XML Technologies

XSLT: Stylesheet Transformations

Christian Grün
Database & Information Systems Group
Universität Konstanz
Introduction

- We have XQuery... why yet another XML language?
- XQuery focuses on querying, XSLT focuses on transformations
- XSLT is much more popular and wide-spread than XQuery, mainly because it’s older (XSLT 1.0 was finalized in 1999)
- XSLT focuses on the creation and generation of single documents

Process Flow

- XML document is specified as source
- stylesheet specifies transformation
- processor transforms document
- output is the result document
XSL: Extensible Stylesheet Language

- caused by the need for an XML-based stylesheet language
- DSSSL functionality (stylesheets for SGML) was ported to XML
- actually a *family* of three languages:
  - **XPath**: addresses parts of XML documents
  - **XSLT**: transforms XML documents, outputs XML
  - **XSL-FO**: XSL Formatting Objects; specifies the visual *formatting* of XML documents, outputs PDF, RTF, HTML, PNG, and other formats
- XSLT *may* (but need not) be used in an FO pipeline
- XSLT 2.0/3.0: many extensions; low-level functionality similar to XQuery (Saxon query processor supports *both* XQuery and XSLT)
Template Rules

- XSLT is a *declarative* language
- template *rules* define how to handle a node that matches an *XPath pattern*
- template *contents* contain *functional expressions*

XSLT Processing

- *stylesheet* is read and prepared
- a *source tree* is built from the XML document
- the best-matching template is *applied* to the tree’s *root node*
- instructions in the template direct the processor to either
  - *create nodes* in the tree, or
  - *process more nodes* in the source tree
Example: Transformation, XQuery style

Input document:

```
<persons>
  <person id="id1"/>
    <name>Joe</name>
  </person>
  <person id="id2"/>
    <name>Jack</name>
  </person>
</persons>
```

XQuery expression:

```
for $persons in persons
return
<Root>
  for $person in $persons/person
return
<Name ID="{ $person/@id }">
  $person/name/data()
</Name>
</Root>
```

(: alternative writing :)  
```
persons ! <Root>
  person ! <Name ID="{ @id }">name/data()
</Name>
</Root>
```
Example: Transformation via XSLT 1.0

Input document:

```xml
<persons>
  <person id="id1">
    <name>Joe</name>
  </person>
  <person id="id2">
    <name>Jack</name>
  </person>
</persons>
```

XSLT stylesheet:

```xml
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="/persons">
    <Root>
      <xsl:apply-templates select="person"/>
    </Root>
  </xsl:template>
  <xsl:template match="person">
    <Name ID="{ @id }">
      <xsl:value-of select="name" />
    </Name>
  </xsl:template>
</xsl:stylesheet>
```

Result of transformation:

```xml
<Root>
  <Name ID="id1">Joe</name>
  <Name ID="id2">Jack</name>
</Root>
```

Execution via BaseX: ```xslt:transform('input.xml','style.xslt')```
Example: Transformation in the Browser

- all current browsers support XSLT 1.0
- an XML document can also be transformed to HTML by the browser:

```xml
<?xml-stylesheet type="text/xsl" href="html.xslt" ?>
<persons>
  <person id="id1">
    <name>Joe</name>
  </person>
  <person id="id2">
    <name>Jack</name>
  </person>
</persons>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html">
    <xsl:template match="/persons">
      <html>
        <body>
          <ul>
            <xsl:apply-templates select="person"/>
          </ul>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="person">
      <li>Name: <xsl:value-of select="name"/></li>
    </xsl:template>
  </xsl:stylesheet>
```
XSLT 1.0 Elements

```xml
<xsl:stylesheet version='...' xmlns:xsl='...'>...
</xsl:stylesheet>
```

- each XSLT file uses `stylesheet` as root document
- can be used synonymously: `transform`
- mandatory: `version` attribute, xsl namespace

```xml
<xsl:import href='...'/>
```

- child of `stylesheet`; must be specified at the beginning of a document
- can be used to import `other stylesheets`
- imported objects are given a `lower precedence`
- comparable to `class inheritance` in object-oriented languages
<xsl:output ...='...' />

- controls the way the document will be *serialized*
- same as the serialization parameters available in XQuery:
  ```xquery
  declare option output:method "html";
  <html><br/></html> → <html><br/></html>
  ```
- can e.g. be evaluated by the browser, similar to an HTTP response
- allowed output keys (among others):
  - **method**: xml, xhtml, html, text
  - **encoding**: utf-8, iso-8859-1, ...
  - **indent**: yes, no
  - **media-type**: returned mime type
  - **doctype-system**, **doctype-public**: specification of a DTD
<xsl:template match='...'>...</xsl:template>

- entry points for processing XML nodes
- consists of a **pattern** and the template **contents**:
  - the **pattern** is specified in XPath, and is placed in the **match** attribute
  - the **contents** are specified as **child nodes**
- a node that matches a template will be set as local **context node**
- contents may contain **result elements** and **instructions**:
  - **result elements** are simply specified as XML
  - **attributes** may contain XPath expressions wrapped in curly brackets. The output will contain the evaluated XPath result
  - **instructions** are using the XSL namespace. They will be evaluated by the XSLT processor, and the result will be added to the output
<xsl:value-of select='...'/>

