XML and Semantic Web Technologies

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

1. Defining and Applying Templates

2. Creating Nodes of the Result Tree

3. Conditional, Repeated and Grouped Processing

4. Parameterized Templates and Functions

5. Template Rule Priority and Stylesheet Modules

XSL(T) Specification

The XML stylesheet language consists of two parts:

1. XSL Transformations (XSLT) Version 2.0 (Rec 2007/01/23)
   – a query / transformation language for XML

   and

2. Extensible Stylesheet Language (XSL) Version 1.1 (Rec 2006/12/05), containing XSL Formatting Objects (XSL FO)
   – an XML application for descriptions of layout / pages.

XSLT uses XPath 2.0 as expression language.

XSLT has an XML Syntax
(like XML Schema, but contrary to RELAXNG, XPath & XQuery).

The XSLT namespace is

http://www.w3.org/1999/XSL/Transform

XSLT Processing Model

Usually XSLT does 1:1-transformations (1 source document, 1 result document).
But it can do any n:m-transformations (see `document` function and `result-docume`
instruction).

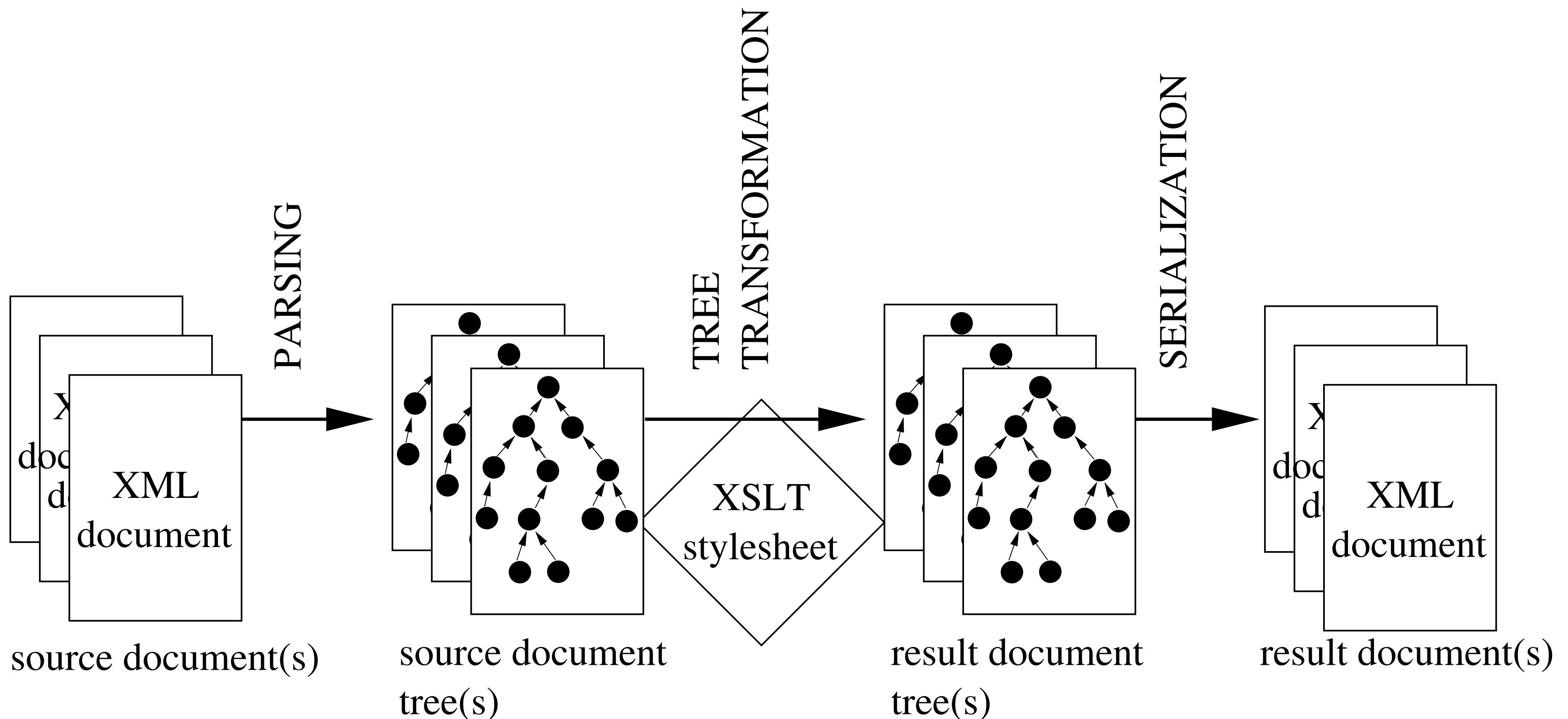

Figure 1: XSLT Processing Model.

Example 1 / Empty Stylesheet (1/3)

```
1 <?xml version="1.1"?>
2 <!-- first ideas -->
3 <?xml-stylesheet href='article.css' type='text/css'?>
4 <article author="John Doe" version="2004/06/07">
5   <title>What <em>others</em> say</title>
6   A <em>short</em><!-- 20 pages--> overview ...
7 </article>
```

Figure 2: Source document.

