# Information Systems

WS 2005, JKU Linz
Course 11: XPath and XSLT

Gábor Bodnár

URL: `http://www.risc.uni-linz.ac.at/education/courses/ws2005/is/`

## Overview

- XPath: data model, location paths.
- XSLT: stylesheets, templates, additional features.

# XPath

XPath is a language whose primary purpose is to provide common syntax and functionality to address parts of XML documents.

It is a member of the XSL (eXtensible Stylesheet Language) family.

XPath operates on the logical structure of an XML document and uses a syntax that resembles to the path constructions in URIs.

XPath models an XML document as a tree of nodes (e.g. elements, attributes, namespaces, etc.)

XPath expressions can compute strings, numbers, sets of nodes from the data of XML documents.

# XPath Data Model

XPath models an XML document as a tree of nodes of the types: root, element, text, attribute, namespace, processing instruction, comment.

The tree structure is an extension of the tree structure of the element nesting of the XML document.

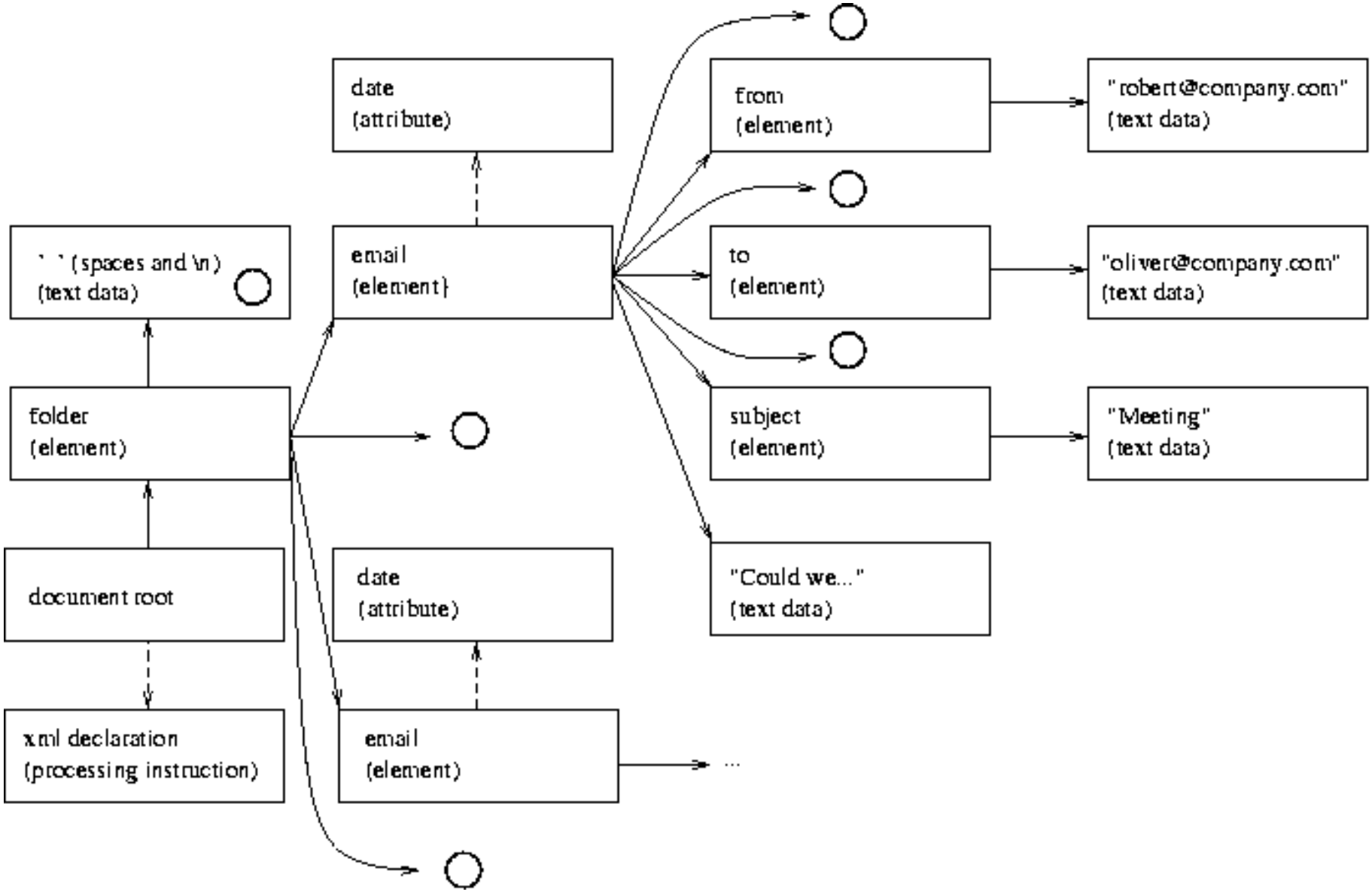
The nodes can be considered in *document order*, which is given by the order of the first characters of the nodes in the XML document.

```
<folder>
  <email date='20 Aug 2003'> ... </email>
  <email date='21 Aug 2003'> ... </email>
</folder>
```

