XML Navigation, Transformation

XSLT—Transforming XML Documents

Lecture "XML in Communications"
Chapter 9

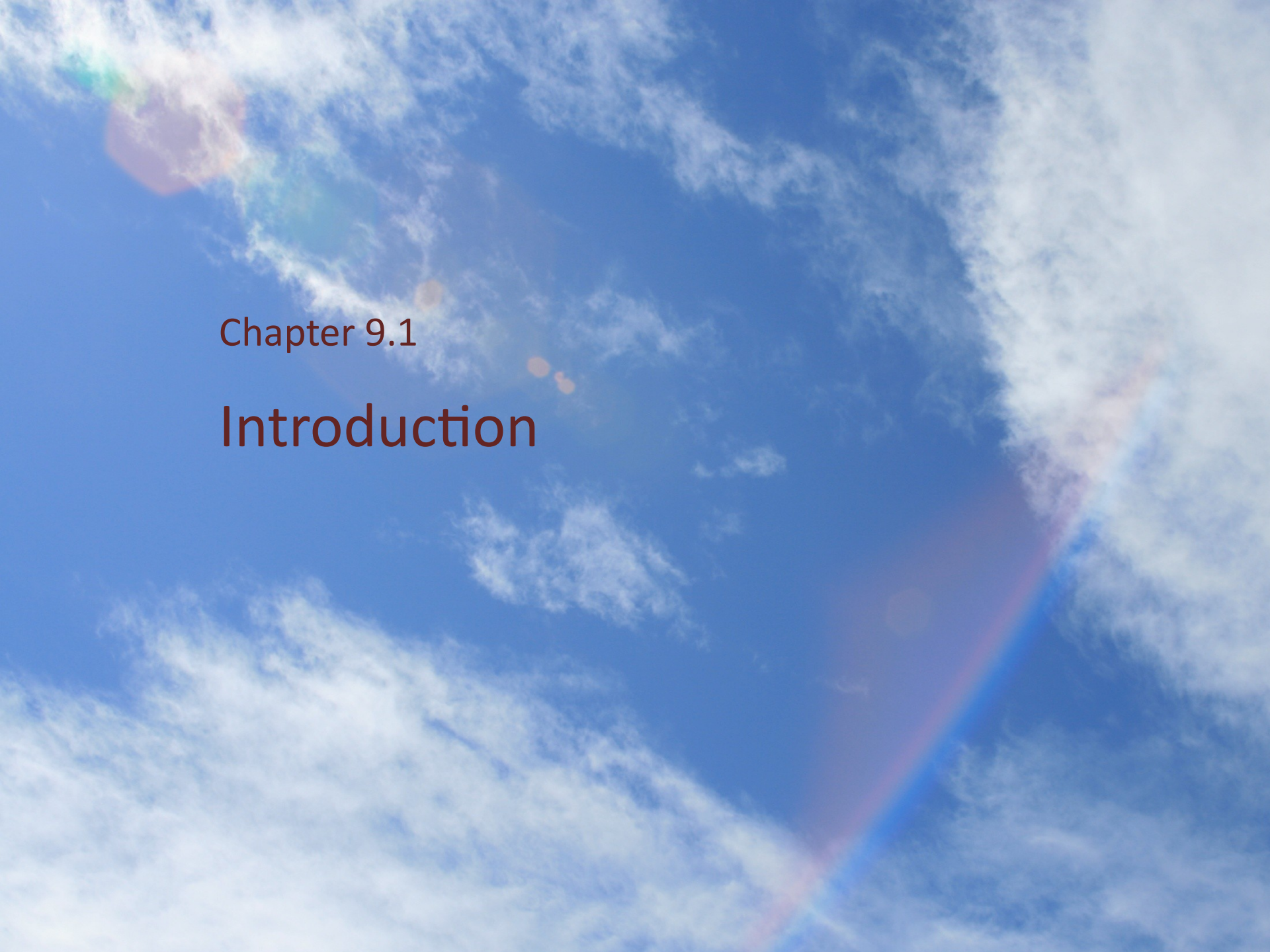
Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel

Recommended Reading

- M. Kay (Editor): XSL Transformations (XSLT) Version 2.0.
  W3C Recommendation 23 January 2007.
  http://www.w3.org/TR/xslt20/

