project Page](#)

News

- v0.5 released at [code.google.com](#), Nov 2006
- v0.4 released, Oct 2005; see [ChangeLog](#)
- v0.2 released, June 2005; see [ChangeLog](#)
- Initial Launch, June 2005

See the [Project Page](#) for news archives.

Google Groups

- [google-ajaxslt-discuss](#)
- [codebase-discuss](#) – general discussion

9

Google AJAXSLT

→ to play around a bit with XSLT, you can use Google AJAXSLT

<http://goog-ajaxslt.sourceforge.net>

Google AJAXSLT

AJAXSLT is an implementation of XSL-T in JavaScript, intended for use in fat web pages, which are nowadays referred to as AJAX applications. Because XSL-T uses XPath, it is also an implementation of XPath that can be used independently of XSL-T.

[code.google.com](#)
[Project Page](#)
[Downloads](#)
[News](#)

Overview

XSL-T stands for [XSL Transformations](#). XSL stands for [eXtensible Stylesheet Language](#).

XSL-T is a language for transforming XML documents from one language to another. An XSL-T style sheet would be used, for instance, to convert the unformatted content from an XML document into the a fully-formatted HTML document.

AJAXSLT takes this process one step forward, by implementing XSL-T in JavaScript and having it run in your browser. Thus, your web browser can fetch XML documents directly from the server, and perform the format conversion locally, thus saving time and bandwidth.

AJAXSLT is distributed under the terms of the [BSD License](#).

For downloads, news, and other information, visit our [Project Page](#)

Download

For downloads, visit our [Project Page](#)

News

- v0.5 released at [code.google.com](#), Nov 2006
- v0.4 released, Oct 2005; see [ChangeLog](#)
- v0.2 released, June 2005; see [ChangeLog](#)
- Initial Launch, June 2005

See the [Project Page](#) for news archives.

Google Groups

- [google-ajaxslt-discuss](#)
- [codebase-discuss](#) – general discussion

NO!
I found this buggy.
Better use
"xsltproc"
(command line tool for linux)

10

1. eXtensible Stylesheet Language

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

11

XSLT—An XML Presentation Processor

XSL Stylesheets

Template pattern → action

- An **XSL stylesheet** defines a set of **templates** ("tree patterns and actions").

Each template ...

- ① **matches** specific elements in the XML doc tree, and then
- ② **constructs** the contribution that the elements make to the transformed tree.

- XSL is an **application of XML** itself:
 - ▶ Each XSL stylesheet is an XML document,
 - ▶ elements with a **name prefix**³⁴ xsl: are part of the XSLT language,
 - ▶ non-"xsl:" elements are used to construct the transformed tree.

³⁴More correctly: elements in the **namespace**
<http://www.w3.org/1999/XSL/Transform>. For details on namespaces, see <http://www.w3.org/TR/REC-xml-names>.

Marc H. Schohl (DBIS, Uni Kln) XML and Databases Winter 2005/06 227

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

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4 <xsl:template match="para">
5   <p><xsl:apply-templates/></p>
6 </xsl:template>
7
8 <xsl:template match="emphasis">
9   <i><xsl:apply-templates/></i>
10 </xsl:template>
11
12 </xsl:stylesheet>

```

Seems implementation dependent..
xsltproc doesn't generate it.

```

1 <?xml version="1.0"?>
2 <para>This is a <emphasis>test</emphasis></para>

```

input.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <p>This is a <i>test</i></p>

```

output.xml

N.B. Note how XSLT acts like a **tree transformer** in this simple example.

XSLT templates

```

<xsl:template match="e">
  cons
</xsl:template>

```

- e** is an **XPath expression**, selecting the nodes in the document tree XSLT will apply the template to.
- cons** is the **result constructor**, describing the transformation result that the XSLT processor will produce for the nodes selected by **e**.

N.B. "xsl:" elements in **cons** will be interpreted by the XSLT processor.

