XSLT

J. Schneeberger

Hochschule Deggendorf

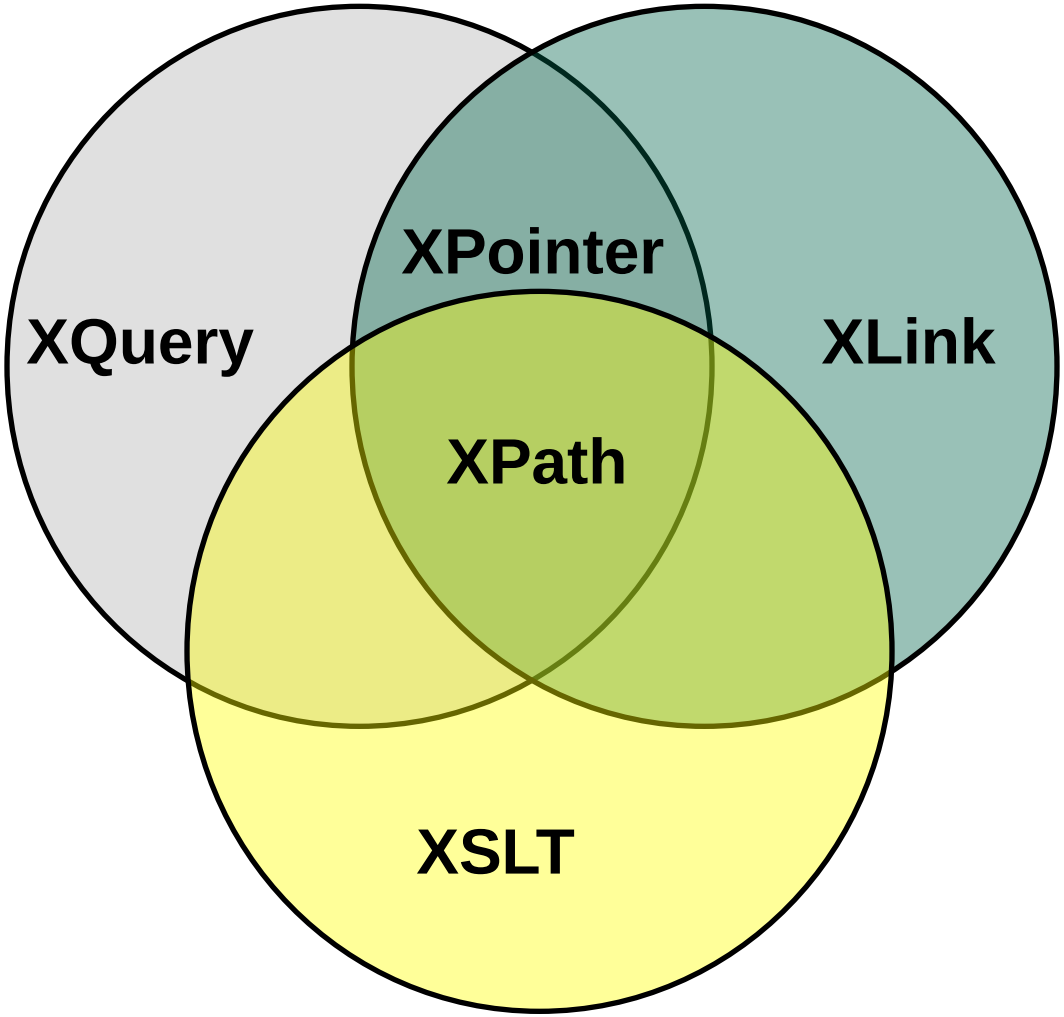
josef.schneeberger@fh-deggendorf.de

Source

Partially from:

- Anders Møller, Michael I. Schwartzbach
  “An Introduction to XML and Web Technologies”
  Addison-Weseley, January 2006
- http://www.brics.dk/ixwt/
- M. Kay “XSLT 2.0 Programmer’s Reference”
  WROX, 2004 (~ 1000 pages worth the price)

