NPRG036

XML Technologies

Lecture 6

XSLT

23. 3. 2020

Author: **Irena Holubová**
Lecturer: **Martin Svoboda**

http://www.ksi.mff.cuni.cz/~svoboda/courses/192-NPRG036/

Lecture Outline

- **XSLT**
  - Principles
  - Templates
  - Instructions

XSLT (XML Stylesheet Language for Transformations)

- Originally: transformation of XML documents for the purpose of their visualization
  - XSL Formatting Objects (XSL-FO)
  - Pages, regions, lines, …
- Now:
  - A language with (almost) the same expressive power as XQuery
    - XML query language
  - Output: any <u>text</u> format

XSLT Basic Principles

- Input: one or more XML documents
- Output: one or more documents
  - Not only XML
  - In the basic version one
  - Input data are not modified
- XSLT script = XML document
  - Must follow the XML rules
    - Prologue, well-formedness, validity, …
  - Can be processed using any XML technology
    - DOM, SAX, XPath, XSLT, XQuery…

XSLT Basic Principles – Input

```
<?xml version="1.0"?>
<order number="322" date="10/10/2008" status="dispatched">
 <customer number="C992">
  <name>Martin Nečaský</name>
  <email>martinnec@gmail.com</email>
 </customer>
 <items>
  <item code="48282811">
   <name>CD</name>
   <amount>5</amount><price>22</price>
  </item>
  <item code="929118813">
   <name>Dell Latitude D630</name>
   <amount>1</amount><price>30000</price><colour>blue</colour>
  </item>
 </items>
</order>
```

XSLT Basic Principles – Output

```
<?xml version="1.0"?>
<html>
 <head><title>Order no. 322 – Martin Nečaský</title></head>
 <body>
  <table>
   <tr>
    <td>CD</td>
    <td>22 CZK</td><td>5 pc</td>
   </tr>
   <tr>
    <td>Dell Latitude D630</td>
    <td>30000 CZK</td><td>1 pc</td>
   </tr>
  </table>
  <div>Total price: 30110 CZK</div>
 </body>
</html>
```

XSLT Basic Principles

- Using XSLT we create a transformation script
- The script consists of templates
- A template is applied on a selected node of the input XML document and produces the specified output
  - It can trigger application of other templates on the same node or other nodes
  - It can read the data from the input document or other documents

XSLT Script – Basics

- XSLT uses XML format
  - Prologue
  - Root element stylesheet

```
<?xml version="1.0" encoding="windows-1250"?>
<stylesheet>
 ...
</stylesheet>
```

XSLT Script – Basics

- Root element xsl:stylesheet
  - Namespace of XSLT language
  - Other namespaces (if necessary)
- Attribute version – XSLT version
  - 1.0, 2.0, 3.0

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml"
    version="1.0">

</xsl:stylesheet>
```

XSLT Script – Basics

- Element xsl:output
  - Child element of element xsl:stylesheet
  - Denotes the type of output document
    - xml, pdf, text, …
      - The XSLT parser may add, e.g., prologue
      - Implementation dependent
  - indent = "yes" denotes whether the XSLT parser indents the output
    - Adds formatting white spaces

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet ...>
  <xsl:output method="xml" indent="yes" />
</xsl:stylesheet>
```

XSLT Templates

- Element xsl:template
  - Child element of element xsl:stylesheet
  - Describes a single template
  - The script can (and usually does) contain multiple templates
    - All at the same level

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet ... >
  <xsl:template> ... </xsl:template>
  <xsl:template> ... </xsl:template>
  ...
</xsl:stylesheet>
```

XSLT Templates

- Input: XML node which can be selected using an XPath path
  - Element, attribute, text, ...
- Output:
  - XML fragment (sequence of XML nodes)
  - In general any text (HTML, PDF, CSV, …)

XSLT Templates

- Two types
- Unnamed templates
  - Element xsl:template with attribute match
  - The value of the attribute is a sequence of XPath paths delimited with '|'
  - Steps of XPath paths can use axes child attribute or abbreviation '//'

```
<xsl:template match="[xpath path ['|' xpath path]*]">
 ...
</xsl:template>
```

XSLT Templates

- <u>Named templates</u>
  - Element xsl:template with attribute name
  - The value of the attribute is the name of the template

