“Markup Sprachen und semi-strukturierte Daten”

http://www.pms.informatik.uni-muenchen.de/lehre/markupsemistrukt/02ss

XSLT 1.0 Tutorial

Dan Olteanu

Dan.Olteanu@pms.informatik.uni-muenchen.de
What means XSLT?

XSL (eXtensible Stylesheet Language) consists of

- **XSL-T** (Transformation)
  - primarily designed for transforming the structure of an XML document
  - W3C Specification: [http://www.w3c.org/TR/xslt](http://www.w3c.org/TR/xslt)

- **XSL-FO** (Formatting Objects)
  - designed for formatting XML documents
  - W3C Specification: [http://www.w3c.org/TR/xsl](http://www.w3c.org/TR/xsl)

Why Transform XML?

XML is a success because it is designed:

- for separation between content and presentation
  (XML is a generic markup language)
- as a format for electronic data interchange (EDI) between computer programs
- as human readable/writable format

Transforming XML is not only desirable, but necessary.

XSLT is an attempt to fulfill this need, by supporting

- publishing data (not necessarily XML).
- conversion between two proprietary formats (not necessarily XML).
Publishing XML data
Data Conversion

Company A
Groß-Apotheke, Börseninformationssystem

WSDL
Service
Format D

SOAP
Format C

Company B
Krankenhaus, Bank

WSDL
Service
Format B

Company C
Krankenkasse, Börsenticker

Legacy System
Format A
SOAP

UDDI

Legacy System
How XML data can be transformed using XSLT? (1/3)

1 a conversion of XML data into a tree structure, e.g. using an XML parser conformant to
   - Document Object Model (DOM) [http://www.w3.org/DOM/](http://www.w3.org/DOM/)
   - Simple Api for XML (SAX) [http://www.megginson.com/SAX/sax.html](http://www.megginson.com/SAX/sax.html)
How XML data can be transformed using XSLT? (2/3)

2. A **structural transformation** of the data: from the input to the desired output structure
   - involves selecting-projecting-joining, aggregating, grouping, sorting data.
   - XSLT vs. custom applications: factoring out common subtasks and present them as transformation rules in a high-level declarative language.

```
Input tree structure
para
  em
    example

Transformation rules

Output tree structure
p
  i
    example

This is an example
```

Transformation rules
3 **formatting** of the data: data in the desired output structure is enriched with target-format constructs, e.g. from PDF (paper-print), VoiceXML (aural presentations), SVG (graphics), HTML (browsing)
How XML data can be transformed using XSLT?
The place of XSLT in the XML family (1/2)

- based on XML InfoSet and Namespaces Specs.

- Styling: XSLT vs. CSS
  CSS can not
  - reorder elements from the XML document.
  - add new elements.
  - decide which elements should be displayed/omitted.
  - provide functions for handling numbers/strings/booleans.

- Processing: XSLT vs. XML Query
  - the same pattern language, i.e. XPath, and the same expressive power.
  - different processing models.

- Linking: XSLT vs. XPointer
  they share XPath as language for localizing fragments of XML documents.
The place of XSLT in the XML family (2/2)
Simple Transformation Examples with XSLT

- XSLTrace from IBM AlphaWorks

- allows a user to visually "step through" an XSL transformation, highlighting the transformation rules as they are fired.

- Add the XSLTrace.jar, xml4j.jar, lotusxsl.jar Java archives $CLASSPATH.

- command line: java com.ibm.xsl.xsltrace.XSLTrace <input> <style>

The XSLT Processing Model

- usually input, output and XSLT program - well-balanced XML documents, represented internally as XPath data model/DOM-like trees.
- different output formats: xml, html, text.
- multiple inputs via \texttt{document()} XSLT function.
- multiple outputs via \texttt{<xsl:document>} XSLT element.
- multiple programs via \texttt{<xsl:include>} and \texttt{<xsl:import>} XSLT elements.
The Supported Information Items
The Transformation Process

- based on template rules.

- a template rule = template pattern + template body.
  
  `<xsl:template match='pattern'> body </xsl:template>`

  the pattern matches nodes in the source tree.

  for the matched nodes, the template body is instantiated.

- template pattern = XPath expression.

- template body = literal result elements + XSLT instructions.

- find templates that apply to nodes in the source tree.

- more templates for the same nodes → processing modes or conflict resolution policy.