XPath, XLink, XPointer und XSLT



XSLT

- Extensible stylesheet Language (XSL)
- A language to transform XML documents
  - Transformation (XSLT)
    - to transform XML into something else
      - Text, XML, HTML, etc…
  - Formatting Objects (XSL:FO)
    - to transform XML into a readable document
      - eg. with FOP to PDF
  - Standard by the W3C

The recipe Example

Overviev

- An example
- Templates and patterns
- Output generation
- practical XSLT

HTML and XML (1)

Spaghetti Carbonara

Zutaten:

250 g Spaghetti
10 g Butter
100 g Bacon
3 Eier
30 g Parmesan
Knoblauch
Pfeffer aus der Mühle
Salz

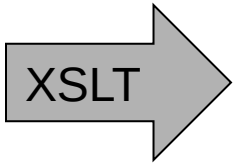
3 Portionen - 625 Kcal pro Portion
*** - 20 min Zub. - 20 min Ges.*

Spaghetti Carbonara ist die klassische Zubereitung von Spaghetti mit frischen rohen Eiern und bedeutet soviel wie Spaghetti nach Köhler Art. Spaghetti Carbonara zählt zu den bekanntesten und beliebtesten Spaghettigerichten. Wichtig ist, dass man ganz frische Eier für Spaghetti Carbonara verwendet, da die Eier nur ganz leicht stocken dürfen - anders als Rührei - und fast roh gegessen werden. Spaghetti in reichlich Salzwasser bissfest kochen. Inzwischen Butter in einer Pfanne auslassen, aber nicht braun werden lassen. Bacon oder Bauchspeck in feine Streifen schneiden und in der heißen Butter knusprig ausbraten. Eine größere Zehe Knoblauch pellen und im Ganzen zum Bacon in die Pfanne geben. Wenn die Knoblauchzehe leicht zu bräunen

[http://www.marions-kochbuch.de/rezept/0300.htm]

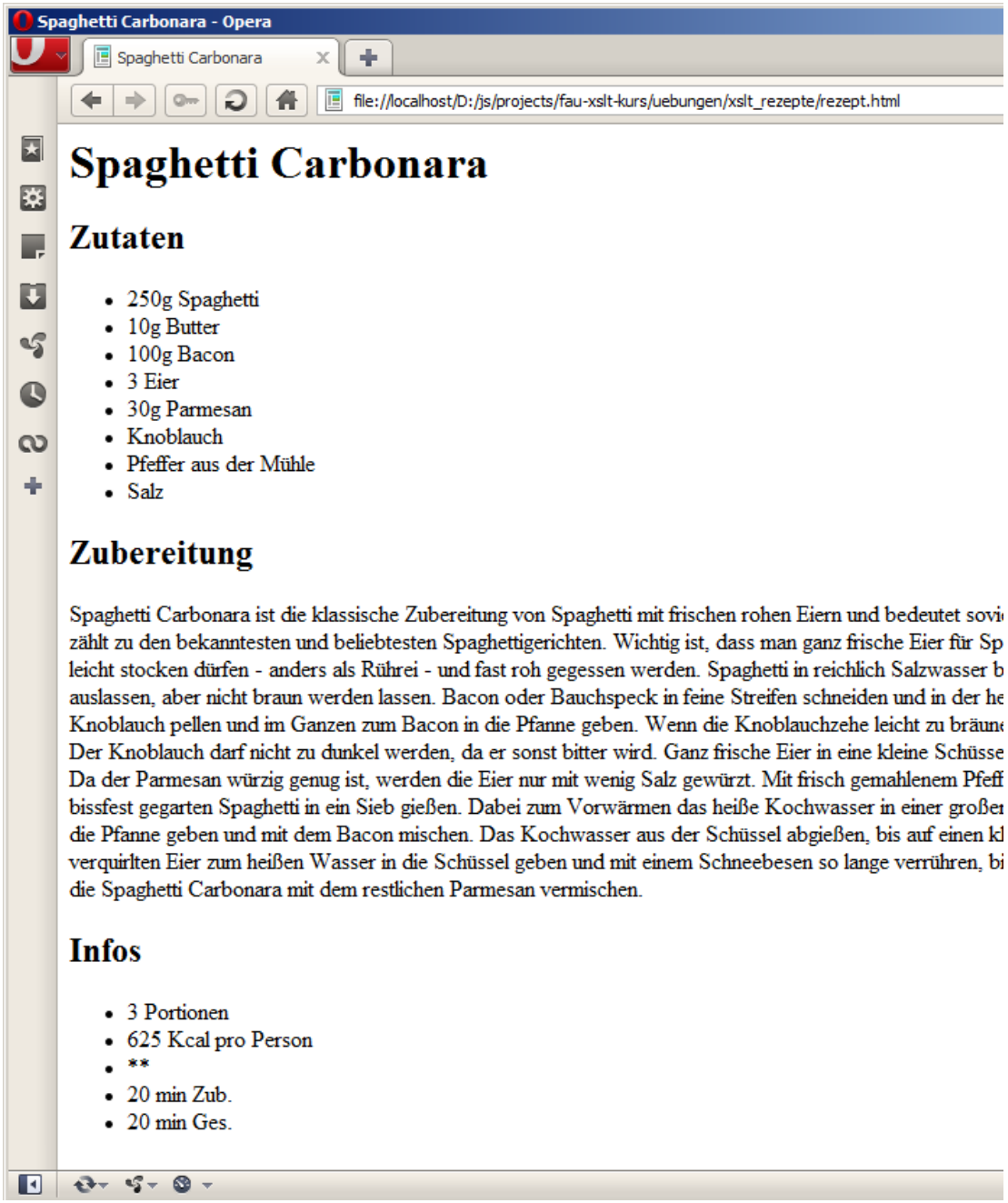
HTML and XML (2)

```
<recipe>
<titel>Spaghetti Carbonara
<incredients>
  <item weight="250g">spag
  <item weight="10g">butte
  <item amount="3">egg</it
  <item>garlic</item>
</incredients>
<preparation>
Spaghetti carbonara is the
<preparation/>
<info>
  <difficulty>2</difficult
  <duration min="20" work=
  <duration min="20" work=
</info>
</recipe>
```