```
<xsl:template name="[template name]">
 ...
</xsl:template>
```

How does it work?

- XSLT script:
  - is executed using a program called XSLT processor
    - saxon, xsltproc, ...
    - Often also built in browsers
  - is executed over an input XML document
    - We can have multiple input documents
    - Others are referenced from the script

How does it work?

- XSLT processor works according to the following algorithm:
  - Create a context set of nodes C and add there the root node of the input XML document
  - While C is non-empty do:
    - Take the first node u from C
      - The order is given by the order in the XML document
    - Find the most suitable template for u and process it according to the template
      - Which template is the most suitable?
      - What if there is no suitable template?
        - What is the output of an empty XSLT script?

    The processing might extend C.

How does it work?

- The algorithm for finding the most suitable template for node u:
  - We search among unnamed templates
    - i.e. those with attribute match
  - We consider only those templates, whose XPath path P in attribute match describes (covers) node u
    - i.e. u is from some part of the document accessible using P

How does it work?

- What if there are multiple suitable templates?
  - We can always apply only one
  - We take the one with the highest priority
    - It can be set explicitly using attribute priority of element xsl:template
    - If it is not set, the priority is evaluated implicitly as follows:
      - 0.5: path with more than one step
      - 0: element/attribute name
      - -0.25: *
      - -0.5: node(), text(), …

How does it work?

- What if there is no suitable template?
  - We have implicit (pre-defined, default) templates → there is <u>always</u> a template to be applied
  - They have the lowest priority
    - i.e. they are applied only if there is no other option
- Consequence: An empty XSLT output applies only implicit templates
  - i.e. an empty XSLT script <u>does</u> something
    - see later

How does it work? – Examples

```
<xsl:template match="/">
 <!-- transformation of root note -->
</xsl:template>

<xsl:template match="item">
 <!-- transformation of element item -->
</xsl:template>

<xsl:template match="name">
 <!-- transformation of element name -->
</xsl:template>

<xsl:template match="customer/name">
 <!-- transformation of element name having a parent
element customer -->
</xsl:template>
```

How does it work? – Examples

```
<xsl:template match="*|@*">
 <!-- transformation of any element or attribute -->
</xsl:template>

<xsl:template match="customer/*">
 <!-- transformation of any child element of element customer -->
</xsl:template>

<xsl:template match="text()">
 <!-- transformation of any text node -->
</xsl:template>

<xsl:template match="order//node()">
 <!-- transformation of any descendant of element order -->
</xsl:template>
```

Body of a Template – Options

1. Creating elements and/or attributes
   - Directly (writing a text) or using elements xsl:element and xsl:attribute
2. Creating text nodes
   - Directly (writing a text) or using element xsl:text
3. Access to input data
   - Using element xsl:value-of

Body of a Template – Options

4. Calling other templates
   - Using elements xsl:apply-templates and xsl:call-template
5. Variables and parameters
   - Using elements xsl:variable and xsl:param
6. Repetition
   - Using element xsl:for-each
7. Branching
   - Using elements xsl:if and xsl:choose

Creating Elements/Attributes

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <!-- creating of the title of the order -->
   </title>
  </head>
  <table border="1">
   <!-- generating of lines for items of the order -->
  </table>
  <!-- generating of the total price -->
 </html>
</xsl:template>
```

Creating Elements/Attributes

- In the body of the template we directly write the output
  - Everything that does not belong to the XSLT namespace forms the output
- Or we use element xsl:element
  - Creates an element with the given name and content
    - Denoted using attribute name and element content
- … and element xsl:attribute
  - Creates an attribute with the given name and value
    - Denoted using attribute name and element content
- Elements xsl:… enable to “calculate" element/attribute name
  - e.g. from input data

Creating Elements/Attributes

```
<xsl:template match="/">
 <html>
  <head>
   <xsl:element name="title">
    <!-- creating of the title of the order -->
   </xsl:element>
  </head>
  <table>
   <xsl:attribute name="border">1</xsl:attribute>
   <!-- generating of lines for items of the order -->
  </table>
  <!-- generating of the total price -->
 </html>
</xsl:template>
```

Creating Elements/Attributes

```
<xsl:template match="/">
 <orders>
  <xsl:for-each select="//order">
   <order>
    <xsl:if test="./@status">
     <xsl:element name="{./@status}">
      YES
     </xsl:element>
    </xsl:if>
   </order>
  </xsl:for-each>
 </orders>
</xsl:template>
```

Creating Text Nodes

- In the body of a template we can directly write text output

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    Order no. <!-- order number --> - <!-- customer name -->
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Creating Text Nodes

- Using xsl:text

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <xsl:text>Order no.</xsl:text>
    <!-- order number -->
    <xsl:text>-</xsl:text>
    <!-- customer name -->
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Input Data