# Example

```
<?xml version="1.0" encoding="UTF-8"?>
<folder>
  <email date='20 Aug 2003'>
    <from>robert@company.com</from>
    <to>oliver@company.com</to>
    <subject>Meeting</subject>
    Could we meet ...
  </email>
  <email date='21 Aug 2003'>
    <from>oliver@company.com</from>
    <to>rob@company.com</to>
    <cc>amy@company.com</cc>
    <subject>Re: Meeting</subject>
    On 20 Aug 2003 rob@company.com wrote ...
  </email>
</folder>
```

# Example


date
(attribute)
from
(element)
"robert@company.com"
(text data)
' ' (spaces and \n)
(text data)
email
(element)
to
(element)
"oliver@company.com"
(text data)
folder
(element)
subject
(element)
"Meeting"
(text data)
"Could we..."
(text data)
document root
date
(attribute)
xml declaration
(processing instruction)
email
(element)
...

# String Values

Every node has a *string value*, which is computed as follows:

- For root and element nodes it is the concatenation of string values of the descendants in document order.
- For attribute nodes it is the value of the attribute.
- For text nodes it is just the character data contained by them.
- For namespace nodes it is the corresponding URI.
- For comment nodes it is the text of the comment.

# Expression Context

XPath expressions are always evaluated in a context, which is described by the following parameters:

- context node,
- context size,
- context position,
- variable bindings,
- namespace declarations.

# Location Paths

Location paths are special expressions for selecting a set of nodes.

A location path consists of location steps composed together from left to right and separated by '/'.

An absolute location path is one that starts with a '/'.

Relative location paths are defined always with respect to the *context node*.

## Example

The node selection is analogous to the file selection in a Unix-like file system.

```
../reports/*/summary
```

# Location Steps

Location steps have the following parts:

**axis** It specifies the (in-tree) relationship between the context node and the nodes selected by the location step:

Available axes: child, descendant, parent, ancestor, following-sibling, preceding-sibling, following, preceding, attribute, namespace, self, descendant-or-self, ancestor-or-self.

**node test** Specifies the node type for the nodes selected by the location step (separated by `::` from the axis).

**predicates** It specifies further expressions with boolean value, to refine the selected node set (enclosed in `[ ]`).

# Example

This location path (using now the "long notation") selects all attributes of all the `email` elements.

```
/child::folder/child::email/attribute::*
```

This selects only the attribute of the first `email` element in the XML document.

```
/child::folder/child::*[position()=1]/@*
```

What does this do?

```
/folder/email/to[string()='rob@company.com']/../@date
```

# XPath Expressions

Simple expressions: numerical and string literals, variable references, function calls.

Values can be bound to variable names in XSLT. The value of `x` can be retrieved by `$x`.

Basic arithmetic operations are available for numbers (but `x-y` and `x - y` are different).

More complex expressions are location paths and boolean expressions (e.g. using `<`,`>`, `!=`, `=` and logical connectives `and`, `or`).

Implicit coercion works from strings to numbers and from numbers to booleans as needed.

# Some Core Functions

Node-set example:

- `number position()` returns the context position from the expression evaluation context.

String example:

- `sting string(object?)` converts an object to a string.

Number example:

- `number number(object?)` converts an object to a number.
- `number sum(node-set)` returns the sum of the values obtained by converting each element in the node set into a number.

# XSL Transformations (XSLT)

The primary purpose of the XSLT language is to enable the transformation of XML documents into other XML documents or even into non-XML text data (e.g. HTML documents).

The components of the transformation process:

- The source XML document.
- The XSLT stylesheet (an XML doc. adhering the XSLT grammar).
- The resulting document, generated in the transformation process.

The XSLT document contains a collection of templates that prescribe patterns and transformation rules for the elements of an XML document that match the pattern.

# XSLT Stylesheets

The data model of XML documents used in XSLT is the same as the one in XPath with whitespace nodes eliminated.

An XSLT stylesheet is represented by an `xsl:stylesheet` element in the containing XML document. The `xsl` prefix must be bound to the `http://www.w3.org/1999/XSL/Transform` namespace.

The `xsl:stylesheet` element can contain the elements:

- `xsl:include` to include other stylesheets,
- `xsl:output` to specify the output format,
- `xsl:variable` to bind a value to a variable name,
- `xsl:template` to define a template of the stylesheet.

# Example

The source XML document:

```
<?xml version="1.0"?>
<info>
  <name>Josef Keller</name>
  <email>jkeller@jku.at</email>
</info>
```

We want to get the result:

```
<html><body>
  <p>Name: Josef Keller<br/>
  Email: jkeller@jku.at</p>
</body></html>
```

# Example

The XSLT stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <p>Name: <xsl:value-of select="./info/name/text()"/><br/>
      Email: <xsl:value-of select="./info/email/text()"/></p>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

# Templates

A general template definition looks as follows.

```
<xsl:template match="pattern" name="name"
              priority="priority">
  <!--Transformation rules come here.-->
</xsl:template>
```

The `pattern` is a location path, or alternatives of such, separated by '|' characters.