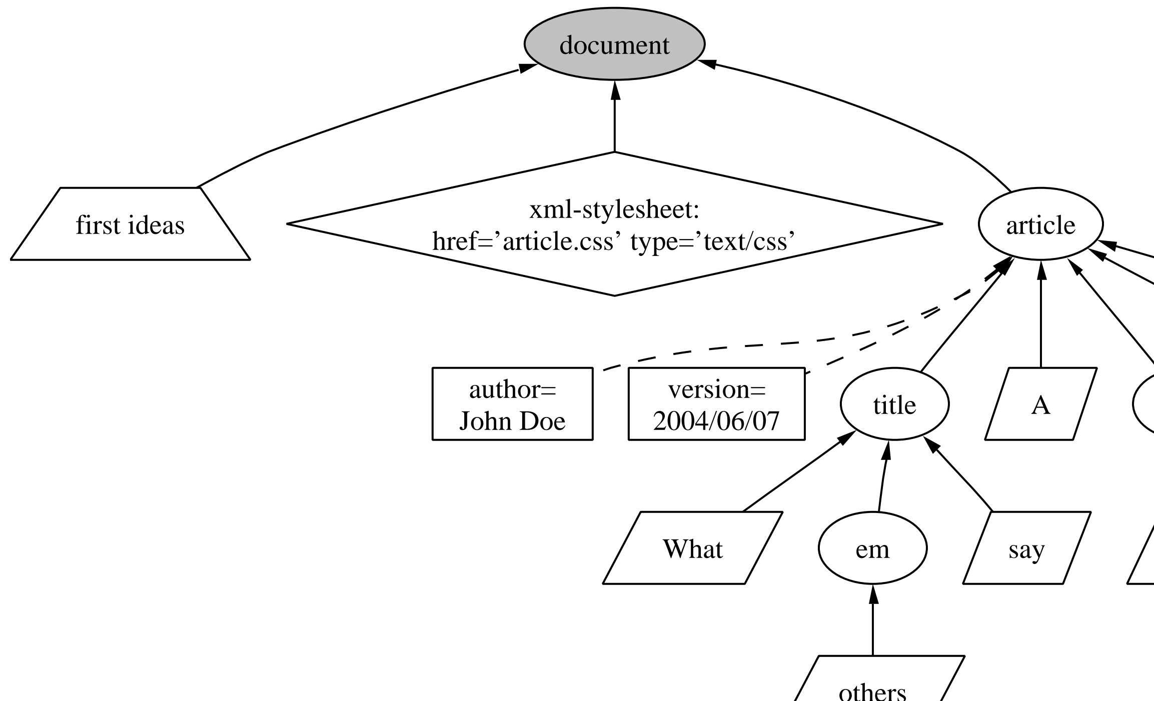

Figure 3: Source Tree.

Example 1 / Empty Stylesheet (2/3)

```
1 <?xml version="1.1"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4 </xsl:stylesheet>
```

Figure 4: The empty stylesheet.

If nodes do not match any template, built-in templates are evoked:

node kind	built-in template action
document, element	apply templates to all (non-attribute) children
attribute, text	output text node containing the node's content
PI, comment, namespace	do nothing

Example 1 / Empty Stylesheet (3/3)

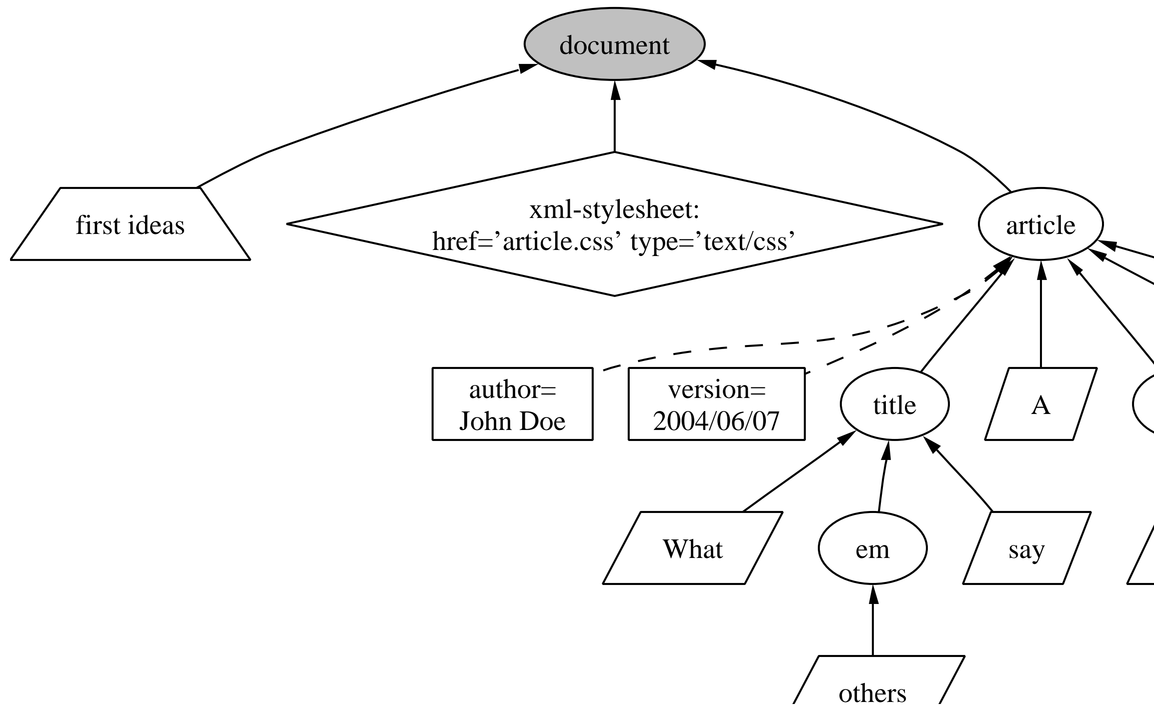

Figure 5: Source Tree.