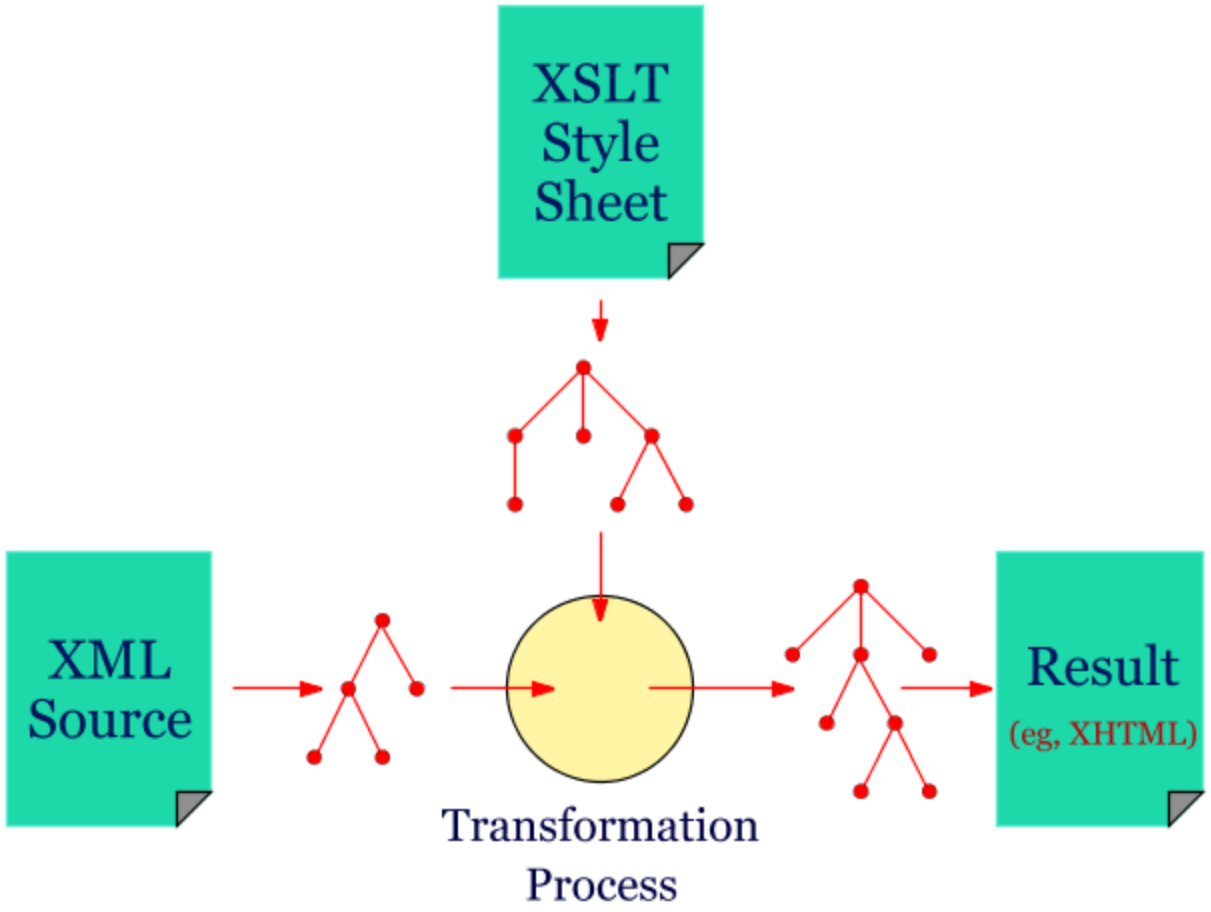



```
<html>
<head><title>Spaghetti Carbo
<body>
<h1>Spaghetti Carbonara</h1>
<h2>Incredients</h2>
<ul>
  <li>250g spaghetti
  <li>10g butter</ul>
<h2>Preparation</h2>
<p>Spaghetti Carbonara is a
<h2>Additional information</
<ul>
  <li>**
  <li>20 min prep.
  <li>20 min in total
</ul>
</body>
</html>
```

