PB138 – Markup Languages

Tomáš Pitner

March 17, 2013

Obsah

Context, history

- XSLT (eXtensible Stylesheet Language Transformation) (`http://w3.org/style/XSL`) 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 (Formatting objects), thus rendering XML.
- XSLT specification was therefore part of XSL (eXtensible Stylesheet Language).
- Later, XSL set aside and began to be seen as a universal general description language XML$\rightarrow$ XML (txt, HTML) transformations.
- The current version is determined by the XSLT 1.0 specification. Work on version 1.1 have been pledged in favor of the development of XSLT 2.0.

The main principles

- XSLT is a functional language, where reduction rules have the form templates, which specify how nodes in the source document override output document.
- XSLT transformation specification is contained in the `stylesheet` element, which is an XML document in the syntax XSLT.
- XSLT stylesheet contains usually a set of templates in `template` elements.

Main principles (2)

- The templates have a selection part (LHS of the reduction rule) and *construction part representing the RHS of the rule*
- Selection part: the attribute `match`
- Construction part: the body of the `template` element
- The own transformation then means that XSLT interpreter (XSLT processor, an XSLT engine) takes the input nodes of the document, it looks to their appropriate templates - according to the match clause and produces a result corresponding to construction content of this template.

The main sources of information - specifications, references, tutorials, FAQ

- XSLT 1.0 W3C Recommendation: `http://www.w3.org/TR/xslt`
- *What is XSLT?* na 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`

The structure of the XSLT style

The root element `xsl: transform` or `xsl: stylesheet` encloses the whole XSLT style and NS specifies the prefix for the XSLT elements.
The root element is:

- Parameter declarations (and their implicit value) - elt. `xsl:param`.
- Parameters can be set when calling XSLT processor - eg `java-o net.sf.saxon.Transform outfile.xml infile.xml style.xsl-Dparam = paramvalueVariables`
- Variables declarations - elt. `xsl:variable` - de facto same as parameters but not settable from outside.
- It should be noted that the XSLT (without processor-specific extension) is a pure functional language, i.e. a template application does not have side effect → variables can be assigned once, then just read!

Overall structure of an XSLT stylesheet

In the root element:

- Declaration (format) of output - elt. `xsl:output`
- ...apart of this, also less frequently used elements appear here - see eg. documentation for SAXONu (`http://saxon.sf.net`)
- own templates - elt. `xsl:template`

XSLT templates

- Template is a specification which node to rewrite (transform) and how.
- Which nodes to rewrite is defined in the *attribute* match.
- The result is given in the template body.
- The template can be explicitly named, in such case it can be directly called using xsl:call-template.

XSLT - input document processing

- First, the processor selects the document root (not the root element) - corresponding to the XPath expression /
- Then the processor finds a matching template (*explicit* or *implicit* - see eg. XSLT/XPath Quick Reference (`http://www.mulberrytech.com/quickref/XSLTquickref.pdf`)), where the match attribute as an XPath predicate returns true in the context of the current node ("matches" the current node).
- if there are more matching templates and they cannot be distinguished/ordered by priority - an error is indicated.
- if there is just one such template, it is applied, ie. *its body is translated into the result tree fragment.*

XSLT - template activation order

Can be specified:

Directly/explicitly calling a named template - this is an imperative approach which should be avoided.

Indirectly/implicitly by activating a template by selecting elements (or other nodes) and letting the processor to find a matching template itself - **functional approach** - preferable.
The selection of nodes is done by:

- Explicitly by "select" at "apply-templates". We can select any nodes specified by the XPath expression in "select".
- Implicitly, letting the processor to select nodes (no "select" at "apply-templates"). *Only child elements are selected then.*

XSLT - specification of template output

- The output of a template is a *result tree fragmentu*.
- The outputs of individual templates are *placed to the result tree fragment*, in the order corresponding to the application order.
- The output is usually generated as a stream of events (eg. SAX2), which are subsequently converted to the resulting document (using specified encoding etc.).

XSLT - outputting text nodes

How to produce a text node:

1. Insert the text into the template body. Note the whitespaces! (space, tab, CR/LF)!
2. Use special elt. ¡xsl:text¿text node¡/xsl:text¿. Whitespaces are preserved.

