XSL Transformations

T. Pitner, L. Bártek, A. Rambousek. L Grolig
FI MU Brno 2020

XSLT - XSL Transformations

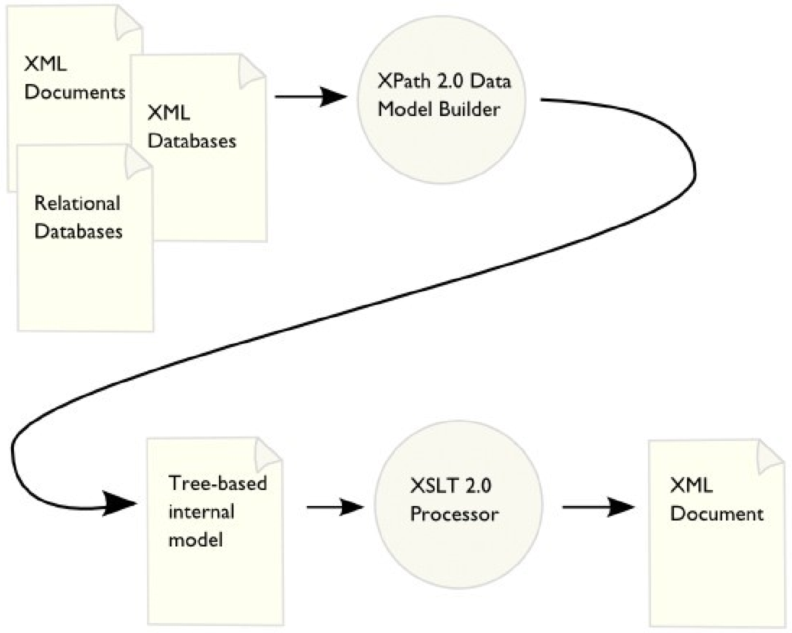
- Motivation: need for tool to **define and run transformations of XML** into XML
  - or possibly into plain-text, LaTeX source, or any other markup (YAML, JSON)
- We could do it programmatically by manipulating DOM model in any universal language but it leads to unreadable and cumbersome code.
- There is a solution: **eXtensible Stylesheet Language Transformations - XSLT**
- https://www.w3.org/TR/xslt/all/ again a **W3C Recommendation**

XSLT - XSL Transformations

- XSLT is a language for specifying transformation of XML documents on the (usually) XML outputs, or text, HTML or other output formats.
- The original application area, the transformation of XML data to XSL:FO (XSL-Formatting Objects), thus rendering XML.
- XSLT specification was therefore part of XSL (eXtensible Stylesheet Language).
- Later, XSL was set aside and XSLT began to be seen as a universal general description language for **XML → XML (txt, HTML) transformations**.
- XSLT is a *Turing-complete language*, ie. one can do “anything” in XSLT

XSLT - XSL Transformations

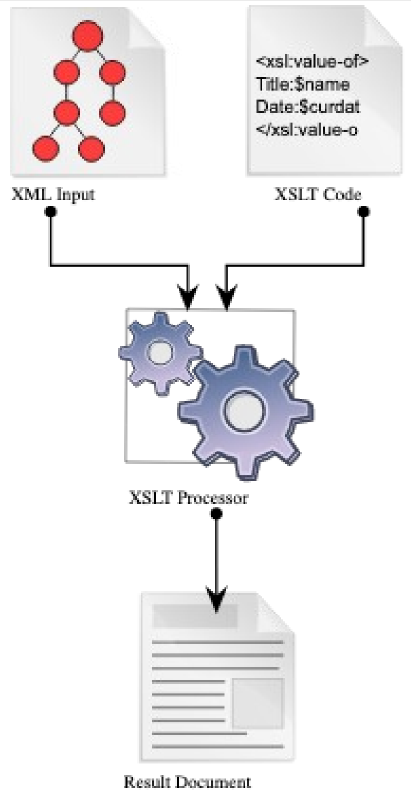
- Transformation process of XML using XSLT



XSLT - Language principles

- XSLT is a **functional language**, where reduction rules have the form of **templates**, specifying how nodes in the source document are converted into the output document.
- XSLT transformation specification is contained in a **style file** (**stylesheet**), which is an XML document written in the XSLT syntax. The root element is either `xsl:stylesheet or xsl:transformation` (which are synonyms) where xsl: is a prefix for the XSL namespace.
- The XSLT style(sheet) is then processed by an **XSLT processor** and subsequently,
- XML file(s) can be transformed using that stylesheet.