HTML and XML (3)



XSLT Transformation



[http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH/#(9)]

An XSL stylesheet

- An empty XSLT document (root element **`xsl:stylesheet`**)

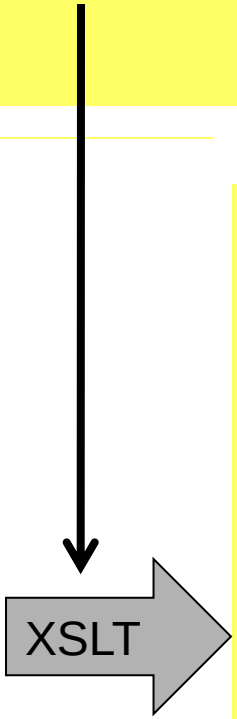
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
</xsl:stylesheet>
```

- ... the most simplest stylesheet
- ... copies the input to the output – without tags

XML stylesheet (1)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
</xsl:stylesheet>
```

```
<recipe>
<titel>Spaghetti Carbonara</tit
<incredients>
  <item weight="250g">spaghett
  <item weight="10g">butter</it
  <item amount="3">egg</item>
  <item>garlic</item>
</incredients>
<preparation>
Spaghetti carbonara is the clas
<preparation/>
<info>
  <difficulty>2</difficulty>
  <duration min="20" work="prep
  <duration min="20" work="tota
</info>
</recipe>
```



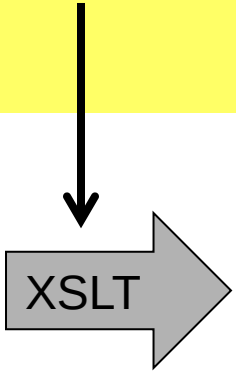

```
<?xml version="1.0"
        encoding="utf-8"?>
Spaghetti Carbonara

  Spaghetti
  butter
  egg
  garlic

Spaghetti Carbonara is the
classical ...
```