Implicit templates

Implicit templates are defined by the specification and are implemented by any conformant XSLT processor in order to:

- enable traversing the document tree even in case the traversal is not explicitly defined
- to define default typical actions: like ignoring comments and PIs
- can be overriden by explicit template(s) with the same match= attribute

Implicit templates overview (1)

- "Default tree (do-nothing) traversal":

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

- "Default tree (do-nothing) traversal for specified mode":

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

Implicit templates overview (2)

- "Copy text nodes and attributes" into the result tree fragment:

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

- "Ignore PIs and comments":

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

Generate element with given attributes

Goal: Generate given element (with a priori known name) but calculated attribute values.
Solution: Use literal result element as usually - and specify att. values in so-called *attribute value templates (AVT)*:

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

Template:

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

The link will be transformed to element "a", the attribute value "href" will be calculated so that it inserts "#" before the original value of "ref"

Element with both attributes and name generated

Task: Generate element with run-time generated name, attribute names, value...
*How-to:*Use `xsl:element`:

```
<generate element="elt\_name"> ... </generate>
```

Šablona:

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

Creates element `elt_name` with attribute `id="ID1"` .

Flow-control inside template - conditional parts

Simple conditional output by xsl:if

```
<rohlik cena="5"> ... </rohlik>
```

Template adding content if price ¿ 2:

```
<xsl:template match="rohlik">
   <p>
      <xsl:if test="cena>2">
         <span class="expensive">Drahý</span>
      </xsl:if> rohlík - cena <xsl:value-of select="@cena"/
</xsl:template>
```

Flow-control inside template - branching

```
<rohlik cena="5"> ... </rohlik>
<rohlik cena="2"> ... </rohlik>
<rohlik cena="0.9"> ... </rohlik>

<xsl:template match="rohlik">
   <p>
      <xsl:when test="cena>2">
         <span class="expensive">Drahý</span>
      </xsl:when>
      <xsl:when test="cena<1">
         <span class="strangely-cheap">Podezřele levný</spa
      </xsl:when>
      <xsl:otherwise>
         <span class="normal-price">Běžný</span>
      </xsl:otherwise> rohlík - cena <xsl:value-of select='
</xsl:template>
```

Flow-control inside template - loops

```
<pecivo>
   <rohlik cena="5"> ... </rohlik>
   <rohlik cena="2"> ... </rohlik>
   <rohlik cena="0.9"> ... </rohlik>
</pecivo>

<xsl:template match="pecivo">
   <xsl:for-each select="rohlik">
      <p>Rohlík - cena <xsl:value-of select="@cena"/> Kč</
   </xsl:for-each>
</xsl:template>
```

For each "rohlik" generates a paragraph with info on "rohlik" and its price. *Note:* The xsl:for-each has a typical procedural character, it is good to use it less frequently, as it is too tightly bound to the precise input structure.

Processing modes

Modes allow to have more templates with the same match patterns for different purposes, eg.:

- one for generating ToC of a document
- second for generating full-text of the document

One can switch on the mode when using `apply-templates` with attribute `mode`:

- if mode= is present, only templates with the same mode will match
- if not present, only templates without any mode attribute will match

Declaring and calling named templates

Declaration - `xsl:template name="jmeno_sablony"`
Might contain parameters:

- `<xsl:param name="jmenoParametru"/>`

Calling - `<xsl:call-template name="jmenoSablony">`
might specify actual parameter values:

- `<xsl:with-param name="jmenoParametru" select="hodnotaParametru"/>` or
- `<xsl:with-param name="jmenoParametru">hodnota parametru</xsl:with-param>`

Automated (generated) numbering /1

If we add the element xsl:number into the template body, a number given by a counter will be produced.
The counter can be specified:

- ordinal number of the source element within its parent - also multi-level, like (sub)chapter 1.1. etc.

Automated (generated) numbering /2

If we apply the style:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Trans
                version="1.0">
   <xsl:template match="/">
      <html>
         <body>
            <xsl:for-each select="devguru_staff/programmer'
               <xsl:number value="position()" format="1. "
               <xsl:value-of select="name" />
               <br/>
            </xsl:for-each>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Automated (generated) numbering /3

to the following source file:

```
<devguru_staff>
    <programmer>
        <name>Bugs Bunny</name>
        <dob>03/21/1970</dob>
        <age>31</age>
        <address>4895 Wabbit Hole Road</address>
        <phone>865-111-1111</phone>
    </programmer>
    <programmer>
        <name>Daisy Duck</name>
        <dob>08/09/1949</dob>
        <age>51</age>
        <address>748 Golden Pond</address>
        <phone>865-222-2222</phone>
    </programmer>
```

Automated (generated) numbering /4

it gains the resulting HTML page (the indentation might differ...)

```
<html>
    <body>1. Bugs Bunny<br>
          2. Daisy Duck<br>
          3. Minnie Mouse<br>
    </body>
</html>
```