The `name` attribute is optional and allows us to define a named template.

The `priority` attribute is also optional and assigns a value to the template which is used in conflict resolution.

# Template Application

An XSLT processor starts by looking for a matching template for the root node of the source document.

Whenever a matching template is found, its transformation is executed (a fragment of the output document might get created).

```
<xsl:value-of select="XPath-expression"/>
```

The transformation parts may call for applications of further templates on other nodes.

```
<xsl:apply-templates select="location-path"/>
```

# Default Templates

For elements and the root node:

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

For text nodes and attributes:

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

For processing instructions and comments:

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

# Example: Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="folder">
    <html><body> <h1>Emails</h1>
      <xsl:apply-templates select="email"/>
    </body></html>
  </xsl:template>
  <xsl:template match="email">
    <p>From: <xsl:value-of select="from/text()"/><br/>
    To: <xsl:value-of select="to/text()"/><br/>
    Date: <xsl:value-of select="@date"/><br/>
    Subject: <xsl:value-of select="subject/text()"/><br/><br/>
    <pre><xsl:apply-templates select="text()"></pre></p>
  </xsl:template>
</xsl:stylesheet>
```

# Example: Result

```
<html><body> <h1>Emails</h1>
  <p>From: robert@company.com<br/>
  To: oliver@company.com<br/>
  Date: 20 Aug 2003<br/>
  Subject: Meeting<br/><br/><pre>
  Could we meet this week to discuss the interface problem
  in the NTL project? --Rob</pre></p>
  <p>From: oliver@company.com<br/>
  To: rob@company.com<br/>
  Date: 21 Aug 2003<br/>
  Subject: Re: Meeting<br/><br/><pre>
  On 20 Aug 2003 rob@company.com wrote
  &gt; Could we meet today to discuss the interface problem
  &gt; in the NTL project? --Rob
  OK. What about today at 16:00?</pre></p>
</body></html>
```

# Additional Features: Attributes

Defining attributes in the result document dynamically:

```
<xsl:template match="message">
  <font>
    <xsl:attribute name="color">
      <xsl:value-of select="@color"/>
    </xsl:attribute>
    <xsl:value-of select="text()"/>
  </font>
</xsl:template>
```

Or by using { }.

```
<xsl:template match="message">
  <font color="{@color}">
    <xsl:value-of select="text()"/>
  </font>
</xsl:template>
```

# Additional Features: Conditional Processing

Conditional processing with `xsl:if`.

```
<catalog>
  <book isbn="0-321-19784-4">
    <author>C. J. Date</author>
    <title>An Introduction to Database Systems</title>
  </book>
</catalog>
```

Template:

```
<xsl:template match="book">
  <tr> <xsl:if test="position() mod 2 = 0">
         <xsl:attribute name="bgcolor">grey</xsl:attribute>
       </xsl:if>
    <td><xsl:value-of select="author/text()"/></td>
    <td><xsl:value-of select="title/text()"/></td>
    <td><xsl:value-of select="@isbn"/></td></tr>
</xsl:template>
```

# Additional Features: Iterative Processing

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="catalog">
    <html><body>
      <h1>Catalog</h1><table>
      <xsl:for-each select="book">
        <tr><xsl:if test="position() mod 2 = 0">
              <xsl:attribute name="bgcolor">grey</xsl:attribute>
            </xsl:if>
        <td><xsl:value-of select="author/text()"/></td>
        <td><xsl:value-of select="title/text()"/></td>
        <td><xsl:value-of select="@isbn"/></td></tr>
      </xsl:for-each>
    </table></body></html>
  </xsl:template>
</xsl:stylesheet>
```

# Additional Features: Sorting

The corresponding extended piece of the last example:

```
<h1>Catalog</h1><table>
<xsl:for-each select="book">

  <xsl:sort select="title" order="ascending"/>
  <xsl:sort select="@isbn" order="ascending"/>

  <tr><xsl:if test="position() mod 2 = 0">
      <xsl:attribute name="bgcolor">grey</xsl:attribute>
      </xsl:if>
  <td><xsl:value-of select="author/text()"/></td>
  <td><xsl:value-of select="title/text()"/></td>
  <td><xsl:value-of select="@isbn"/></td></tr>
</xsl:for-each></table>
```

# Summary

- XPath
  - ⋆ Data model, String values
  - ⋆ Contexts, Location paths
  - ⋆ Other XPath expressions, Core functions
- XSLT
  - ⋆ General structure of XSLT stylesheets
  - ⋆ Templates: structure, application, defaults
  - ⋆ Generating attributes dynamically
  - ⋆ Conditional- and Iterative processing, Sorting