XSLT - Workflow



XSLT Example: XML source file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
     <name>Morka</name>
     <family-name>Ismincius</family-name>
  </person>
</persons>
```

XSLT Example: XSLT style

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>
  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

XSLT Example: Output

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <name username="JS1">John</name>
  <name username="MI1">Morka</name>
</root>
```

XSLT - Versions

- The current versions are defined by the XSLT 1.0, 2.0 and 3.0 specifications:
  - XSLT 1.0 XSL Transformations (XSLT) — W3C Recommendation 16 November 1999
  - XSL Transformations (XSLT) Version 2.0 — W3C Recommendation 23 January 2007
  - XSL Transformations (XSLT) Version 3.0 — W3C Last Call Working Draft 2 October 2014
  - The version 1.0 is still widely used, mainly due to lacking (free) implementations of the newer specification
    - Also, for many purposes, the features of 1.0 are sufficient.

XSLT - Processing

1) XSLT processor (interpreter) takes the stylesheet (XSLT code)
2) Usually compiles it into an internal form.
3) Then it takes the nodes from the input document, looks for an appropriate template and select it.
4) 4. Then it produces a result fragment corresponding to the construction part of the selected template.
5) Recursive takes next nodes from the input document and applies the same procedure for them.

Open-source XSLT Processors

- Xalan
  - an open source XSLT 1.0 processor from the Apache Software Foundation available stand-alone and for Java and C++. Integrated into Java SE.
- Web browsers
  - Safari, Chrome, Firefox, Opera and Internet Explorer all support XSLT 1.0. None supports XSLT 2.0 natively, although the third party products like Saxon-CE (Saxon-Client Edition) and Frameless can provide this functionality.
  - Browsers can perform on-the-fly transformations of XML files and display the transformation output in the browser window. This is done either by embedding the XSL in the XML document or by referencing a file containing XSL instructions from the XMLdocument.
  - The latter may not work with Chrome because of its security model.
- [illegible]

Open-source XSLT Processors

- libxslt
  - is a free library released under the MIT License that can be reused in commercial applications. Itis based on libxml and implemented in C. It can be used at the command line via xsltproc which is included in OS X and many Linux distributions.
- WebKit, Blink
  - used for example in the Safari and Chrome web browsers respectively, uses the libxslt library to do XSL transformations.
- Saxon
  - an XSLT (2.0 and partial 3.0) and XQuery 3.0 processor with open-source and proprietary commercial versions for stand-alone operation and for Java, JavaScript and .NET.
  - [illegible]

Commercial XSLT Processors

- MSXML and .NET
  - includes an XSLT 1.0 processor. From MSXML 4.0 it includes the command line utility msxsl.exe.
- QuiXSLT
  - an XSLT 3.0 processor doing streaming implemented in Java by Innovimax and INRIA.
- Saxon
  - commercial versions support the newest standards such as XSLT 3.0.