<xsl:apply-templates/> applies the template matching process **recursively to all child nodes** of the matched node.

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 constructs
 - Conditional instructions (if, choose, etc.)
 - Auto-numbering hacks

Applying the template ...

The actual tree transformation in our previous example goes like this:

Something else must be going on here:

- The **Text** nodes have *automatically* been copied into the result tree.
- How could the **para** and **emphasis** elements match anyway? (The XPath patterns for both templates used relative paths expressions.)

Default templates

Each XSLT stylesheet contains two **default templates** which

- copy Text and Attr (attribute) nodes** into the result tree:


```

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

```
- recursively drive the matching process**, starting from the document root:


```

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

```

generates concatenation of the string values of e's nodes

<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**.

Overriding default XSLT templates

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

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4 <xsl:template match="text()">foo</xsl:template>
5
6 <xsl:template match="para">
7   <p><xsl:apply-templates/></p>
8 </xsl:template>
9
10 <xsl:template match="emphasis">
11   <i><xsl:apply-templates/></i>
12 </xsl:template>
13
14 </xsl:stylesheet>

```

More XSLT defaults

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

```

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

```

Overriding default XSLT templates

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

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4 <xsl:template match="text()">foo</xsl:template>
5
6 <xsl:template match="para">
7   <p><xsl:apply-templates/></p>
8 </xsl:template>
9
10 <xsl:template match="emphasis">
11   <i><xsl:apply-templates/></i>
12 </xsl:template>
13
14 </xsl:stylesheet>

```

every text node becomes "foo"

More XSLT defaults

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

```

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

```

Marc H. Schoff (DBIS, Uni Kln) XML and Databases Winter 2005/06 222

Overriding default XSLT templates

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

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4 <xsl:template match="text()">foo</xsl:template>
5
6 <xsl:template match="para">
7   <p><xsl:apply-templates/></p>
8 </xsl:template>
9
10 <xsl:template match="emphasis">
11   <i><xsl:apply-templates/></i>
12 </xsl:template>
13
14 </xsl:stylesheet>

```

every text node becomes "foo"

Are deleted!!

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

```

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

```

Marc H. Schoff (DBIS, Uni Kln) XML and Databases Winter 2005/06 222

Overriding default XSLT templates

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

```

1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3
4 <xsl:template match="text()">foo</xsl:template>
5
6 <xsl:template match="para">
7   <p><xsl:apply-templates/></p>
8 </xsl:template>
9
10 <xsl:template match="emphasis">
11   <i><xsl:apply-templates/></i>
12 </xsl:template>
13
14 </xsl:stylesheet>

```

every text node becomes "foo"

Are deleted!!

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

```

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

```

Marc H. Schoff (DBIS, Uni Kln) XML and Databases Winter 2005/06 222

Question

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="emphasis">
    <i>TEXT<xsl:apply-templates select="@*|node()"/>TEXT2</i>
  </xsl:template>
  <xsl:template match="node()">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
  </xsl:template>
</xsl:stylesheet>

```

Which default rules are overridden through this match expression?

22

Question

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="emphasis">
    <i>TEXT<xsl:apply-templates select="@*|node()"/>TEXT2</i>
  </xsl:template>
  <xsl:template match="node()">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
  </xsl:template>
</xsl:stylesheet>

```

Which default rules are overridden through this match expression?

```

<?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>

```

What is the output, if above XSLT transformation is applied?

23

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 denotes 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](#).

24

Running the empty stylesheet

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

On this document

```
<?xml version="1.0"?>
<?php sql (SELECT * FROM ) ?>
<!-- a comment -->
<a h="99">
  <para size="5pt">This is a <emphasis>test</emphasis>.</para>
</a>
```

smameth@comqu:~/xslt\$ xsltproc empty.xslt 1.xml

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

This is a test.

smameth@comqu:~/xslt\$

Not XML, of course...
Root node is generated by default root-rule.

25

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
<xsl:value-of select="self::node()"/>
</xsl:template>
```

Change "self" into "parent"

What is the corresponding result for this input?