XML stylesheet (2)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:template match="/">
        <html>
            <xsl:value-of select="rezept"/>
        </html>
    </xsl:template>
</xsl:stylesheet>
```




```
<recipe>
<titel>Spaghetti Carbonara</tite
<incredients>
  <item weight="250g">spaghetti<
  <item weight="10g">butter</ite
  <item amount="3">egg</item>
  <item>garlic</item>
</incredients>
<preparation>
Spaghetti carbonara is the class
```

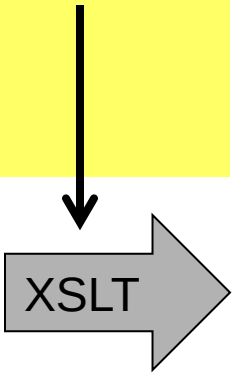
```
<?xml version="1.0"
        encoding="utf-8"?>
<html>
Spaghetti Carbonara

  Spaghetti
  butter
  egg
  garlic

Spaghetti Carbonara is the
classical ...
```

XML stylesheet (3)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:template match="/">
        <html>
            <xsl:apply-templates select="*"/>
        </html>
    </xsl:template>
    <xsl:template match="recipe">
        <head><title><xsl:value-of select="title"/></title></head>
        <body>
            <h1><xsl:value-of select="title"/></h1>
            ...
        </body>
    </xsl:template>
</xsl:stylesheet>
```




```
<html>
   <head>
      <meta http-equiv="Content-..

      <title>Spaghetti Carbonara</title>
   </head>
   <body>
      <h1>Spaghetti Carbonara</h1>
        Spaghetti
        butter
```

XML stylesheet (4)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:template match="/">
        <html>
            <xsl:apply-templates select="*"/>
        </html>
    </xsl:template>
    <xsl:template match="recipe">
        <head><title><xsl:value-of select="title"/></title></head>
        <body>
            <h1><xsl:value-of select="title"/></h1>
            <xsl:apply-templates select="*"/>
        </body>
    </xsl:template>
    <xsl:template match="ingredients">
        <ul>
            <xsl:for-each select="item">
                <li><xsl:value-of select="."/></li>
            </xsl:for-each>
        </ul>
    </xsl:template>
    <xsl:template match="preparation">
        <p><xsl:value-of select="."/></p>
    </xsl:template>
    <xsl:template match="info">
        ...
    </xsl:template>
</xsl:stylesheet>
```

stylesheet (4) – result