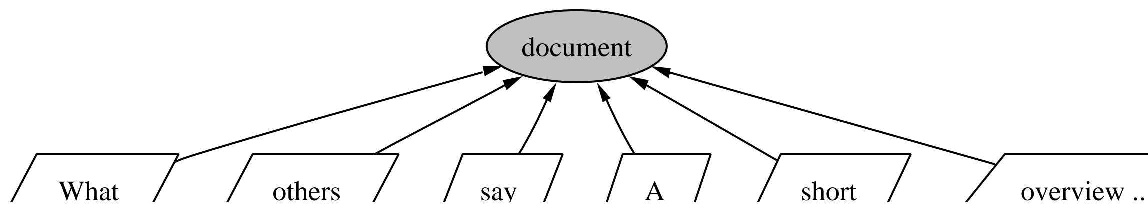

Figure 6: Result Tree.

Example 2 / Terminal Template (1/2)

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4  <xsl:template match="/">
5   <html>
6    <body>
7     A HTML page.
8    </body>
9   </html>
10 </xsl:template>
11
12 </xsl:stylesheet>
```

Figure 7: A stylesheet with a single terminal template.

Example 2 / Terminal Template (2/2)

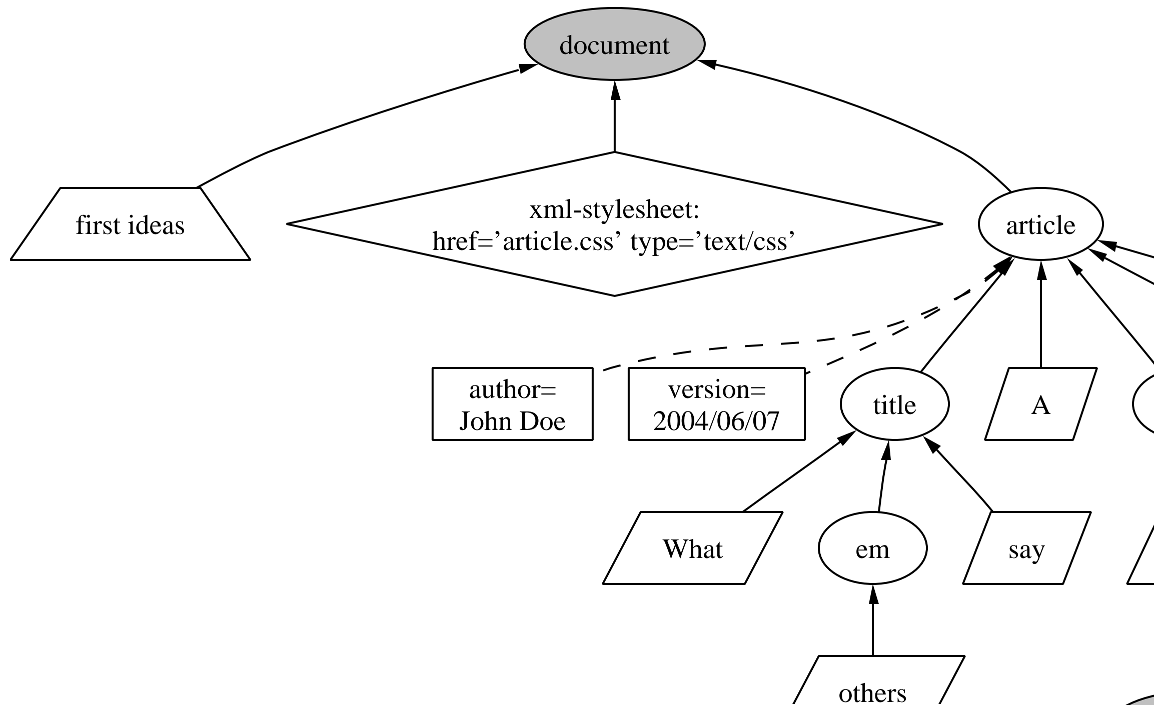

Figure 8: Source Tree.

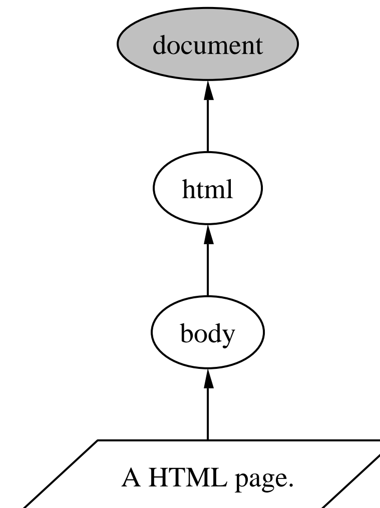

Figure 9: Result Tree.

Example 3 / Recursive Template (1/2)

`xsl:apply-templates` applies templates to all (non-attribute) children of the context node.

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4   <xsl:template match="/">
5     <html>
6       <body>
7         <xsl:apply-templates/>
8       </body>
9     </html>
10  </xsl:template>
11  <xsl:template match="title">
12    <h1><xsl:apply-templates/></h1>
13  </xsl:template>
14 </xsl:stylesheet>
```

Figure 10: A stylesheet with a recursive template.