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

26

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
<xsl:value-of select="self::node()"/>
</xsl:template>
```

Change "self" into "parent"

What is the corresponding result for this input?

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

Here, generate the string-value
of the parent node (= "This is a test.")

27

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
<xsl:value-of select="self::node()"/>
</xsl:template>
```

Change "self" into "parent"

What is the corresponding result for this input?

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

Here, generate the string-value
of the parent node (= "This is a test.")

→ We get

```
<?xml version="1.0"?>
<p>This is a test.<i>test</i>This is a test.</p>
```

28

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
<xsl:value-of select="self::node()"/>
</xsl:template>
```

Change "self" into "parent::*parent"

And now?

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

29

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
<xsl:value-of select="self::node()"/>
</xsl:template>
```

Change "self" into "parent::*parent"

And now?

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

↓

```
<?xml version="1.0"?>
<p>This is a test.<i>This is a test.</i>This is a test.</p>
```

30

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
  <xsl:value-of select="self::node()" />
</xsl:template>
```

Change "self" into "parent::/parent"

And now?

What if there is an attribute `at1="val1"` here?

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

```
<?xml version="1.0"?>
<p>This is a test.<i>This is a test.</i>This is a test.</p>
```

31

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
  <xsl:value-of select="self::node()" />
</xsl:template>
```

Change "self" into "parent::/parent"

And now?

What if there is an attribute `at1="val1"` here?

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

```
<?xml version="1.0"?>
<p>This is a test.<i>This is a test.</i>This is a test.</p>
```

→ Same output. Attributes are deleted!

32

Question

(para and emphasis templates as before)

```
<xsl:template match="text()">
  <xsl:value-of select="self::node()" />
</xsl:template>
```

Change "self" into "parent::/parent"

And now?

What if there is an attribute `at1="val1"` here?

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

How can we copy para's attributes to the p-element?

```
<?xml version="1.0"?>
<p>This is a test.<i>This is a test.</i>This is a test.</p>
```

→ Same output. Attributes are deleted!

33

Intermediate summary

XSLT Instruction	Effect
<code><xsl:template match="e"> cons </xsl:template></code>	Replace nodes matching path expression <i>e</i> by <i>cons</i> .
<code><xsl:apply-templates [select="e"]></code>	Initiate template matching for those nodes returned by path expression <i>e</i> (default path <i>e</i> = <code>child::node()</code>).
<code><xsl:value-of select="e" /></code>	Returns the (string value ³⁵ of the) result of XPath expression <i>e</i> .

³⁵Read: The string value of an XML element node is the concatenation of the contents of all Text nodes in the subtree below that element (in document order).

Manv H. Scholl / FHOBS, Lin. KN, XML and XPath, Winter 2005/06 333

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="a" f="a">test</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"?>
sql (SELECT * FROM ) a comment
This is a test.
```

35

7.5 Copying

The `xsl:copy` element provides an easy way of copying the current node. Instantiating the `xsl:copy` element creates a copy of the current node. The namespace nodes of the current node are automatically copied as well, but the attributes and children of the node are **not** automatically copied.

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

← Identity transformation
Copies all nodes as is into the output.

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

White-space deleted.
Why?

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

36

7.5 Copying

The `xsl:copy` element provides an easy way of copying the current node. Instantiating the `xsl:copy` element creates a copy of the current node. The namespace nodes of the current node are automatically copied as well, but the attributes and children of the node are **not** automatically copied.

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

← Identity transformation
Copies all nodes as is into the output.

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

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

```
<?xml 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>
```

43

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

  <xsl:template match="text()">
    <xsl:value-of select="parent::node()"/>
  </xsl:template>

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

  <xsl:template match="emphasis">
    <i><xsl:apply-templates select="self::node()"/></i>
  </xsl:template>

</xsl:stylesheet>
```

What happens if this template is applied??