```
<html>
   <head>
     <title>Spaghet
   </head>
   <body>
      <h1>Spaghetti  bonara</h1>
      <ul>
         <li>Spaghetti</li>
          <li>butter</li>
          <li>egg</li>
          <li>garlic</li>
      </ul>
      <p>
         Spaghetti Carbonara is...
      </p>
      <ul></ul>
   </body>
</html>
```

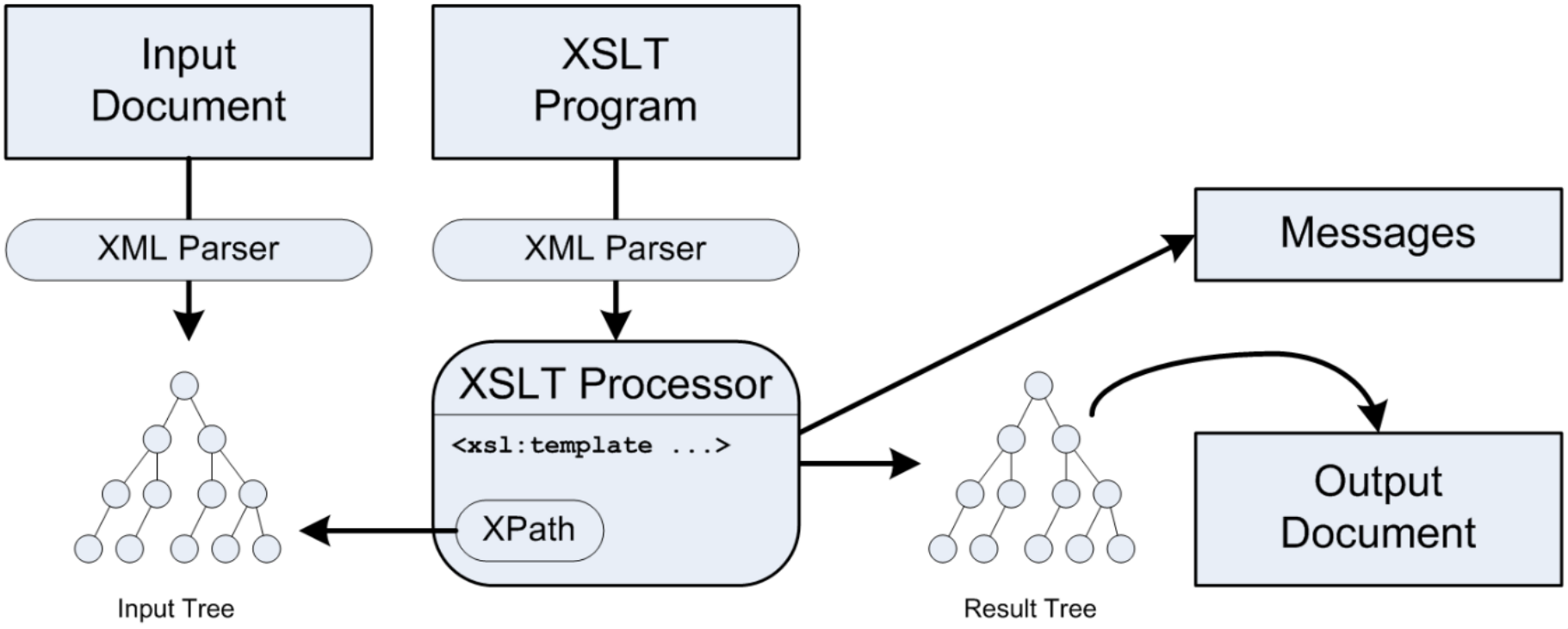
```
<item weight="250g">Spaghetti</item>
<item weight="10g">butter</item>
<item amount="3">egg</item>
```

```
<serves>3</serves>
<kcal>625</kcal>
<difficulty>2</difficulty>
<duration min="20" work="preparation"/>
```

still missing ...

Language Elements

XSLT Transformation



XSLT

- A programming language for trees
- Feeling / Important constructs:
  - traversal of trees
  - recursive calls
  - rules for subtrees

Procedural Semantics

- Assign the document root to the context
  - first template: <xsl:template match="/">
- ... for all nodes in the context
  - select the next node from the context
  - search for those templates, that match the node
- ... if one or more templates match:
  - select one (eg. the most specific) and process the node
- ... continue recursively

XSLT stylesheet

- An XSLT stylesheet consists of template rules

```
<xsl:template match="...">
    ...
</xsl:template>
```

- find all template rules that match the context
- select the most specific rule
- evaluate the body of the template – constructs a sequence

XPath in XSLT

- is used for the patterns within the templates
- selects nodes for subsequent processing
- calculates Boolean conditions (predicates)
- generates text content for the output document

Evaluation Context

- A context item
  - a node in the source tree
  - or an atomic value
- A context position and size
- ~~A~~ A set of variable bindings
  - mapping variable names to values
- ~~A~~ A function library (including those from XPath)
- A set of namespace declarations

The Initiale Context

- The context item is the document root
- The context position and size both have value 1
- The set of variable bindings contains only global parameters
- The function library is the default one
- The namespace declarations are those defined in the root element of the stylesheet

Patterns and Matching

- A *pattern* is a restricted XPath expression
  - it is a union of path expressions
  - each path expression contains a number of steps separated by / or //
  - each step may only use the **`child`** or **`attribute`** axis
- *f*A pattern *matches* a node if
  - starting from *some* node in the tree:
  - the given node is *contained* in the resulting sequence

Attributes of Templates

- **`@name`**: used to call templates like a function by its name.
- **`@mode`**: used to restrict the candidate templates
  - the mode attribute separates the templates into selectabl classes
- **`@priority`**: used to determine specificity

Generating Output

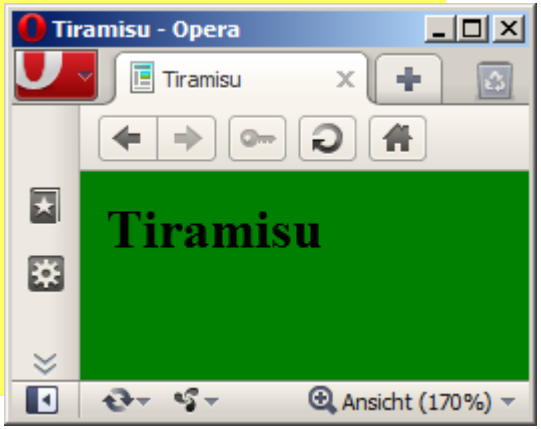
Sequence Constructors

- **Text constructors**
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Literate Text