- no template for nodes → built-in templates.

- after processing a node, start to process its children:
  
  `<xsl:apply-templates>`
Push Processing

How is working?

- a template rule for each kind of node.
- apply templates for children.
- use built-in templates if needed.

Application: similar structure for input and output.

Example

- Chapter 2 from *XSLT Programmer’s Reference*, M. Kay. [http://www.wrox.com](http://www.wrox.com)
- XML Source: books.xml
- XSLT StyleSheet: books.xsl
Pull Processing

How is working?

- explicitly select and process the required nodes.
  
  `<xsl:value-of select='''pattern'''/>
  `<xsl:apply-templates select='''pattern'''/>`
  `<xsl:for-each select='''pattern'''/>`

- greater control over which nodes are to be processed.

Application: very different structure for input and output.

Example (Chapter 1)

- XML Source: books.xml
- XSLT StyleSheet: books_pull.xsl
Processing Modes

- for processing the same node in the source tree more than once, but in different ways.
- another (not general) possibility: push and pull processing for the same node.
- example: handling the section headings of a book in two different ways
  - for the table of contents (mode toc).
    ```xml
    <xsl:apply-templates select='''heading''' mode='''toc'''/>
    <xsl:template match='''heading''' mode='''toc'''/>
    ```
  - inside the body of the document (mode body).
    ```xml
    <xsl:apply-templates select='''heading''' mode='''body'''/>
    <xsl:template match='''heading''' mode='''body'''/>
    ```

Example

- Formatting the XML Specification
- Chapter 10 from  *XSLT Programmer’s Reference*, M. Kay. [http://www.wrox.com](http://www.wrox.com)
- XML Source: REC-xml-19980210.xml XSLT StyleSheets: xmlspec.xsl, xpath.xsl, xslt.xsl
Conflict Resolution Policy

- more templates with patterns matching the same node in the source tree.
- no processing modes are used.
- appears when several stylesheets are imported, or included.

Solution: each template has a priority

- set by an XSLT instruction.
  \[
  \langle \text{xsl:template match='pattern' priority='1'} \rangle
  \]
- given by the selectivity of its pattern.

<table>
<thead>
<tr>
<th>Patterns</th>
<th>Default priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>node(), text(), *</td>
<td>-0.5</td>
</tr>
<tr>
<td>abc:*</td>
<td>(-0.5, 0.0)</td>
</tr>
<tr>
<td>title, @id</td>
<td>0.0</td>
</tr>
<tr>
<td>book[@isbn], para[1]</td>
<td>&gt; 0.0</td>
</tr>
</tbody>
</table>

A numerically higher value indicates a higher priority.
**Built-in Templates**

- `<xsl:apply-templates>` is invoked to process a node, and there is no template rule in the stylesheet that matches that node.

- Built-in template rule for each type of node.

<table>
<thead>
<tr>
<th>Node type</th>
<th>Built-in template rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>call <code>&lt;xsl:apply-templates&gt;</code> to process its children.</td>
</tr>
<tr>
<td>element</td>
<td>call <code>&lt;xsl:apply-templates&gt;</code> to process its children.</td>
</tr>
<tr>
<td>attribute</td>
<td>copy the attribute value to the result tree.</td>
</tr>
<tr>
<td>text</td>
<td>copy the text to the result tree.</td>
</tr>
<tr>
<td>comment</td>
<td>do nothing.</td>
</tr>
<tr>
<td>pi</td>
<td>do nothing.</td>
</tr>
<tr>
<td>namespace</td>
<td>do nothing.</td>
</tr>
</tbody>
</table>
The XSLT Language

- XML syntax.
  Benefits reuse of XML tools for processing XSLT programs (or stylesheets).
  In practice Visual development tools needed to avoid typing angle brackets.

- free of side-effects, i.e. obtain the same result regardless of the order/number of execution of the statements.
  Benefits Useful for progressive rendering of large XML documents.
  In practice a value of a variable can not be updated.

- processing described as a set of independent pattern matching rules.
  Benefits XSLT - a declarative language.
    similar to CSS, but much more powerful.
  In practice a rule specifies what output should be produced when particular patterns occur in the input.

- dynamically-typed language.
  types are associated with values rather than with variables, like JavaScript.
Data Types in XSLT

- five data types available: boolean, number, string, node-set, external object.
- addition with XSLT 1.1: result tree fragment (RTF).
- implicit conversion is generally carried out when the context requires it.
- explicit conversion with functions boolean, number, string.