- extracts the string value of an XML element and adds it to the output
- the XML target is contained in the select attribute

<xsl:for-each select='...'>...</xsl:for-each>

- iterates through all elements addressed by an XPath expression
- may be used to avoid additional template definitions
- the example on the right is equivalent to the example on Page 7!
<xsl:sort select='...' order='...' data-type='...'/>

- sorts data according to the value of the specified XPath
- only makes sense in loops (e.g. as child of <xsl:for-each>)

<xsl:if test='...'>...</xsl:if>

- conditional test, filters nodes according to the given XPath test
- in many cases, can also be represented as predicate tests:

```xml
<xsl:for-each select="person">
  <xsl:if test="@id='id1'">
    ...
  </xsl:if>
</xsl:for-each>
```
<xsl:choose>
  <xsl:when test='...'>...</xsl:when>
  <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>

- used if *more than one* branch is needed
- multiple *when* branches and single *otherwise* branch allowed

<xsl:number level='...' count='...' format='...'/>

- used within loops/templates
- outputs the current iteration counter (incremented with each call):
  - *level* (single/any/multiple): specifies the way the number is calculated
  - *count*: makes numbering dependent on the specified XPath expression
  - *format*: numering format (examples: *I*, *1*, *A*, ...)

<xsl:apply-templates select='...' />

- applies the **best matching** template to the results of the specified XPath
- if the **select** attribute is omitted, **all child nodes** will be processed
- the currently tested node will be set as **context node**

**Recursive Template Matching**

- if there is **no template** defined for a node, its **children** will be tested
- this is **recursively** repeated until all nodes have been processed
- **texts** and **attribute values** will be output as strings
- the following **hidden template rules** are responsible for this behavior:

```
<xsl:template match="/\|@*">  
  <xsl:apply-templates/>
</xsl:for-each>

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

Stylesheets:

- `<styleSheet xmlns="http://www.w3.org/1999/XSL/Transform" version="1.0">`<template match="/"/>
</stylesheet>

- `<styleSheet xmlns="http://www.w3.org/1999/XSL/Transform" version="1.0">`<template match="name">`<value-of select="/"/>
</template>`
</stylesheet>

- `<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform" version="1.0">`<template match="name">`<value-of select="/"/>
</template>`
</stylesheet>

Input document:

`<persons>`
  `<person id="id1">`
    `<name>`Joe`</name>`
  `</person>`
  `<person id="id2">`
    `<name>`Jack`</name>`
  `</person>`
`</persons>`

Which results are generated by the stylesheets?
Alternative Writings

- similar to XQuery, there is more than way to do it
- imperative vs. declarative style:

```
<xsl:stylesheet>
  <!-- handles the root element -->
  <xsl:template match="/"/>
  <html>
    <body>
      <ul>
        <xsl:for-each select="://person">
          <li>Name: <xsl:value-of select="name"/>
        </li>
        </xsl:for-each>
      </ul>
    </body>
  </html>
</xsl:stylesheet>
```

```
<xsl:stylesheet>
  <!-- handles persons elements -->
  <xsl:template match="persons">
    <html>
      <body>
        <ul>
          <xsl:apply-templates/>
        </ul>
      </body>
    </html>
  </xsl:template>
  <!-- handles person elements -->
  <xsl:template match="person">
    <li><xsl:apply-templates/></li>
  </xsl:template>
  <!-- handles name elements -->
  <xsl:template match="name">
    Name: <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```
XSLT Functions

- all XPath functions can be used within `match`, `test` and `select` attributes
- some functions are specific to XSLT, such as:
  - `document()`: accesses an external document
  - `current()`: returns the current node (will not change during the evaluation of an XPath expression, as happens with the context item.)
  - `system-property()`: returns the value of a system property
  - `unparsed-entity-uri()`: returns the URI of an entity
  - `key()`: returns a node set from an index created via `<xsl:key/>`
- more and more XPath, XQuery and XSLT features are merged:
  - `generate-id()`: returns a string that uniquely identifies a node
  - `format-number()`: converts a number into a string
XSL-FO: Formatting Objects

- XML dialect for *formatting* XML data to screen, paper or other media

**Structure**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
    <fo:layout-master-set>
        <!-- one or more page templates -->
        <fo:simple-page-master master-name="...">
            <!-- page template -->
        </fo:simple-page-master>
    </fo:layout-master-set>
    <!-- one or more page contents -->
    <fo:page-sequence master-reference="...">
        <!-- content of a single page -->
    </fo:page-sequence>
</fo:root>
```
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <!-- page contents -->
  <fo:page-sequence master-reference="A4">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="14pt" font-family="Ubuntu" color="red">
        いらっしゃい！ Добро пожаловать！ أهلاً وسهلاً!
      </fo:block>
      <fo:block text-indent="5mm">
        XSL-FO is a subset of XSL that concentrates on presenting data.
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```