Information Resources

- W3C XSLT 1.0 Recommendation
  - XSLT 1.0 is still the most used version.
- What is XSLT? on XML.COM
  - http://www.xml.com/pub/a/2000/08/holman/index.html
- Mulberrytech.com XSLT Quick Reference (2xA4, PDF)
  - http://www.mulberrytech.com/quickref/XSLTquickref.pdf
- Dr. Pawson XSLT FAQ
  - http://www.dpawson.co.uk/xsl/xslfaq.html
- Zvon XSLT Tutorial
  - http://zvon.org/xxl/XSLTutorial/Books/Book1/index.html
- Safari online XSLT Reference

XSLT Syntax

Basic XSLT Elements

- Viz http://en.wikipedia.org/wiki/XSLT_elements
- xsl:stylesheet
  - (or xsl:transform) is the top-level element. Occurs only once in a stylesheet document.
  - The attribute version specifies which XSLT version is being used.
  - The NS declaration xmlns:xsl specifies the URL, which is always http://www.w3.org/1999/XSL/Transform regardless of the XSLT version.
- xsl:output
  - Child element of stylesheet.
  - It describes how data will be returned.
  - The attribute method designates what kind of data is returned (such as xml, text, html).
  - The attribute omit-xml-declaration indicates if the initial <?xml heading should be included.
  - The attribute encoding designates the encoding used for output.

Basic XSLT Elements

- xsl:template
  - Specifies processing templates “match” is when the template should be used.
  - “name” gives the template a name which xsl:call-template can use to call this template.

Declarations in xsl:stylesheet

- xsl:param
  - parameter declarations (and their implicit value). Such parameters can then be set when calling XSLT processing, e.g:.

```
java net.sf.saxon.Transform -o outfile.xml infile.xml style.xsl -Dparam=paramvalue
```

- xsl:variable
  - similarly to parameters, it declares and initializes variables.
  - They however cannot be set from outside.
  - It should also be noted that XSLT (without processor-specific extension) is a pure functional language
    - applications of templates do not have side effects → variables (or parameters) can be assigned just once, then just read!

XSLT Templates

- **Template (**xsl:template) is a specification which node should be rewriten and how
  - Which node - attribute match
  - Result of transformation - body of the element
  - After processing of the source node, the processing continues at the nodes selected by xsl:apply-templates select="*xpath expression*".
- The template can also be explicitly named (named template) using the name attribute, in which case it can be called directly / explicitly using xsl:call-template.

Modularization

- **xsl:import**
  - Retrieves another XSLT file addressed by the href (URI of the file).
  - The templates in the linking (originating) stylesheet have priority over the imported ones.
- **xsl:include**
  - Similarly, but works as a textual (verbatim) include, so no prioritization of the linking stylesheet is done.

Processing modes

- A template can specify a (processing) mode in which it can be activated.
  - The mode is indicated using the mode attribute at the xsl:template element.
- Processing starts in no mode and can be switched into another mode by using the attribute mode in the xsl:apply-templates or xsl:call-templates.

Where to use processing modes?

- Motivation
    - Modes allow a parallel set of templates with the same patterns match, but used for different purposes, for example:
        - one set of templates for generating table of contents (index) from the document,
        - one for the full text of the document itself.

Transformation Process in Detail

1. First, the processor selects the root (document node) as the current node, i.e. the node corresponding to the XPath expression /
2. Then, it finds the template matching it. If it is found, the template is used for processing this node.
3. Otherwise, a default (implicit) template is used to process the document node.
4. The processing recursively continues at nodes selected by the template that has been used for processing in the previous step.

Template Priority

- If there are two or more template matching, then ambiguity occurs and an error is emmited.
- This situation can be avoided by distinguishing the templates by setting their priorities using their priority attribute.
    - The priority can be an integer, greater number means higher priority.
- Implicit templates have lower priority than explicit ones.

Template Invocation

- **Direct**
  - xsl:call-template (possibly using xsl:with-params)
- **With implicit node selection**
  - by applying a matching template (again may be with parameters) using xsl:apply-templates without explicit selection of nodes for further processing. Then, all child elements of the current (context) node will be selected and processed.
  - Equivalent to xsl:apply-templates select="*".
- **With explicit node selection**
  - when using xsl:apply-templates with explicit selection of nodes for further processing by using the select attribute.
    - In general, the preferred way is to avoid direct invocation because the other ways correspond better to the functional nature of the XSLT language.