44

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

  <xsl:template match="text()">
    <xsl:value-of select="parent::node()"/>
  </xsl:template>

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

  <xsl:template match="emphasis">
    <i><xsl:apply-templates select="self::node()"/></i>
  </xsl:template>

</xsl:stylesheet>
```

smraneth@ComQu-Desktop:~\$ xsltproc 1.xslt 2.xml
runtime error: file 1.xslt line 13 element in
xsltApplyXSLTTemplate: A potential infinite template recursion was detected.
You can adjust xsltMaxDepth (-maxdepth) in order to raise the maximum number of
nested template calls and variables/params (currently set to 3000).
Templates:
#0 name emphasis
#1 name emphasis

45

Example: Dilbert comic strips ...

Transform a DilbertML document into an HTML representation that reflects the comic strip's story:

- From the prolog, generate the HTML header, title, heading, copyright information.
- From characters, generate an unordered HTML list (ul) of all featured comic characters.
- For all panels, reproduce the scene as well as all spoken bubbles, indicating who is speaking to whom (if available).

Note:

<xsl:if test="p"/> cons </xsl:if> reproduces cons in the result tree, if the XPath predicate *p* evaluates to true.³⁶

³⁶Remember from XPath: an empty node sequence is interpreted as false, a non-empty sequence as true.

Marc H. Schohl (DBIS, Uni KN)

XML and Databases

Winter 2005/06

235

```
1 <?xml version="1.0"?>
2 <xsl:stylesheet version="1.0"
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5 <!-- Generate document head and body, insert prolog information -->
6 <xsl:template match="/">
7   <html>
8     <head>
9       <title>
10        <xsl:value-of select="/strip/prolog/series"/>
11      </title>
12    </head>
13    <body>
14      <h1> <xsl:value-of select="/strip/prolog/series"/>
15      </h1>
16      <p>A comic series by
17        <xsl:value-of select="/strip/prolog/author"/>,
18        copyright (C)
19        <xsl:value-of select="/strip/@year"/> by
20        <xsl:value-of select="/strip/@copyright"/>
21      </p>
22      <xsl:apply-templates/>
23    </body>
```

Marc H. Schohl (DBIS, Uni KN)

XML and Databases

Winter 2005/06

236

```
24 </html>
25 </xsl:template>
26
27 <!-- The next 2 templates generate the
28   "Featured Characters" bullet list -->
29 <xsl:template match="characters">
30   <h2> Featured Characters </h2>
31   <ul>
32     <xsl:apply-templates/>
33   </ul>
34 </xsl:template>
35
36 <xsl:template match="character">
37   <li> <xsl:value-of select="."/> </li>
38 </xsl:template>
39
40 <!-- Reproduce the panel and the scene it displays -->
41 <xsl:template match="panel">
42   <h3> Panel <xsl:value-of select="@no"/> </h3>
43   <p> <xsl:value-of select="scene"/> </p>
44   <xsl:apply-templates select="bubbles"/>
45 </xsl:template>
46
```

Marc H. Schohl (DBIS, Uni KN)

XML and Databases

Winter 2005/06

237

XSLT—An XML Presentation Processor Examples

```

47 <!-- Reproduce spoken text, indicating tone and
48 who is speaking to whom -->
49 <xsl:template match="bubble">
50 <p> <xsl:value-of select="id(@speaker)"/> speaking
51 <xsl:if test="@to">
52 to <xsl:value-of select="id(@to)"/>
53 </xsl:if>
54 <xsl:if test="@tone">
55 (<xsl:value-of select="@tone"/>)
56 </xsl:if>
57 <br/>
58 <em>
59 <xsl:value-of select="."/>
60 </em>
61 </p>
62 </xsl:template>
63
64 <!-- Suppress all other text/attributes -->
65 <xsl:template match="text()|@*" />
66
67 </xsl:stylesheet>

```

Marc H. Schoell (DBIS, Uni KN) XML and Databases Winter 2005/06 238

XSLT—An XML Presentation Processor Examples

dilbert.html