- The access to the input data is enabled by element xsl:value-of
  - Attribute select specifies the value
    - Using an XPath path
    - The expression is evaluated in the context of the current node being processed by the template
  - The resulting value forms the output
  - The resulting value is text
    - String value

Input Data

```
<xsl:template match="/">
 <html>
  <head>
   <title>
    <xsl:text>Order no.</xsl:text>
    <xsl:value-of select="order/@number" />
    <xsl:text>-</xsl:text>
    <xsl:value-of select=".//customer/name" />
   </title>
  </head>
  ...
 </html>
</xsl:template>
```

Calling Other Templates

- Problem: the XSLT parser finds the most suitable template for transformation of root node (usually match="/") of the input XML document
  - What next?
  - We want to transform also other nodes in the document tree

Calling Other Templates

- Element xsl:apply-templates
  - At the place of calling it initiates transformation of other nodes
    - By default child nodes of the currently processed node
  - Using attribute select we can specify other nodes than child nodes
    - Using an XPath path
  - The selected nodes are processed in the same way as the current node
    - They are added to the context set C
    - The most suitable template is found for each node, …

Calling Other Templates

```
<xsl:template match="/">
 <html>
  <head>
   ...
  </head>
  <table>
   <xsl:apply-templates />
  </table>
  ...
 </html>
</xsl:template>
...
```

Calling Other Templates

```
...
<xsl:template match="/">
 <html>
  <head>
   ...
  </head>
  <table>
   <xsl:apply-templates select=".//item"/>
  </table>
  ...
 </html>
</xsl:template>
```

Calling Other Templates

```
<xsl:template match="item">
 <tr>
  <td>
   <xsl:value-of select="name" />
  </td>
  <td>
   <xsl:value-of select="price" />
   <xsl:text> CZK</xsl:text>
  </td>
  <td>
   <xsl:value-of select="amount" />
   <xsl:text> pc</xsl:text>
  </td>
 </tr>
</xsl:template>
```

Calling Other Templates

- Element xsl:call-template
  - Application of a particular template on a particular set of nodes
    - The template is specified using its name (attribute name)
  - XSLT parser does not look for the most suitable template, but it applies the one with the specified name
    - Similar to calling a function/procedure

Calling Other Templates

```
<xsl:template match="item">
 <tr>
  ...
  <td>
   <xsl:call-template name="value-added-tax" />
   <xsl:text> CZK</xsl:text>
  </td>
  ...
 </tr>
</xsl:template>

<xsl:template name="value-added-tax">
 <xsl:value-of select = "./price * 1.19" />
</xsl:template>
```

Variables and Parameters

- Variable enables to store a value and refer to it
  - Element xsl:variable with attribute name and (optional) attribute select
  - Local (within templates) and global (child nodes of element xsl:stylesheet)
- Parameter is a variable which is “visible outside” a template
  - When calling a template, we can specify also its parameters
  - Element xsl:param with attribute name and (optional) attribute select

Variables and Parameters

```
<xsl:variable name="number-of-items">
 <xsl:value-of select="count(//item)" />
</xsl:variable>

<xsl:template match="/">
 <tr>
   ...
  <xsl:text>Number of items: </xsl:text>
  <xsl:value-of select="$number-of-items" />
 </tr>
</xsl:template>
```

Variables and Parameters

```
<xsl:template match="item">
 <tr>
  ...
  <td>
   <xsl:call-template name="value-added-tax">
    <xsl:with-param name="price" select="./price" />
   </xsl:call-template>
   <xsl:text> CZK</xsl:text>
  </td>
  ...
 </tr>
</xsl:template>

<xsl:template name="value-added-tax">
 <xsl:param name="price" select="0" />
 <xsl:value-of select = "$price * 1.19" />
</xsl:template>
```

Variables and Parameters

- Note: The values of variables and parameters cannot be changed
  - Once we set the value, we cannot modify it
  - We are in functional programming, not imperative

Wrong Usage of Variables

```
<xsl:variable name="total-price">
 <xsl:value-of select="0" />
</xsl:variable>

<xsl:template match="/">
 ...<xsl:apply-tempates select=".//item" />...
 <xsl:text>Total price: </xsl:text>
 <xsl:value-of select="$total-price" />
</xsl:template>

<xsl:template match="item">
 <tr>
  ...
 </tr>
 <xsl:variable name="total-price"
               select="$total-price + (./price * ./amount)"/>
</xsl:template>
```

It does not work!!