Example 3 / Recursive Template (2/2)

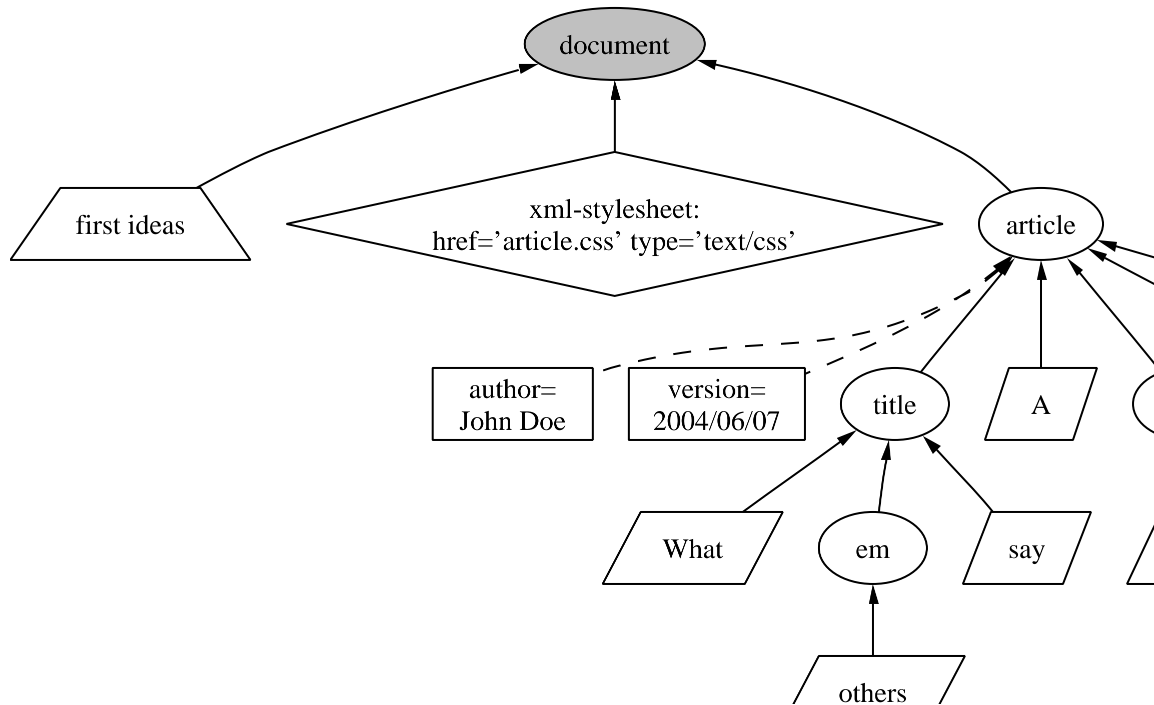

Figure 11: Source Tree.

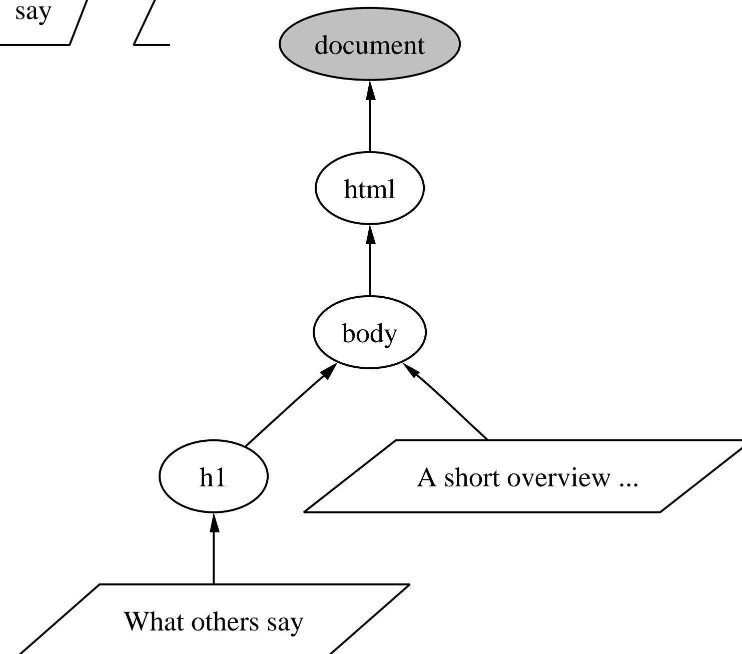

Figure 12: Result Tree.

Performing XSLT Queries / Transformations by Saxon

XSLT queries / transformations can be performed, e.g., by Saxon.

call (with saxon8.jar in classpath):

```
java net.sf.saxon.Transform anarticle.xml recursivetemplate.xsl
```

```
<html>
  <body>

    <h1>What others say</h1>
    A short overview ...

  </body>
</html>
```

Figure 13: Result of the XSLT query / transformation above.

Defining Templates

<xsl:template match = ⟨*pattern*⟩>
<!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:template>

A template consists of

1. **a pattern** that specifices to which nodes the template applies and
2. **a sequence constructor** that specifies the contribution to the result tree.