```

1 <html>
2 <head> <title>Dilbert</title> </head>
3 <body>
4 <h1>Dilbert</h1>
5 <p>A comic series by Scott Adams, copyright
6 (C) 2000 by United Feature Syndicate </p>
7 <h2> Featured Characters </h2>
8 <ul>
9 <li>The Pointy-Haired Boss</li>
10 <li>Dilbert, The Engineer</li>
11 <li>Wally</li>
12 <li>Alice, The Technical Writer</li>
13 </ul>
14 <h3> Panel 1</h3>
15 <p>Pointy-Haired Boss pointing to presentation slide.
16 </p>
17 <p>The Pointy-Haired Boss speaking : <br>
18 <em>Speed is the key to success.</em>
19 </p>
20 <h3> Panel 2</h3>
21 <p>Wally, Dilbert, and Alice sitting at conference table.
22 </p>
23 <p>Dilbert, The Engineer speaking

```

Marc H. Schoell (DBIS, Uni KN) XML and Databases Winter 2005/06 239

XSLT—An XML Presentation Processor Examples

```

24 to The Pointy-Haired Boss : <br>
25 <em>Is it ok to do things wrong if
26 we're really, really fast?</em>
27 </p>
28 <h3> Panel 3</h3>
29 <p>Wally turning to Dilbert, angrily.
30 </p>
31 <p>The Pointy-Haired Boss speaking
32 to Dilbert, The Engineer : <br>
33 <em>Um... No.</em>
34 </p>
35 <p>Wally speaking
36 to Dilbert, The Engineer (angry) : <br>
37 <em>Now I'm all confused. Thank you very much.</em>
38 </p>
39 </body>
40 </html>

```

Marc H. Schoell (DBIS, Uni KN) XML and Databases Winter 2005/06 240

XSLT—An XML Presentation Processor Examples

Screenshot of Mozilla rendering file dilbert.html:

Marc H. Schoell (DBIS, Uni KN) XML and Databases Winter 2005/06 241

XSLT—An XML Presentation Processor Conflict Resolution and Modes

Conflict Resolution and Modes in XSLT

- Note that for each node visited by the XSLT processor (cf. default template ②), **more than one template might yield a match**.
- XSLT assigns a **priority** to each template. The more specific the template pattern, the higher the priority:


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

Pattern e	Priority
*	-0.5
ns:*	-0.25
element/attribute name	0
any other XPath expression	0.5
- Example:** Priority of author is 0, priority of /strip/prolog/author is 0.5.
- Alternatively, make priority explicit: `<xsl:template priority="p" ...>` any real number.

Marc H. Schoell (DBIS, Uni KN) XML and Databases Winter 2005/06 242

Delete all nested <list>s

```

<xsl:template match="list/list"
  priority="2">
  <!-- deleted nested list -->
</xsl:template>

<xsl:template match="list"
  priority="1">
  <list><xsl:apply-templates/></list>
</xsl:template>

```

What if there are no priorities?

```

<d>
<list>text1
</list>
blah
</list>
</list>
next
</list>
</d>

```

→

```

<?xml version="1.0"?>
<list>text1
</list>
</list>
next
</list>

```

54

Delete all nested <list>s

```
<xsl:template match="//list/list">
  <!-- deleted nested list -->
</xsl:template>

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

And now?

```
<d>
  <list>text1
  </list>
  blah
</list>
</list>
</list>
next
</list>
</d>
```



```
<?xml version="1.0"?>
<list>text1
</list>
</list>
next
</list>
```

What if there are no priorities?