Repetition

- Using xsl:for-each
  - Similar to for loops
  - Attribute select selects a set of nodes on which the body of element xsl:for-each is applied

```
<xsl:for-each select=".//item">
 <xsl:call-template name="process-item" />
</xsl:for-each>
```

Conditions

- Using element xsl:if we can execute a part of a template only in case a condition is satisfied
  - Attribute test contains a logical XPath condition
- Note: It does not have an else branch!!

```
<xsl:if test="@dispatched">
 <xsl:text>The order was dispatched on </xsl:text>
 <xsl:value-of select="@dispatched" />
</xsl:if>
```

Branching

- Generalization of xsl:if is xsl:choose
  - One of more branches xsl:when
    - With attribute test containing the condition
    - Executed, when the condition is satisfied, others are ignored
  - One branch xsl:otherwise
    - Executed, if no xsl:when branch was executed

Branching

```
<xsl:choose>
 <xsl:when test="@dispatched">
  <xsl:text>The order was dispatched.</xsl:text>
 </xsl:when>
 <xsl:when test="@delivered">
  <xsl:text>The order was delivered.</xsl:text>
 </xsl:when>
 <xsl:otherwise>
  <xsl:text>The order is being processed.</xsl:text>
 </xsl:otherwise>
</xsl:choose>
```

Example: Recursion I.

```
<xsl:template name="total-price">
 <xsl:param name="inter-result" />
 <xsl:param name="item" />
 <xsl:variable name="newinter-result"
       select="$inter-result + ($item/price * $item/amount)" />
 <xsl:choose>
  <xsl:when test="count($item/following-sibling::item)>0">
   <xsl:call-template name="total-price">
    <xsl:with-param name="inter-result" select="$newinter-result" />
    <xsl:with-param name="item"
                    select="$item/following-sibling::item[1]" />
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="value-added-tax">
    <xsl:with-param name="price" select="$newinter-result" />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Example: Recursion I.

```
<xsl:template match="/">
 ...
 <xsl:text>Total price: </xsl:text>
 <xsl:call-template name="total-price">
  <xsl:with-param name="inter-result" select="0" />
  <xsl:with-param name="item"
                  select="./order/items/item[1]" />
 </xsl:call-template>
 ...
</xsl:template>
```

Example: Recursion II.

```
<xsl:template name="total-price">
 <xsl:param name="inter-result" />
 <xsl:param name="item-position" />
 <xsl:variable name="item"
              select="/descendant::item[$item-position]" />
 <xsl:variable name="newinter-result"
      select="$inter-result + ($item/price * $item/amount)" />
 <xsl:choose>
  <xsl:when test="count($item/following-sibling::item)>0">
   <xsl:call-template name="total-price">
    <xsl:with-param name="inter-result" select="$newinter-result" />
    <xsl:with-param name="item-position" select="$item-position + 1" />
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="value-added-tax">
    <xsl:with-param name="price" select="$newinter-result" />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Example: Recursion II.

```
<xsl:template match="/">
 ...
 <xsl:text>Total price: </xsl:text>
 <xsl:call-template name="total-price">
  <xsl:with-param name="inter-result"   select="0" />
  <xsl:with-param name="item-position" select="1" />
 </xsl:call-template>
 ...
</xsl:template>
```

Example: Intermediate Results

```
<xsl:template match="/">
 ...
 <xsl:text>Total price: </xsl:text>
 <xsl:variable name="price-inter-results">
  <mz:inter-results>
   <xsl:for-each select="//item">
    <mz:inter-result>
     <xsl:value-of select="./price * ./amount" />
    </mz:inter-result>
   </xsl:for-each>
  </mz:inter-results>
 </xsl:variable>
 <xsl:call-template name="value-added-tax">
  <xsl:with-param name="price"
       select="sum($price-inter-results//mz:inter-result)" />
 </xsl:call-template>
</xsl:template>
```

- Note: Works in XSLT 2.0. XSLT 1.0 does not allow querying of a variable set using element content, not attribute select.

Implicit Templates

- An empty XSLT script applied on a non-empty input produces a non-empty output
  - Why?
    - Due to implicit templates
  - When a node should be transformed and we cannot find a suitable user-specified template, an implicit template is used

Implicit Templates

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

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

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

Implicit Templates

- How to “switch off” implicit templates?
  - We can re-define them

```
<xsl:template match="node()" />
```

- This template says that we should do nothing for any node
- All our templates with attribute match with value other than “node()” have higher priority
- But this is not a good strategy in general!!

XSLT programming – Two Approaches

1. Unnamed templates + apply-templates
   - The processing is driven by the XSLT parser searching for the most suitable template
2. Named templates + for-each + if + choose
   - The processing is driven by the programmer

- Can be combined arbitrarily