A Pattern is a `|`- (or `union`-)XPath-expression whose operands are

- path expressions made only of
  - AxisSteps that use only the child or attribute axes or the // operator and
  - arbitrary predicates,
- `id` or `key` function call, where the id must be supplied as a literal or a reference to a variable or parameter, and the key name as literal.

Applying Templates

```
<xsl:apply-templates select? = ⟨expression⟩>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

The `select`-expression specifies the nodes to apply templates to (may be any XPath expression).

If no `select`-expression is given, it defaults to "`child::node()`" (and not to "`.`" !).

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

1. **Defining and Applying Templates**

2. **Creating Nodes of the Result Tree**

3. **Conditional, Repeated and Grouped Processing**

4. **Parameterized Templates and Functions**

5. **Template Rule Priority and Stylesheet Modules**

Literal Result Elements

Element, attribute, text and namespace nodes can be created by **literal result elements**, i.e.,

- elements from a namespace different from the XSLT namespace,
- their attributes from a namespace different from the XSLT namespace (e.g., the empty namespace)
  and their namespace declarations,
- character data as top-level children of sequence constructors or literal result elements.

Comments and PIs cannot be created by literal result elements.

Attribute Value Templates / Computed Attribute Values

Values of literal result attributes can contain **attribute value templates**, i.e., XPath expressions in curly brackets { }.

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3   <xsl:template match="/">
4     <keywords><xsl:apply-templates select="//em"/></keywords>
5   </xsl:template>
6   <xsl:template match="em">
7     <keyword source="{ancestor::article/@author}">
8       <xsl:apply-templates/></keyword>
9   </xsl:template>
10 </xsl:stylesheet>
```

Figure 14: Stylesheet with attribute value template.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <keywords>
3   <keyword source="John Doe">others</keyword>
4   <keyword source="John Doe">short</keyword></keywords>
```

Figure 15: Result document.

Creating Text / Computed Text

Text nodes also can be created by `xsl:text`:

```
<xsl:text>   <!-- Content: #PCDATA -->
</xsl:text>
```

The only difference to literal result text is whitespace handling:

- in literal result text whitespace is normalized
  (i.e., whitespace at beginning and end removed
  and any inter-word whitespace collapsed to a single space),
- in `xsl:text` whitespace is preserved.

Text can be computed by means of the `xsl:value-of` instruction:

```
<xsl:value-of select? = ⟨expression⟩>
  <!-- Content: sequence-constructor -->
</xsl:value-of>
```

There must be either a `select` attribute or a sequence-constructor.

Creating Text / Computed Text

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3   <xsl:template match="article">
4     <article>
5       <author>
6         <fn><xsl:value-of select="substring-before(@author,' ')"/></fn>
7         <sn><xsl:value-of select="substring-after(@author,' ')"/></sn>
8       </author>
9       <title><xsl:value-of select="string(title)"/></title>
10    </article>
11  </xsl:template>
12 </xsl:stylesheet>
```

Figure 16: XSLT Stylesheet computing some text.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <article>
3   <author>
4     <fn>John</fn>
5     <sn>Doe</sn>
6   </author>
7   <title>What others say</title>
8 </article>
```

Figure 17: Result document.

Creating Elements and Attributes / Computed Names

While attribute value templates allow to compute

the values of attributes,

the `xsl:element` and `xsl:attribute` instructions allow to compute

the names of elements and attributes

using the same Attribute Value Template mechanism.

General, attributes of XSLT instructions that allow Attribute Value Templates are marked with curly brackets { } in syntax diagrams.

```
<xsl:element name = { ⟨qname⟩ }>
  <!-- Content: sequence-constructor -->
</xsl:element>
```

```
<xsl:attribute
  name = { ⟨qname⟩ }
  select? = ⟨expression⟩>
<!-- Content: sequence-constructor -->
</xsl:attribute>
```

Creating Elements and Attributes / Example

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3   <xsl:template match="article">
4     <article>
5       <xsl:apply-templates select="@*|title"/>
6     </article>
7   </xsl:template>
8   <xsl:template match="title">
9     <title><xsl:apply-templates/></title>
10  </xsl:template>
11  <xsl:template match="@*">
12    <xsl:element name="{name()}">
13      <xsl:value-of select="string()"/>
14    </xsl:element>
15  </xsl:template>
16 </xsl:stylesheet>
```

Figure 18: XSLT Stylesheet creating elements with variable names.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <article>
3   <author>John Doe</author>
4   <version>2004/06/07</version>
5   <title>What others say</title>
6 </article>
```

Figure 19: Result document.

Creating Namespaces, PIs, Comments

```
<xsl:namespace
  name = { ⟨ncname⟩ }
  select? = ⟨expression⟩>
  <!-- Content: sequence-constructor -->
</xsl:namespace>
```

```
<xsl:processing-instruction
  name = { ⟨ncname⟩ }
  select? = ⟨expression⟩>
  <!-- Content: sequence-constructor -->
</xsl:processing-instruction>
```

```
<xsl:comment select? = ⟨expression⟩>
  <!-- Content: sequence-constructor -->
</xsl:comment>
```

Comments in the stylesheet are not copied in the result tree,
but can be created with the `xsl:comment` instruction.

Comments in the source tree are not copied in the result tree by default,
but can be copied by a template as, e.g.,

```
4 <xsl:template match="comment()">
5   <xsl:comment select="string()"/>
6 </xsl:template>
```

Copying Nodes

Shallow copy of context node/item:

```
<xsl:copy>
  <!-- Content: sequence-constructor -->
</xsl:copy>
```

Deep copy:

```
<xsl:copy-of select = ⟨expression⟩>
```

What is the difference between

```
<?xml version="1.1" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
  <xsl:template match="article">
    <xsl:copy/>
  </xsl:template>
</xsl:stylesheet>
```

Figure 20: Stylesheet making a shallow copy.