55

Delete all nested <list>s

```
<xsl:template match="//list/list">
  <!-- deleted nested list -->
</xsl:template>

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

Does NOT generate a comment node!
→ use
<xsl:comment>...</xsl:comment>

```
<d>
  <list>text1
  </list>
  blah
</list>
</list>
</list>
next
</list>
</d>
```



```
<?xml version="1.0"?>
<list>text1
</list>
</list>
next
</list>
```

What if there are no priorities?

56

Question

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

57

Question

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

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

  <xsl:template match="book/author/first" priority="5">
    </xsl:template>

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

</xsl:stylesheet>
```

```
<bib>
  <book>
    <author>
      <first>
        Jeffrey
      </first>
      <last>
        Ullman
      </last>
    </author>
    <title>
      blah2
    </title>
  </book>
</bib>
```



```
<bib>
  <book>
    <author>
      Ullman
    </author>
    <title>
      blah2
    </title>
  </book>
</bib>
```

58

Context

Quite often, an XSLT stylesheet wants to be **context-aware**.

- Since the XSLT priority mechanism is *not* dynamic, this can cause problems.

Example: Transform the following XML document (sectioned text with cross references) into XHTML:

```
self-ref.xml
1 <section id="intro">
2   <title>Introduction</title>
3   <para> This section is self-referential: <xref to="intro"/>. </para>
4 </section>
```

We want to generate XHTML code that looks somewhat like this:

```
self-ref.html
1 <h1>Introduction</h1>
2 <p> This section is self-referential: <em>Introduction</em>. </p>
```

The section title needs to be processed twice, once to produce the heading and once to produce the cross reference.

The "obvious" XSLT stylesheet produces erroneous output:

```
buggy-self-ref.xml
1 <?xml version="1.0"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3   version="1.0">
4
5 <xsl:template match="title">
6   <h1><xsl:apply-templates/></h1>
7 </xsl:template>
8
9 <xsl:template match="para">
10  <p><xsl:apply-templates/></p>
11 </xsl:template>
12
13 <xsl:template match="xref">
14  <xsl:apply-templates select="id(@to)/title"/>
15 </xsl:template>
16
17 </xsl:stylesheet>

buggy-output.html
1 <h1>Introduction</h1>
2 <p> This section is self-referential: <h1>Introduction</h1>. </p>
```

XSLT—An XML Presentation Processor Conflict Resolution and Modes

XSLT modes

- We need to make the processing of the **title** element aware of the context (or **mode**) it is used in: inside an **xref** or not.
- This is a job for **XSLT modes**.
 - In `<xsl:apply-templates>` switch to a certain mode *m* depending on the context:


```
<xsl:apply-templates mode="m" .../>
```
 - After mode switching, only `<xsl:template>` instructions with a mode attribute of value *m* will match:


```
<xsl:template mode="m" .../>
```
 - As soon as `<xsl:apply-templates mode="m" .../>` has finished matching nodes, the previous mode (if any) is restored.

Marc H. Schohl (DBIS, Uni KN) XML and Databases Winter 2005/06 245

XSLT—An XML Presentation Processor Conflict Resolution and Modes

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes" self-ref="xsl"
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    version="1.0">
4    <xsl:template match="title">
5      <h1><xsl:apply-templates/></h1>
6    </xsl:template>
7
8    <xsl:template match="title" mode="ref">
9      <em><xsl:apply-templates/></em>
10    </xsl:template>
11
12    <xsl:template match="xref">
13      <xsl:apply-templates select="id(@to)/title" mode="ref"/>
14    </xsl:template>
15
16  </xsl:stylesheet>
  
```

output.html