Outputting text nodes

- Type in text directly (as a literal) to output (body part) of the template.
  - Be careful with the whitespace characters (spaces, CR/LF) as everything gets into the output.
- When the whitespace handling is important, eg. no unnecessary whitespaces should be produced, use the special element xsl:text. Inside of it, whitespaces are always maintained!

Implicit/Default templates

- Purpose:
  - provide at least some standard "fallback" way to process basic structures such as traverse the document tree structure
  - to "save typing" frequently used templates (ignoring comments and PI).
- These default templates have low priority and can be overriden by specifying explicit templates the same (or overlapping) match clause.
- The following default templates are implicitly "embedded" in each correct XSLT processor.

Default tree (do-nothing) traversal

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

- Selects any element and the root.
- Produces nothing for it but traverses its all child elements.

Default tree (do-nothing) traversal for specified mode

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

- Does the same but only for the specified mode.

Copy text nodes and attributes

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

- Copies text nodes and attributes to the result

Ignore PIs and comments

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

- Ignores (does not include the results of the PI and comments)

Generating values programmatically

- Not only elements, attributes and texts from the source are copied to the output.
- All can be programmatically dynamically generated.

Generating Element with Calculated Attribute Value

- Task:
    - Generate the output of a predetermined element (with pre-known name), but with attributes with values with calculated during transformation.
- Solution:
    - Use the normal procedure - literal result element - attributes and values specified as the attribute value templates (AVT)

Generating Element with Calculated Attribute Value (Example)

- Input:

```
<link ref="a_link_href">
...
</link>
```

- Template:

```
<xsl:template match="link">
     <a href="#{@ref}"> ... </a>
</xsl: template>
```

Alternative:

```
<xsl:template match=”link”>
 <xsl:element name=”a”>
  <xsl:attribute name=”href”>#<xsl:value-of select=”./@ref”/></xsl:attribute>
   …
 </xsl:element>
</xsl:template>
```

Generating Element/Attribute with Calculated Name

- Objective:
  - Generate the output element whose name, attributes and content is NOT known in advance when writing the style. So it must be determined (calculated) in runtime (when transforming).
- Solution:
  - Use a template to component xsl:element
- Input:

```
<generate element-name="elt_name"> ... </generate>
```

- Template:

```
<xsl:template match="generate">
        <xsl:element name="{@element-name}">
                <xsl:attribute
        name="id">ID1</xsl:attribute>
        </xsl:element>
</xsl:template>
```

Generating Element/Attribute with Calculated Name

- Result:
  - Creates an element with the name elt_name, equipping it with the attribute id="ID1". Also the attribute name could be generated if we wished so.

XSLT Conditional processing

- Objective
    - To influence the output based on a condition.
- Solution
    - Use branching in the template - either
        - xsl:if for single then/else branches or
        - multiway xsl:choose / xsl:when / xsl:otherwise

Example xsl:if

- Input:

```
<bread price="50"> ... </bread>
```

- Template:

```
<xsl:template match="bread">
 <p>
  <xsl:if test="@price > 30">
    <span class="expensive">Expensive </span>
  </xsl:if>
  bread - price <xsl:value-of select="@price"/> CZK
 </p>
</xsl:template>
```

- Result:

Creates an element p with a record about the bread. If the bread was expensive, also

the"Expensive" indication is produced

Example xsl:choose

- Input:
```
<bread price="12"> ... </bread>
<bread price="19"> ... </bread>
<bread price="30"> ... </bread>
```
- Template:
```
<xsl:template match="bread">
  <xsl:choose>
    <xsl:when test="@price > 30">
     <span class="expensive">Expensive</span>
    </xsl:when>
    <xsl:when test="@price < 10">
     <span class="strangely-cheap">Suspiciously cheap</span>
    </xsl:when>
    <xsl:otherwise>
     <span class="normal-price">Normal</span>
    </xsl:otherwise>
  </xsl:choose> bread - price <xsl:value-of select="@price"/> CZK
</xsl:template>
```

Result: Filters out the two extreme price level — normal prices remain for xsl:otherwise.