```
<?xml version="1.1" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.o
                version="2.0">
  <xsl:template match="article">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Figure 21: Stylesheet making a deep copy.

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

1. Defining and Applying Templates

2. Creating Nodes of the Result Tree

3. Conditional, Repeated and Grouped Processing

4. Parameterized Templates and Functions

5. Template Rule Priority and Stylesheet Modules

Conditional Processing (`if`, `choose`)

<xsl:if test=⟨*expression*⟩>
<!-- Content: sequence-constructor -->
</xsl:if>

<xsl:choose>
<!-- Content: (xsl:when+, xsl:otherwise?)
</xsl:choose>

<xsl:when test = ⟨*expression*⟩>
<!-- Content: sequence-constructor -->
</xsl:when>

<xsl:otherwise>
<!-- Content: sequence-constructor -->
</xsl:otherwise>

For evaluating the test conditions XPath's **effective boolean value** is computed.

`if` does not have an `else` construct (contrary to XPath's `if`-expressions). Instead, one has to use

```
<xsl:choose>
  <xsl:when test="condition"> [if-part] </xsl:when>
  <xsl:otherwise> [else-part] </xsl:otherwise>
</xsl:choose>
```

Conditional Processing (`if`, `choose`) / example (1/2)

```
1 <?xml version="1.1"?>
2 <books>
3   <book isbn="0-596-00420-6">
4     <author>Erik T. Ray</author><title>Learning XML</title><year>2003</year></bo
5   <book isbn="1-565-92580-7">
6     <author>Norman Walsh</author><author>Leonard Muellner</author>
7     <title>DocBook: The Definitive Guide</title><year>1999</year></book>
8   <book isbn="no">
9     <author>Jon Doe</author><author>Alice Smith</author><author>Bob Miller</au
10    <title>About something</title><year>1990</year></book>
11 </books>
```

Figure 22: Sample document.

Conditional Processing (`if`, `choose`) / example (2/2)

```
4  <xsl:template match="books">
5   <html><body><ol><xsl:apply-templates/></ol></body></html>
6  </xsl:template>
7  <xsl:template match="book">
8   <li>
9   <xsl:value-of select="author[1], if (count(author)=2) then ('and', author[2])
10                            else if (count(author)=3) then 'et al.' else ''"/>
11    : <xsl:value-of select="title"/>, <xsl:value-of select="year"/>.
12   <xsl:if test="@isbn!='no'"><em>ISBN <xsl:value-of select="@isbn"/></em>.</xsl:
13   </li>
14 </xsl:template>
```

Figure 23: XSLT stylesheet with XPath `if` expressions and XSLT `if` instruction (excerpt).

```
4   <li>Erik T. Ray: Learning XML, 2003.
5      <em>ISBN 0-596-00420-6</em>.</li>
6   <li>Norman Walsh and Leonard Muellner:
7      DocBook: The Definitive Guide, 1999.
8      <em>ISBN 1-565-92580-7</em>.</li>
9   <li>Jon Doe et al.: About something, 1990.</li>
```

Figure 24: Result document (excerpt).

Repeated Processing (`for-each`)

<xsl:for-each select = ⟨*sequence-expression*⟩>
 <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each>

Repeated Processing (`for-each`) / Example (1/2)

```
1 <?xml version="1.1"?>
2 <books>
3   <book isbn="0-596-00420-6">
4     <author>Erik T. Ray</author><title>Learning XML</title><year>2003</year></bo
5   <book isbn="1-565-92580-7">
6     <author>Norman Walsh</author><author>Leonard Muellner</author>
7     <title>DocBook: The Definitive Guide</title><year>1999</year></book>
8   <book isbn="1-565-92051-1">
9     <author>Norman Walsh</author>
10    <title>Making TeX Work</title><year>1994</year></book>
11 </books>
```

Figure 25: Example document.

Repeated Processing (`for-each`) / Example (2/2)

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4   <xsl:template match="books">
5     <html><body>
6     Authors: <ol>
7       <xsl:for-each select="distinct-values(//author)">
8         <li><xsl:value-of select="."/></li>
9       </xsl:for-each>
10    </ol></body></html>
11  </xsl:template>
12 </xsl:stylesheet>
```

Figure 26: Stylesheet using `for-each` instruction.