```

1  <h1>Introduction</h1>
2  <p>This section is self-referential: <em>Introduction</em>. </p>
  
```

Marc H. Schohl (DBIS, Uni KN) XML and Databases Winter 2005/06 246

XSLT—An XML Presentation Processor More on XSLT

More on XSLT

XSLT Instruction	Effect
<code>xsl:choose</code> , <code>xsl:when</code>	switch statement (ala C)
<code>xsl:call-template</code>	explicitly invoke a (named) template
<code>xsl:for-each</code>	replicate result construction for a sequence of nodes
<code>xsl:import</code>	import instructions from another stylesheet
<code>xsl:output</code>	influence XSLT processor's output behaviour
<code>xsl:variable</code>	set/read variables

- For a complete XSLT reference, refer to [W3C http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt)
- Apache's Cocoon is an XSLT-enabled web server (see <http://xml.apache.org/cocoon/>).

Marc H. Schohl (DBIS, Uni KN) XML and Databases Winter 2005/06 247

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**?

64

What about this one?

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="AA/*[1]" priority="5">
    <xsl:apply-templates select="//BB/*[position()=last()]" />
  </xsl:template>
  <xsl:template match="BB/*[position()=last()]" priority="5">
    <xsl:apply-templates select="//AA/*[1]" />
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
  
```

65

What about this one?

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="AA/*[1]" priority="5">
    <xsl:apply-templates select="//BB/*[position()=last()]" />
  </xsl:template>
  <xsl:template match="BB/*[position()=last()]" priority="5">
    <xsl:apply-templates select="//AA/*[1]" />
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
  
```

⊗ This loops forever...

66

```

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

<xsl:template match="AA/*[1]" priority="5">
  <xsl:apply-templates select="//BB/*[position()=last()]" mode="c"/>
</xsl:template>

<xsl:template match="BB/*[position()=last()]" priority="5">
  <xsl:apply-templates select="//AA/*[1]"/>
</xsl:template>

<xsl:template match="@*|node()" priority="5" mode="c">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

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

</xsl:stylesheet>

```

"copy" mode
(does not
select
//AA/*[1]
again..!)

67

```

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

<xsl:template match="AA/*[1]" priority="5">
  <xsl:apply-templates select="//BB/*[position()=last()]" mode="c"/>
</xsl:template>

<xsl:template match="BB/*[position()=last()]" priority="5">
  <xsl:apply-templates select="//AA/*[1]"/>
</xsl:template>

<xsl:template match="@*|node()" priority="5" mode="c">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="AA/*[1]" priority="5">
  <xsl:copy>
    <xsl:apply-templates select="//BB/*[position()=last()]" mode="c"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="BB/*[position()=last()]" priority="5">
  <xsl:copy>
    <xsl:apply-templates select="//AA/*[1]"/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>

```

"copy" mode
(does not
select
//AA/*[1]
again..!)

68

```

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

<xsl:template match="AA/*[1]" priority="5">
  <xsl:apply-templates select="//BB/*[position()=last()]" mode="c"/>
</xsl:template>

<xsl:template match="BB/*[position()=last()]" priority="5">
  <xsl:apply-templates select="//AA/*[1]" mode="c"/>
</xsl:template>

<xsl:template match="@*|node()" priority="5" mode="c">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="AA/*[1]" priority="5">
  <xsl:copy>
    <xsl:apply-templates select="//BB/*[position()=last()]" mode="c"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="BB/*[position()=last()]" priority="5">
  <xsl:copy>
    <xsl:apply-templates select="//AA/*[1]" mode="c"/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>

```

"copy" mode
(does not
select
//AA/*[1]
again..!)

69

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 of the variable-binding element 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.

70

Thus

```

<xsl:variable name="x"/> is equivalent to (empty string)
<xsl:variable name="x" select="''"/>

```

NOTE:When a variable is used to select nodes by position, be careful not to do:

```

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

```

In filter → "true"

This will output the value of the first item element, because the variable n will be bound to a result tree fragment, not a number. Instead, do either

```

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

```

NOTE:One convenient way to specify the empty node-set as the default value of a parameter is:

```

<xsl:param name="x" select="."/>

```

71

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