<table>
<thead>
<tr>
<th>From/To</th>
<th>boolean</th>
<th>number</th>
<th>string</th>
<th>node-set</th>
<th>external object</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>n.app.</td>
<td>false → 0</td>
<td>false → 'false'</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true → 1</td>
<td>true → 'true'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>0 → false</td>
<td>n.app.</td>
<td>decimal</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>other → true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>null → false</td>
<td>decimal</td>
<td>n.app.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>other → true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node-set</td>
<td>empty → false</td>
<td>string()</td>
<td>string value</td>
<td>n.app.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>other → true</td>
<td>function</td>
<td>of first node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>external</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.app.</td>
</tr>
<tr>
<td>object</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XSLT variables & parameters

Variables

- global variables - accessible throughout the whole stylesheet.
- local variables - available only within a particular template body.
- variable name and value defined with XSLT element `<xsl:variable>`, e.g.
  `<xsl:variable name='sum' value='0'/>
- can be referenced in XPath expressions as $sum.

Parameters

- global parameters - set from outside the stylesheet, e.g. command line, API.
  defined with XSLT element `<xsl:param>`.
- local parameters - available only within a template.
  defined with XSLT element `<xsl:with-param>`.
XPath Expressions

- evaluated in a context, consisting of a static and dynamic context.

- static context - depends on where the expression appears.
  - set of namespace declarations in force at the point where the expression is written.
  - set of variable declarations in scope at the point where the expression is written.
  - set of functions available to be called.
  - base URI of the stylesheet element containing the expression.
    for document() function.

- dynamic context - depends on the processing state at the time of expression evaluation.
  - current values of the variables in scope.
  - current location in the source tree, i.e.
  - current node - the node currently being processed.
  - context node - different from previous only for qualifiers inside expressions.
  - context position - position in the current node list.
  - context size - size of the current node list.
Stylesheet Structure

  the outermost elements of any stylesheet.

  used within an XML source to identify the stylesheet that should be used to process it.

- stylesheet modules, using
  - `<xsl:include>` - textual inclusion of the referenced stylesheet module.
    Example (Chapter 03): `sample.xml`, `principal.xsl`, `date.xsl`, `copyright.xsl`
  - `<xsl:import>` - the definitions in the imported module have lower import precedence.

- embedded stylesheets - included within another XML document,
  typically the document whose style it is defining.
XSLT Elements

- define template rules and control the way they are invoked:
  - `<xsl:template>`, `<xsl:apply-templates>`, `<xsl:call-template>`
- define the structure of a stylesheet:
  - `<xsl:stylesheet>`, `<xsl:include>`, `<xsl:import>`
- generate output:
- define variables and parameters:
  - `<xsl:variable>`, `<xsl:param>`, `<xsl:with-param>`
- copy information from the source to the result:
  - `<xsl:copy>`, `<xsl:copy-of>`
- conditional processing and iteration:
  - `<xsl:if>`, `<xsl:choose>`, `<xsl:when>`, `<xsl:otherwise>`, `<xsl:for-each>`
- sort and number:
  - `<xsl:sort>`, `<xsl:number>`
- control the final output format:
  - `<xsl:output>`, `<xsl:document>`
Finally an Example Break :-)  

- generate XSLT transformations from mappings defined using a visual interface.  
- Input examples from Chapter 4.
XSLT Design Patterns

repertoire of programming techniques in XSLT which were found useful.

- Fill-in-the blanks stylesheets.
- Navigational stylesheets.
- Rule-based stylesheets.
- Computational stylesheets.
Fill-in-the-blanks Stylesheets

- the template looks like a standard HTML file.
- addition of extra tags used to retrieve variable data.
- useful for non-programmers with HTML authoring skills.
- useful when the stylesheet has the same structure as the desired output.
- fixed content included as text or literal result elements.
- variable content included by means of `<value-of>` instructions, that extract the relevant data from the source.
- similar to a wide variety of proprietary templating languages.
- Example: `orgchart.xml`, `orgchart.xsl` (Chapter 9).
  table with one row per person, with three columns for person’s name, title, and the name of the boss.
Navigational Stylesheets