```
1 <html>
2   <body>
3     Authors:
4     <ol>
5       <li>Erik T. Ray</li>
6       <li>Norman Walsh</li>
7       <li>Leonard Muellner</li>
8     </ol>
9   </body>
10 </html>
```

Figure 27: Result document.

Sorting

```
<xsl:sort
  select? = ⟨expression⟩
  data-type? = { "text" | "number" | ⟨qname-but-not-ncname⟩ }>
  order? = { "ascending" | "descending" }
  <!-- Content: sequence-constructor -->
</xsl:sort>
```

A **sort key specification** (=sequence of `sort` instructions) may be used

- with xsl:for-each or xsl:apply-templates to process a sequence in sorted order or
- with xsl:perform-sort to create a sequence in sorted order.

Create a sorted sequence:

```
<xsl:perform-sort select? = ⟨expression⟩>
  <!-- Content: (xsl:sort+, sequence-constructor) -->
</xsl:perform-sort>
```

Sorting / Example

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4   <xsl:template match="books">
5     <html><body>
6     Authors: <ol>
7       <xsl:for-each select="distinct-values(//author)">
8         <xsl:sort select="substring-after(.,' ')"/>
9         <xsl:sort select="substring-before(.,' ')"/>
10        <li><xsl:value-of select="."/></li>
11      </xsl:for-each>
12    </ol></body></html>
13  </xsl:template>
14 </xsl:stylesheet>
```

Figure 28: Stylesheet sorting by author.

```
1 <html>
2   <body>Authors:
3     <ol>
4       <li>Leonard Muellner</li>
5       <li>Erik T. Ray</li>
6       <li>Norman Walsh</li>
7     </ol>
8   </body>
9 </html>
```

Figure 29: Result document.

Sequential Queries (1/2)

```
1  <html>
2    <body>
3      Authors:
4      <ol>
5        <li>Leonard Muellner
6          <ol>
7            <li>DocBook: The Definitive Guide,
8              1999.
9            </li>
10         </ol>
11       </li>
12       <li>Erik T. Ray
13         <ol>
14           <li>Learning XML,
15             2003.
16           </li>
17         </ol>
18       </li>
19       <li>Norman Walsh
20         <ol>
21           <li>Making TeX Work,
22             1994.
23           </li>
24           <li>DocBook: The Definitive Guid
25             1999.
26           </li>
27         </ol>
28       </li>
29     </ol>
30   </body>
31 </html>
```

Figure 30: Intended Result document.

```
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3  <xsl:variable name="books" select="/descendant::book"/>
4  <xsl:template match="books">
5   <html><body>Authors: <ol>
6    <xsl:for-each select="distinct-values(//author)">
7     <xsl:variable name="author" select="."/>
8     <xsl:sort select="substring-after(.,' ')"/>
9     <xsl:sort select="substring-before(.,' ')"/>
10    <li><xsl:value-of select="."/>
11     <ol>
12      <xsl:for-each select="$books[$author=author]">
13       <xsl:sort select="year" data-type="number"/>
14       <li><xsl:apply-templates select="title"/>,
15         <xsl:apply-templates select="year"/>.</li>
16      </xsl:for-each>
17     </ol></li>
18    </xsl:for-each>
19   </ol></body></html>
20  </xsl:template>
21 </xsl:stylesheet>
```

Figure 31: XSL Stylesheet producing the intended result using nested `for-each` instructions.

Grouping

```
<xsl:for-each-group
  select = ⟨expression⟩
  group-by? = ⟨expression⟩
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each-group>
```

`for-each-group`

- groups the selected items in (maybe overlapping) sequences (**groups**), one for each distinct value of the group-by expression (**keys**) and
- executes its sequence-constructor once for each group.

Additional XPath functions:

- `current-grouping-key()` returns the value of the group-by expression of the current group,
- `current-group()` returns the current group as sequence (of nodes).

```
1  <?xml version="1.1" encoding="utf-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3   <xsl:template match="books">
4    <html><body>
5    Authors: <ol>
6     <xsl:for-each-group select="//book" group-by="author">
7      <xsl:sort select="substring-after(current-grouping-key(),' ')"/>
8      <xsl:sort select="substring-before(current-grouping-key(),' ')"/>
9      <li><xsl:value-of select="current-grouping-key()"/>
10      <ol>
11       <xsl:for-each select="current-group()">
12        <xsl:sort select="year" data-type="number"/>
13        <li><xsl:apply-templates select="title"/>,
14          <xsl:apply-templates select="year"/>.</li>
15       </xsl:for-each>
16      </ol></li>
17     </xsl:for-each-group>
18    </ol></body></html>
19   </xsl:template>
20  </xsl:stylesheet>
```

Figure 32: XSL Stylesheet producing the intended result efficiently using a `group` instruction.

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

1. Defining and Applying Templates

2. Creating Nodes of the Result Tree

3. Conditional, Repeated and Grouped Processing

4. Parameterized Templates and Functions

5. Template Rule Priority and Stylesheet Modules

Variables

```
<xsl:variable
  name = ⟨qname⟩
  select? = ⟨expression⟩
  as? = ⟨sequence-type⟩>
  <!-- Content: sequence-constructor -->
</xsl:variable>
```

Variables can be used in XPath expressions via XPath syntax `$`⟨*NCName*⟩.

Variables can be

- **global** if declared top-level
  – visible everywhere,
  or
- **local** if declared in a sequence constructor
  – visible for all its following siblings and their descendants.

Once set a variable cannot subsequently be changed
(so a better name would be named constants).