Overview

1. Introduction
2. Models for Tree Editing
3. Stylesheet Structure
4. XSLT Programming Examples
5. XSLT-based Schematron Implementation

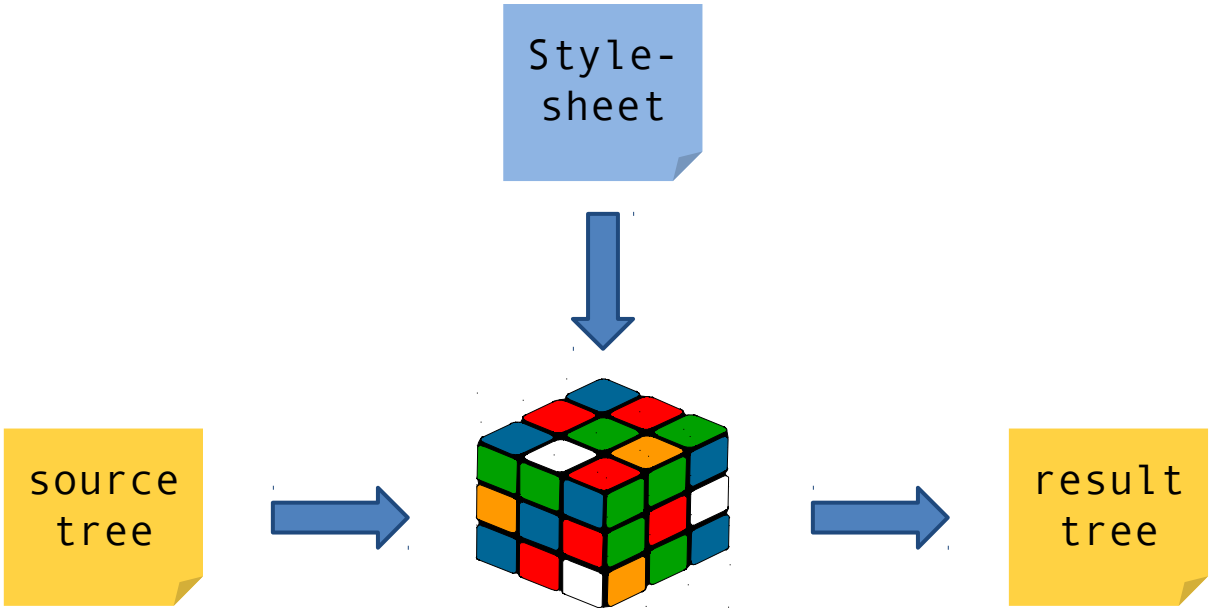


Chapter 9.1

Introduction

Introduction

- XSLT: *eXtensible Stylesheet Language for Transformations*
  - Rule-based (declarative) language for transformations
  - Transformation of an XML *source tree* into an (arbitrary) *result tree*



Transformation process

Introduction

- Where does XSLT fit?
  - Dependency path: XML → XPath → XSLT
- Status
  - Full W3C Recommendation, in wide use
  - Version 2.0 available since 2007: http://www.w3.org/TR/xslt20/

Introduction

- Transformation?
  - generate (new) constant content,
  - suppress content,
  - move subtrees (e.g., swap day/month in a date),
  - copy subtrees (e.g., copy section titles into tables of contents),
  - sort content,
  - general transformations that compute new from given content.

Introduction

- Where to transform documents?
  - in the server:
    A server program, such as a Java servlet, can use a stylesheet to transform a document automatically and serve it to the client.
  - in the client:
    A client program, such as a browser, can perform the transformation and render the transformed document to the user.
  - with a separate program:
    Several standalone programs (XSLT processors), e.g. Saxon, may perform XSLT transformations.