```

<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>

```

72

- *Used to generate auto-numbering*
- ```
<xsl:number
 level="single|multiple|any"
 count="pattern" -- which nodes count?
 from="pattern" -- starting point
 value="number-expr" -- force value
 format="s" -- (not covering)
 lang="lg" -- lang to use
 letter-value="alphabetic|traditional"
 grouping-separator="char" -- 1_000
 grouping-size="number" -- 3 in EN
>
```

# Numbering example

```
<xsl:template select="list">
 <xsl:element name="toplist">
 <xsl:attribute name="marker">
 <xsl:number level="single"/>
 <!--count defaults to siblings-->
 </xsl:attribute>
 </xsl:element>
</xsl:template>
```

'multiple' -- gathers up sibling numbers of ancestors

```
<xsl:number level="multiple"
 format="1.1.1" count="chap|sec|ssec"/>
```

|                                                                                                                                                                                                                                                                                                                                                                         |                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> &lt;?xml version="1.0"?&gt; &lt;xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"&gt;  &lt;xsl:template match="@*[node()]"&gt;   &lt;xsl:copy&gt;     &lt;xsl:number level="multiple" count="*" /&gt;     &lt;xsl:apply-templates select="@*[node()]" /&gt;   &lt;/xsl:copy&gt; &lt;/xsl:template&gt;  &lt;/xsl:stylesheet&gt; </pre> |  | <pre> &lt;?xml version="1.0"?&gt; &lt;A&gt;1 &lt;A&gt;1.1 &lt;A&gt;1.1.1&lt;a1&gt; &lt;a2&gt;1.1.1.1&lt;a2&gt; &lt;a3&gt;1.1.1.2&lt;a3&gt; &lt;a3&gt;1.1.1.3&lt;a3&gt;1.1.1.3.1&lt;u&gt;&lt;a3&gt; &lt;/A&gt; &lt;/A&gt; &lt;b&gt;1.2 &lt;BB&gt;1.2.1 &lt;b1&gt;1.2.1.1&lt;b1&gt; &lt;b2&gt;1.2.1.2&lt;b2&gt; &lt;b3&gt;1.2.1.3&lt;b3&gt; &lt;BB&gt; &lt;/b&gt; &lt;/A&gt; </pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

75

|                                                                                                                                                                                                                                                                                                                                                                               |                                                                                       |                                                                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;?xml version="1.0"?&gt; &lt;xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"&gt;  &lt;xsl:template match="@*[node()]"&gt;   &lt;xsl:copy&gt;     &lt;xsl:number level="multiple" count="a1 AA a3 u"/&gt;     &lt;xsl:apply-templates select="@*[node()]" /&gt;   &lt;/xsl:copy&gt; &lt;/xsl:template&gt;  &lt;/xsl:stylesheet&gt;</pre> |  | <pre>&lt;?&gt; &lt;?&gt; &lt;AA&gt;1 &lt;a1&gt; &lt;a2&gt; &lt;a3&gt;&lt;u&gt;&lt;a3&gt; &lt;AA&gt; &lt;/&gt; &lt;b&gt; &lt;BB&gt; &lt;b1&gt;&lt;/b1&gt; &lt;b2&gt;&lt;/b2&gt; &lt;b3&gt;&lt;/b3&gt; &lt;BB&gt; &lt;/b&gt; &lt;/&gt;</pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

76

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

<xsl:template match="@*|node()">
 <xsl:copy>
 <xsl:number level="multiple" count="a1|AA|a3|u"/>
 <xsl:apply-templates select="@*|node()"/>
 </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

→

```
<c>1
<c>1.1
<AA>1.1.1
<a1>1.1.1.1<a1>
<a2>1.1.1.1<a2>
<a3>1.1.1.2<u>1.1.1.2.1</u><a3>
<AA>
<c>
1
<BB>1
<b1>1</b1>
<b2>1</b2>
<b3>1</b3>
<BB>

</c>
```

**Annotations:**

- Insert:** Points to the table structure in the HTML output.
- What will be the result??:** Points to the final HTML output.

END  
Lecture 11