Variables / Example

```
1 <?xml version="1.1" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4  <xsl:template match="books">
5   <html><body><ol><xsl:apply-templates/></ol></body></html>
6  </xsl:template>
7  <xsl:template match="book">
8   <li>
9   <xsl:variable name="numAuthors" select="count(author)"/>
10  <xsl:value-of select="author[1], if ($numAuthors=2) then ('and', author[2])
11                           else if ($numAuthors=3) then 'et al.' else ''"/>
12    : <xsl:value-of select="title"/>, <xsl:value-of select="year"/>.
13  <xsl:if test="@isbn!='no'"><em>ISBN <xsl:value-of select="@isbn"/></em>.</xsl:
14  </li>
15 </xsl:template>
16 </xsl:stylesheet>
```

Figure 33: Stylesheet with variable.

Defining XPath Functions

```
<xsl:function
  name = ⟨qname⟩
  as? = ⟨sequence-type⟩>
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:function>
```

Functions are not polymorphic, but identified by name and arity.

Function names **must** be put in their own namespace.

As normally namespaces of functions should not be declared in the result document, the

`exclude-result-prefixes`

attribute of the `stylesheet` element allows to specify a list of prefixes to exclude.

Defining XPath Functions / Parameters

```
<xsl:param
  name = ⟨qname⟩
  as? = ⟨sequence-type⟩
  required? = "yes" | "no"
  select? = ⟨expression⟩>
  <!-- Content: sequence-constructor -->
</xsl:param>
```

If `required='no'`, `select` or the sequence constructor specify the default value.

Parameters can be used in all XPath expressions
in the same manner as variables via XPath syntax `$`⟨*NCName*⟩.

```
1 <?xml version="1.1"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
3         xmlns:f="http://www.cgnm.de/xml/xslt/functions"
4         xmlns:xs="http://www.w3.org/2001/XMLSchema"
5         exclude-result-prefixes="f xs">
6  <xsl:function name="f:shortauthor" as="xs:string">
7   <xsl:param name="author" as="node()*"/>
8   <xsl:value-of select="$author[1], if (count($author)=2) then ('and', $author[2])
9                  else if (count($author)=3) then 'et al.' else ''"/>
10 </xsl:function>
11 <xsl:template match="books">
12  <html><body><ol><xsl:apply-templates/></ol></body></html>
13 </xsl:template>
14 <xsl:template match="book">
15  <li><xsl:value-of select="f:shortauthor(author)"/>
16     : <xsl:value-of select="title"/>, <xsl:value-of select="year"/>.
17     <xsl:if test="@isbn!='no'"><em>ISBN <xsl:value-of select="@isbn"/></em>.</x
18  </li>
19 </xsl:template>
20 </xsl:stylesheet>
```

Figure 34: Stylesheet with an XPath function.

Template Parameters

```
<xsl:with-param
  name = ⟨qname⟩
  select? = ⟨expression⟩
  as? = ⟨sequence-type⟩>
  <!-- Content: sequence-constructor -->
</xsl:with-param>
```

I. XML / 5. XML Stylesheet Language Transformations (XSLT)

1. Defining and Applying Templates

2. Creating Nodes of the Result Tree

3. Conditional, Repeated and Grouped Processing

4. Parameterized Templates and Functions

5. Template Rule Priority and Stylesheet Modules

Template Rule Priority

If more than one template rule matches a given node,
the template to apply is selected by **priority**.

Priority can be specified explicitly via the attribute

`priority`

of the `template` instruction (real number).

If for a given node, more than one pattern has maximal priority,
an error can be thrown or the template last declared be choosen.

Template Rule Priority / Example

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 version="2.0">
4   <xsl:template match="author">
5     a <xsl:apply-templates/>
6   </xsl:template>
7
8   <xsl:template match="node()">
9     b <xsl:apply-templates/>
10  </xsl:template>
11 </xsl:stylesheet>
```

Figure 35: Template rules with explicitly specified priorities.

Default Template Rule Priority

If no priority is specified, a default priority is assigned as follows:

priority	pattern
0.5	any other case, e.g., with predicates, specifying ancestry, etc.
0.25	element or attribute test specifying name and type, i.e., element(⟨*QName*⟩,⟨*QName*⟩), attribute(⟨*QName*⟩,⟨*QName*⟩)
0	element or attribute test specifying either name or type, i.e., ⟨*QName*⟩, @⟨*QName*⟩, element(⟨*QName*⟩,*), element(*, ⟨*QName*⟩), et
-0.25	element or attribute test specifying either namespace or local name, i.e., *:⟨*QName*⟩, ⟨*QName*⟩:*
-0.5	kind test specifying neither name nor type, and any other node test, i.e., /, *, `@*`, element(*,*), etc.

Template rules with |-patterns are handled as separate template rules per operand.

In most cases, the more selective a pattern is, the higher its priority
(the most selective pattern takes precedence).

The default priority is assigned based on **syntactic properties**,
i.e., `node()[true()]` has a higher priority than say `author`.

Default Template Rule Priority / Example

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  version="2.0">
4    <xsl:template match="author">
5      a <xsl:apply-templates/>
6    </xsl:template>
7
8    <xsl:template match="node()[true()]">
9      b <xsl:apply-templates/>
10   </xsl:template>
11 </xsl:stylesheet>
```

Figure 36: Template rules with questionable default priorities.

Including and Importing Stylesheet Modules

<xsl:include href=⟨*uri-reference*⟩ />

<xsl:import href=⟨*uri-reference*⟩ />

Stylesheets can include or import other stylesheets (**stylesheet modules**).

`import` differs from `include` only in:

template rules and other declarations in the importing module take precedence over template rules and declarations in the imported module.

References