Loops

- Input:

```
<grocery>
 <bread price="12"> ... </bread>
 <bread price="19"> ... </bread>
 <bread price="30"> ... </bread>
</grocery>
```

- Template:

```
<xsl:template match="grocery">
  <xsl:for-each select="bread">
   <p> bread - price <xsl:value-of select="@price"/> CZK </p>
  </xsl:for-each>
</xsl:template>
```

- Result: Creates series of elements p with bread prices.
- Caution - Construction xsl:for-each typically has procedural nature, which is generally not recommended for XSLT as it namely gives minimum flexibility to iterate through the contents of a set of nodes — we must know its exact structures beforehand. The style is also more difficult to modify if the structure changes (eg. new or altered element names).

Template calls and parameters

- Named template declaration
  <xsl:template name="_thistemplatename_">. The template may contain declarations of parameters: <xsl:param name="_parametername_"/> (parameter type is not specified — i.e. dynamic typing)
- Template call using <xsl:call-template name="_atemplatename_"/>
- The call can also specify the parameters (if they were declared at the template definition):
  <xsl:with-param name="_parametername_" select="_parametervalueexpression_"/>
  or
  <xsl:with-param name="_parametername_">_parametervalue_</xsl:with-param>
- Default parameter value can also be specified using
  <xsl:param name="_parametername_" select="_defaultvalueexpression_"/>

Automatic (generated) numbering

- Achieved by using xsl:number element
- For either (or both): counting elements in input to allow automatic numbering — for example to number book chapters sequentially, or formatting numbers, eg. writing them in Arabic or Roman numbers. Resembles part of the internationalization support seen in java.text.
- The autonumbering can also be multilevel eg. (sub)chapter numbers like 1.1 etc.

Example

- Template:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>
  <xsl:template match="group">
    <xsl:text>Group </xsl:text>
    <xsl:number count="group" level="multiple"/>
    <xsl:text>&#xa;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="person">
    <xsl:number count="group" level="multiple" format="1.1.1."/>
    <xsl:number count="person" level="single" format="1 "/>
    <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Example

- Source data

```
<people>
 <group>
  <person name="Al Zehtooney" age="33" sex="m" smoker="no"/>
  <person name="Brad York" age="38" sex="m" smoker="yes"/>
 </group>
 <group>
  <person name="Greg Sutter" age="40" sex="m" smoker="no"/>
  <person name="Harry Rogers" age="37" sex="m" smoker="no"/>
  <group>
   <person name="John Quincy" age="43" sex="m" smoker="yes"/>
   <person name="Kent Peterson" age="31" sex="m" smoker="no"/>
  </group>
  <person name="John Frank" age="24" sex="m" smoker="no"/>
 </group>
</people>
```

Example

- Result:
  Group 1
  1.1 Al Zehtooney
  1.2 Brad York
  Group 2
  2.1 Greg Sutter
  2.2 Harry Rogers
  Group 2.1
  2.1.1 John Quincy
  2.1.2 Kent Peterson
  2.3 John Frank

Namespace Handling

- XSLT allows to select and produce nodes (elements, attributes) in namespaces.
- However, it has some pitfalls, see Namespaces in XSLT issues
- Multiple namespaces in XSLT
- Avoid Namespace mistakes in XSLT at Developerworks

Where to do XSLT?

- Online (just for fun)
- In all XML professional editors and many programmers' IDE such as NetBeans
- Command-line tools, such as xsltproc or xmlstarlet
- From within Java programs using Java Core API (javax.xml.transform package)
- Using specialized tools programmatically (via API) or command-line, such as Saxon
- Similarly for other languages, almost all now have XML/XSLT support

Online Tools

- Good for simple try-and-see:
  - Online XSLT engine @Freeformatter.com plus a demo XML input/XSLT/XML output
  - W3Schools online XSLT engine maybe even better :-)
- With XSLT 2.0 support:
  - http://xslttest.appspot.com/

Using XSLT in Java (Core API)

- See Using the XSLT Processor for Java from Oracle.