- still essentially output-oriented.
- use named templates as subroutines to perform commonly-needed tasks.
- use variables to calculate values needed in more than one place.
- looks very like a conventional procedural program with variables, conditional statements, loops, and subroutine calls.
- often used to produce reports on data-oriented XML, where the structure is regular and predictable.
- Example: booklist.xml, booksales.xsl (Chapter 9). report on the total number of sales for each publisher.
Rule-based Stylesheets

- primarily consists of template rules, describing how different informations from the source should be processed.
- represents the principal way that it is intended to be used.
- is not structured according to the desired output layout.
- like an inventory of components that might be encountered in the source, in arbitrary order.
- good for sources with flexible or unpredictable structure.
- natural evolution of CSS, with reacher pattern language and actions.
- Example: scene2.xml, scene.xsl (Chapter 9).
  HTML format for Scene 2 from Shakespeare's *Othello*. 
Computational Stylesheets

• for generating nodes in the result tree that do not correspond directly to nodes in the source, e.g.
  – there is structure in the source document that is not explicit in markup.
  – complex aggregation of data.

• based heavily on functional programming paradigm
  – no side-effects, i.e. no assignment instructions
  – recursion instead of iteration

• Example: number-list.xml, number-total.xsl (Chapter 9). totaling a list of numbers.
More XSLT Examples

- Finding the type of a node.
- Finding the namespaces of elements and attributes.
- Differentiate with XSLT.
- Computation of $n!$.
- The Sieve of Erasthothenes.
- XML to SVG.
Example: Finding the Type of a Node

<xsl:template name="node:type">
  <xsl:param name="node" select="."/>
  <xsl:choose>
    <xsl:when test="$node/self::*">
      <xsl:text> element </xsl:text>
    </xsl:when>
    <xsl:when test="$node/self::text()">
      <xsl:text> text </xsl:text>
    </xsl:when>
    <xsl:when test="$node/self::comment()">
      <xsl:text> comment </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text> processing instruction </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
Example: Finding the Namespaces of Elements and Attributes

```xml
<xsl:template match="*" mode="namespace">
  <xsl:for-each select="namespace::*">
    <xsl:variable name="uri" select="."/>
    <xsl:if test="namespace-uri(..) = $uri">
      <p> <span style="text-width:bold;color:blue;"> <xsl:value-of select="name(..)"/>
        is in namespace <code> <a href="$uri"> <xsl:value-of select="$uri"/> </a> </code>
        with prefix <code> <xsl:value-of select="name()"/> </code> </span> </p>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```
Example: Differentiate with XSLT (1/2)

\[ f(x) = (1 \cdot x^3) + (2 \cdot x^2) + (3 \cdot x^1) + (4 \cdot x^0) \]
\[ f'(x) = (3 \cdot x^2) + (4 \cdot x^1) + (3 \cdot x^0) + (0 \cdot x^{-1}) \]

DTD:

```xml
<!ELEMENT function-of-x (term+)>
<!ELEMENT term (coeff, x, power)>
  <!ELEMENT coeff (#PCDATA)>
  <!ELEMENT x EMPTY>
  <!ELEMENT power (#PCDATA)>
Instance:

<function-of-x>
  <term> <coeff> 1 </coeff> <x/> <power> 3 </power> </term>
  <term> <coeff> 2 </coeff> <x/> <power> 2 </power> </term>
  <term> <coeff> 3 </coeff> <x/> <power> 1 </power> </term>
  <term> <coeff> 4 </coeff> <x/> <power> 0 </power> </term>
</function-of-x>
```
Example: Differentiate with XSLT (2/2)

```xml
<xsl:stylesheet version='1.0' xmlns:xsl='http://.../Transform'>
  <xsl:strip-space elements='*'/>
  <xsl:output method='xml' indent='yes'/>
  <xsl:template match='function-of-x'>
    <xsl:element name='function-of-x'>
      <xsl:apply-templates select='term'/>
    </xsl:element>
  </xsl:template>

  <xsl:template match='term'>
    <term>
      <coeff> <xsl:value-of select='coeff * power'/></coeff>
      <x/>
      <power> <xsl:value-of select='power - 1'/></power>
    </term>
  </xsl:template>
</xsl:stylesheet>
```
Example: Computation of $n!$ Factorial