Introduction

- Stylesheet location
  - Specify a stylesheet in the document preamble using a processing instruction

```
<?xml version="1.0" ?>
…
<?xml-stylesheet  type="text/xsl"
                  href="http://xyz.edu/Report/report.xsl" ?>

<Report Date="2008-11-11">
  …
</Report>
```

Chapter 9.2

Models for Tree Editing

Models for Tree Editing

- Different kinds of approaches to tree editing
  - Functional
  - Rewrite rule-based
  - Template-based
  - Imperative

Models for Tree Editing

- Functional tree rewriting
  - Recursive processing
  - Invoke start function at the root, construct a new tree
  - Can think of this as "node functions"
  - Result is "compositional" — substitution is generally nested
  - Side effects often avoided: caching values, clarity

Models for Tree Editing

- Rule-based (rewriting systems)
  - A transformation is defined by a list of pattern/result pairs
  - Each is a piece of a tree with "holes" (variables)
  - A match leads to replacement of the matched tree nodes by a result tree
  - Variables shared between pattern and result allow preservation and re-arrangement of arbitrary data
  - Powerful, incremental definitions
  - Non-deterministic processing

Models for Tree Editing

- Template based processing
  - Starting point is a pattern document.
  - This model is very familiar from many web-based systems.
  - It contains literal results interleaved
    with queries and sometimes imperative code.
  - Well-suited to repetitive or rigid structures.
  - Often requires extensions to deal with recursion and looping.

Models for Tree Editing

- Imperative
  - Parser calls imperative code, which uses
    - stacks
    - global variables
    - explicit output commands
  - Results are obtained by side effects.
  - Reasoning about the program may be hard, but creating it often starts out easily.
  - This approach makes it easy to create non-XML, or ill-formed XML documents.

Models for Tree Editing

- What's the biggest drawback to tree editing?
  - Buffering!
    - You need a copy of the tree to edit:
      a document entirely in-memory!
    - Doing this from secondary storage is fairly subtle,
      and has its own performance penalties.

Models for Tree Editing

- What side is XSLT on?
  - Rule-based substitution
  - Results are like template languages
  - XPath addressing also looks like queries in traditional template languages
  - Limited non-determinism

Chapter 9.3

Stylesheet Structure

Stylesheet Structure

- An XSLT stylesheet is an XML document
  - document element: `xsl:stylesheet` or `xsl:transform`
  - version attribute values: `1.0` or `2.0`
  - namespace URI: `http://www.w3.org/1999/XSL/Transform`
  - MIME media type: `text/xml` or `application/xml`

Stylesheet Structure

- Basic structure
  - Bunch of templates of the form:

```
<xsl:template match="pattern">
   … rules …
</xsl:template>
```

    - `match` attribute specifies elements in *source tree*
    - `rules` specify contribution to *result tree*
  - Rules are "instantiated" per matching node.

Stylesheet Structure

- Closer look to "templates"
  - The `match` attribute
    - derived from (abbreviated) XPath
    - selects the node set in the source tree the template rules are applied to
  - Rules generate
    - Literal output
    - Results of XSLT functions
    - Results of further template applications
    - Results of queries on the document