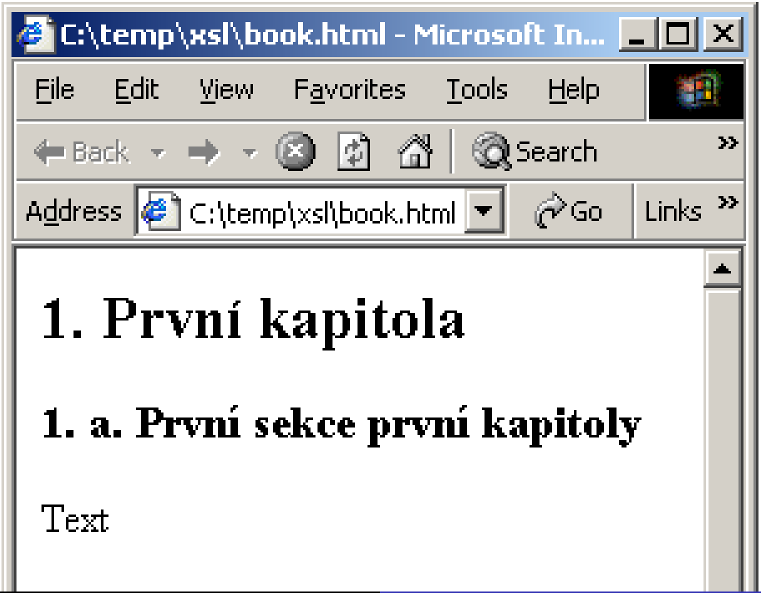
Automatic numbering (2)

```
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
    <xsl:template match="/book">
        <html>
            <body>
                <xsl:for-each select="chapter">
                    <h2>
                        <xsl:number count="chapter" format=
                        <xsl:value-of select="title" />
                    </h2>
                    <xsl:for-each select="sect1">
                        <h3>
                            <xsl:number count="chapter" for
                            <xsl:number count="sect1" forma
                            <xsl:value-of select="title" /
```

Automatic numbering (3)

```
<book>
    <title>Moje nová kniha</title>
    <chapter>
        <title>První kapitola</title>
        <sect1>
            <title>První sekce první kapitoly</title>
            <para>Text</para>
        </sect1>
        <sect1>
            <title>Druhá sekce první kapitoly</title>
            <para>Text druhé sekce</para>
        </sect1>
    </chapter>
    <chapter>
        <title>Druhá kapitola</title>
        <sect1>
```

Automatic numbering (4)


C:\temp\xsl\book.html - Microsoft In...
File Edit View Favorites Tools Help
Back Search
Address C:\temp\xsl\book.html Go Links

1. První kapitola

1. a. První sekce první kapitoly

Text

What is recommended?

- Prefer functional approach - eg. `xsl:template match=` and `xsl:apply-templates select=`
- before procedural approach - `xsl:template name=` and `xsl:call-template name=`

Use the working *modes* (xsl:template ... mode= and xsl:apply-templates ... mode=)
modes can be combined with functional approach:

- xsl:apply-templates select=... mode=...
- xsl:template match=... mode=...

Reusability of styles

What to do to achieve reusability?

- Restructure styles to more, simpler files
- Include them into others using `xsl:include` (almost like textual inclusion)
- or better by `xsl:import` (as import preferres the directly present templates before those imported)

Design patterns

Identical transformation 1 (no root element attributes) `http://wwbota.free.fr/XSLT_models/identquery.xslt`
Identical transformation 2 `http://wwbota.free.fr/XSLT_models/identquery2.xslt`
Identical transformation suppressing elements no having any text nodes in //* `http://wwbota.free.fr/XSLT_models/suppressEmptyElements.xslt`
Replace attributes with elements `http://wwbota.free.fr/XSLT_models/attributes2elements.xslt`
Dtto, but placing former attributes into elements in a specific namespace xslt/attributes2elements.xslt (`http://www.fi.muni.cz/~tomp/xml03/xslt/attributes2elements.xslt`)
Reverse transformation xslt/elements2attributes.xslt (`http://www.fi.muni.cz/~tomp/xml03/xslt/elements2attributes.xslt`)

XSLT Processors

Popular free XSLT processors (Transformation Engines) in Java:

- SAXON (author M.H.Kay) (`http://saxon.sf.net`)
- XALAN (author Apache Software Foundation) (`http://xml.apache.org/xalan-j/index.html`)
- ... more free and commercial XSLT tools: XML Software (`http://www.xmlsoftware.com/xslt.html`)

Advanced topics

XSLT Design Patterns - selection
(`http://www.dpawson.co.uk/xsl/sect1/N169.html`)
The Functional Programming Language XSLT
(`http://www.topxml.com/xsl/articles/fp/1.asp`)