```xml
<xsl:template name="factorial">
  <xsl:param name="n" select="1"/>
  <xsl:variable name="sum">
    <xsl:if test="$n = 1"> 1 </xsl:if>
    <xsl:if test="$n != 1">
      <xsl:call-template name="factorial">
        <xsl:with-param name="n" select="$n - 1"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:if>
  </xsl:variable>
  <xsl:value-of select="$sum * $n"/>
</xsl:template>
```
Example: The Sieve of Erastothenes (1/2)

- Compute prime numbers
Example: The Sieve of Erastothenes (2/2)

<!-- Mark all multiples of $number in $array with '*' -->
<xsl:template name="mark">
  <xsl:param name="array"/>
  <xsl:param name="number"/>
  <xsl:choose>
    <xsl:when test="string-length($array) &gt; $number">
      <xsl:value-of select="substring($array, 1, $number - 1)"/>
      <xsl:text> * </xsl:text>
      <xsl:call-template name="mark">
        <xsl:with-param name="array" select="substring($array,$number+1)"/>
        <xsl:with-param name="number" select="$number"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$array"/>
    </xsl:otherwise>
  </xsl:choose> </xsl:template>
Example: XML to SVG

<sales> <caption> 3Q 2000 Sales Figures </caption>  
  <region> <name> Southeast </name>  
    <product name="Heron"> 38.3 </product>  
    <product name="Kingfisher"> 12.7 </product>  
  </region> </sales>

3Q 2000 Sales Figures  
(in millions of dollars)
**XSLT Processors: Saxon**

- open source, available at [http://users.iclway.co.uk/mhkay/saxon/](http://users.iclway.co.uk/mhkay/saxon/).
- runs on Java 1.1 or Java 2 platform.
- Installation
  - fetch instant-saxon.zip or saxon.zip.
  - set CLASSPATH accordingly: CLASSPATH=saxon.jar:$CLASSPATH.
- Invocation
  - command line: `saxon source.xml style.xsl > output.html`
  - Java application: via the TrAX API defined in JAXP 1.1
    java com.icl.saxon.StyleSheet source.xml style.xsl > output.html
- built-in extension XPath functions:
  `after(ns1, ns2)`, `before(ns1, ns2)`, `difference(ns1, ns2)`, `intersection(ns1, ns2)`, `distinct(ns1)`, `evaluate(string)`.
- built-in extension XSLT elements:
  `<saxon:function>`, `<saxon:return>`, `<saxon:while>`.
XSLT Processors: Xalan

- Java and C++ versions.
- Installation
  - fetch xalan.jar, xerces.jar.
  - set CLASSPATH accordingly: CLASSPATH=xerces.jar:xalan.jar:$CLASSPATH.
- Invocation
  - command line:
- user-defined and built-in extension functions and elements.
- built-in extension functions:
  difference(ns1, ns2), intersection(ns1, ns2), distinct(ns1),
  evaluate(string).
- SQL extension functions for JDBC connections.
- multiple output files.
XSLT Processors: Architecture
XSLT Processors: Comparison

XSLTMark 2.0 Results [Parse & Transform]
2001-03-14 i686-500 Linux 2.2/HT 4.0

XT 19991105
Saxon 6.2.1
Xalan 2.0.0
Sablotron 0.51
Xalan C 1.1

Aggregate KB/s

XSLTMark 2.0 Results [Transform Only]
2001-03-14 i686-500 Linux 2.2/HT 4.0

Microsoft XML Parser 3.0
Sun XSLTC alpha 4
Oracle XSLT 2.0
Gnome XSLT 0.5.0
TransformMIX 0.8
Fourthought 4Suit

Aggregate KB/s
What’s coming? XSLT 2.0

XSLT 1.1 standardizes a small number of urgent features.

- multiple output documents via `<xsl:document>`.
- temporary trees via `nodeset()`.
- standard bindings to extension functions written in Java and ECMAScript.

XSLT 2.0 at [http://www.w3.org/TR/xslt20req](http://www.w3.org/TR/xslt20req).

- simplify manipulation of XML Schema-typed content.
- support for reverse IDREF attributes, e.g. `key()` function.
- support sorting nodes based on XML Schema type.
- simplify grouping.
Tutorials: Useful links

- XSLT W3C Specification
  http://www.w3c.org/TR/xslt

  www.wrox.com

- XSLT Tutorial at Zvon

- XSL Tutorial at W3Schools
  http://www.w3schools.com/xsl/

- Practical transformation using XSLT and XPath

- The XML Cover Pages
  http://xml.coverpages.org/xsl.html