Stylesheet Structure

- A trivial stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    x
  </xsl:template>
</xsl:stylesheet>
```

  – replaces the source tree by "x"

Stylesheet Structure

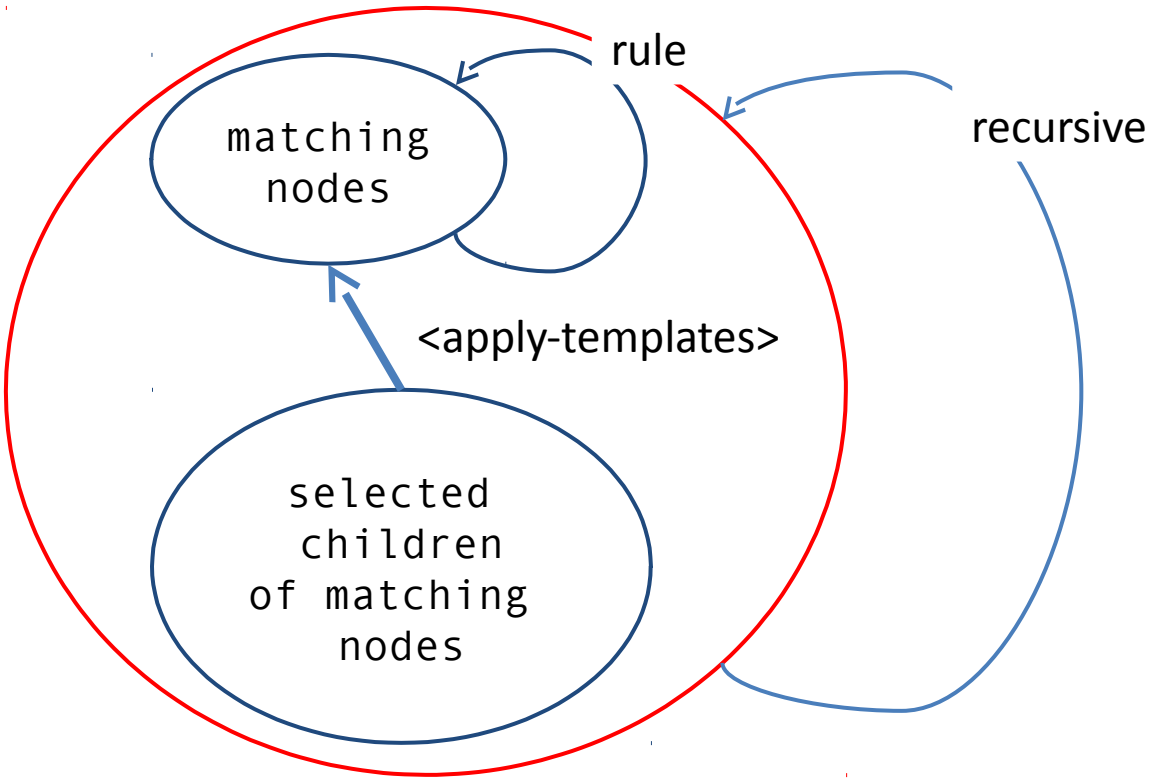
- Result tree creation
  - Literals – any element not in xsl namespace
  - <xsl:text> send content directly to output (retain whitespace)
  - <xsl:value-of> expression processing
  - <xsl:copy> copy current node into result tree
  - <xsl:element> instantiate an element
  - <xsl:attribute> instantiate an attribute
  - …

Stylesheet Structure

- Result tree creation (cont'd.)
  - Further template applications
    - controlled by the <xsl:apply-templates> element:
      apply matching templates to the current element's child nodes.
    - <xsl:apply-templates> element may have a
      select attribute: reduce set of nodes
    - If no template available after matching and selection:
      add string value of remaining nodes to result tree

Stylesheet Structure

- Result tree creation (cont'd.)



Stylesheet Structure

- Conflict Resolution
  - If several templates match: use the best matching template, i.e. the template with the smallest (by inclusion) set of matching nodes.
  - If several of those:
    - Imported templates have lower priority.
    - Take priority attribute value into account if available.
    - If several patterns with equal priority → error.
  - If no template matches, use the matching default template—later.

Stylesheet Structure

- XSLT processing uses built-in 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()"/>
```

  – "The stylesheet author can override a built-in template rule by including an explicit template rule."

Chapter 9.4

XSLT Programming Examples

XSLT Programming Examples

- Stylesheet elements

xsl:import	href = uri-reference	import of further stylesheets; existing definitions and rules are overwritten
xsl:include	href = uri-reference	same as import, but without overwriting
xsl:strip-space	elements = tokens	controls removal of text nodes with white space only
xsl:preserve-space	elements = tokens	
xsl:output	method = "xml" | "html" | "text" …	output syntax
xsl:key	name = …	declaration of cross references by patterns
xsl:decimal-format		definition of formats for decimals
xsl:namespace-alias		definition of a namespace alias
xsl:attribute-set		definition of attribute sets
xsl:variable		declaration of variables and parameters
xsl:param		
xsl:template		

XSLT Programming Examples

- Stylesheet elements

<xsl:apply-templates select = node-set-expression …/>	In the absence of a select attribute, the xsl:apply-templates instruction processes all of the children of the current node, including text nodes. A select attribute can be used to process nodes selected by an expression instead of processing all children.
<xsl:for-each select = node-set-expression/>	iteration
<xsl:value-of select = string-expression …/>	inserts string values into result tree
<xsl:copy-of select = expression />	copies node set
<xsl:sort select = string-expression …/>	sorting a node set
<xsl:element name = { qname } namespace = { uri-reference } use-attribute-sets = qnames> <!-- Content: template --> </xsl:element>	generates elements in result tree

XSLT Programming Examples

- Sample XSLT document

```
<xsl:template match="/">
  <html><body><h1>
    <xsl:value-of select="message"/>
  </h1></body></html>
</xsl:template>
```

  - The `<xsl:template match="/">` chooses the root
  - The `<html><body><h1>` is written to the output file
  - The contents of the `message` element is written to the output file
  - The `</h1></body></html>` is written to the output file
  - The result is: `<html><body><h1>Howdy!</h1></body></html>`

XSLT Programming Examples

- How XSLT works
  - The XML text document is read in and stored as a tree of nodes
  - The `<xsl:template match="/">` template selects the entire tree
  - The rules within the template are applied to the matching nodes, thus changing the structure of the tree
  - Unmatched parts of the XML tree are not changed
  - After the template is applied, the tree is written out as a text document

XSLT Programming Examples

- Some functions

  `xsl:value-of`

  - `<xsl:value-of select="XPath expression"/>`
    selects the contents of an element and adds it to the output stream
  - The `select` attribute is required.

XSLT Programming Examples

- Some functions

  `xsl:for-each`

  – loop statement

  – The syntax is

```
<xsl:for-each select="XPath expression">
      Text to insert and rules to apply
</xsl:for-each>
```

  – Example: Make an unordered list of book titles

```
<ul>
  <xsl:for-each select="//book">
    <li><xsl:value-of select="title"/></li>
  </xsl:for-each>
</ul>
```

XSLT Programming Examples

- Sample XML file

```
<?xml version="1.0" ?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J.K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

XSLT Programming Examples

- More precise

```
<xsl:for-each select="//book">
  <li>
  <xsl:value-of
  select="title[../author='J.K. Rowling']"/>
  </li>
</xsl:for-each>
```

  - This will output <li> and for every book, so we will get empty bullets for authors other than J.K. Rowling
  - There is no obvious way to solve this with just xsl:value-of

XSLT Programming Examples

- Conditional transformation

```
<xsl:for-each select="//book">
  <xsl:if test="author='J.K. Rowling'">
    <li>
      <xsl:value-of select="title"/>
    </li>
  </xsl:if>
</xsl:for-each>
```

  - xsl:if allows us to include content
    if a given condition (in the test attribute) is true
  - This does work correctly!

XSLT Programming Examples

- Choose
  - The `xsl:choose ... xsl:when ... xsl:otherwise` construct is XSLT's equivalent of Java's `switch ... case ... default` statement

```
<xsl:choose>
     <xsl:when test="some condition">
           ... some code ...
     </xsl:when>
     <xsl:otherwise>
           ... some code ...
     </xsl:otherwise>
</xsl:choose>
```

XSLT Programming Examples

- Sorting elements
  - You can place an `xsl:sort` inside an `xsl:for-each`
  - The attribute of the sort tells what field to sort on
  - Example

```
<ul>
<xsl:for-each select="//book">
  <xsl:sort select="author"/>
  <li>
    <xsl:value-of select="title"/> by
    <xsl:value-of select="author">
  </li>
</xsl:for-each>
</ul>
```

  - This example creates a list of titles *and* authors, sorted by author

XSLT Programming Examples

- Creating tags from XML data
  - Suppose the XML contains

```
<name>jze's Home Page</name>
<url>http://www.uni-kiel.de/~jze</url>
```

  - and we want to turn this into

```
<a href="http://www.uni-kiel.de/~jze">
jze's Home Page</a>
```

XSLT Programming Examples

- Creating tags, solution 1
  - `<xsl:attribute name="...">` adds the named attribute to the enclosing tag
  - The *value* of the attribute is the content of this tag

```
<a>
  <xsl:attribute name="href">
    <xsl:value-of select="url"/>
  </xsl:attribute>
  <xsl:value-of select="name"/>
</a>
```

  - Result:

```
<a href="http://www.uni-kiel.de/~jze">
jze's Home Page</a>
```

XSLT Programming Examples

- Creating tags, solution 2
  - An attribute value template (AVT) consists of braces { } inside the attribute value
  - The content of the braces is replaced by its value

```
<a href="{url}">
        <xsl:value-of select="name"/>
</a>
```

  - Result:

```
<a href="http://www.uni-kiel.de/~nl">
jze's Home Page</a>
```

XSLT Programming Examples

- Calling named templates
  - You can name a template, then call it, similar to the way you would call a method in Java
  - The named template:

```
<xsl:template name="myTemplateName">
        ...body of template...
</xsl:template>
```

  - A call to the template:

```
<xsl:call-template name="myTemplateName">
        ...parameters...
    </xsl:call-template>
```