```
<xsl:stylesheet  version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3. org/1999/xhtml">

   <xsl:template  match="/">
      <html>
         <head>
            <title>Tiramisu</title>
         </head>
         <body bgcolor="green">
            <b>Tiramisu</b>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```



Text Constructors

- by specification in the template

```
lore ipsum
```

- using xsl:text – may be used to generate whitespace

```
<xsl:text>  lore ipsum</xsl:text>
```

- The (atomized) value of an XPath expression:

```
<xsl:value-of select=".//@gewicht"/>
```

Text Constructors

- **`xsl:value-of`** outputs the value of an XPath expression (transformed into a string)
- Attributes
  - **`select`**: the XPath expression
  - **`disable-output-escaping`** (optional): specifies the output of special characters (‘>‘ or >)

Sequence Constructors

- Text constructors
- **Element- and attribute constructors**
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Element Constructor

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/1999/xhtml">

   <xsl:template match="/">
      <xsl:element name="html">
         <xsl:element name="head">
            <xsl:element name="title">Tiramisu</xsl:element>
         </xsl:element>
         <xsl:element name="body">
            <xsl:attribute name="bgcolor" select="'green'"/>
            <xsl:element name="b">Tiramisu</xsl:element>
         </xsl:element>
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Attribute Constructor

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/1999/xhtml">

   <xsl:template match="/">
      <xsl:element name="html">
         <xsl:element name="head">
            <xsl:element name="title">Tiramisu</xsl:element>
         </xsl:element>
         <xsl:element name="body">
            <xsl:attribute name="bgcolor" select="'green'"/>
            <xsl:element name="b">Tiramisu</xsl:element>
         </xsl:element>
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Attribute Constructor

```
<xsl:stylesheet version="2.0" ... >

   <xsl:template match="/">
      <xsl:element name="html">
         <xsl:element name="head">
            <xsl:element name="title">Tiramisu</xsl:element>
         </xsl:element>
         <xsl:element name="body">
            <xsl:attribute name="bgcolor" select="//@bgcolor"/>
            <xsl:element name="b">Tiramisu</xsl:element>
         </xsl:element>
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

or

```
<xsl:attribute name="bgcolor">
   green
</xsl:attribute>
```

Attribute Constructor

```
<xsl:stylesheet  version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3. org/1999/xhtml">

   <xsl:template  match="/">
      <html>
         <head>
            <title>Tiramisu</title>
         </head>
         <body bgcolor="{//@bgcolor}">
            <b>Tiramisu</b>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

More XSLT Elements

- <xsl:text>: writes Text to the output
- <xsl:comment>: generates a comment
- <xsl:include>: includes external XSLT tempates
- <xsl:import>: like xsl:include – but higher priority

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- **Recursion**
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Recursion

- **`apply-templates`**

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

  - matches nodes by an XPath expression (select)
  - applies the stylesheet to the node
  - and concatenates the results.
- If no **`select`** is specified: `child::node()`
- Processing is often a simple recursive traversal of the input tree.

The Builtin Rule

- An XSL stylesheet contains a default rule:

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

- This rule generates as default output the text content of the context node.
- To avoid this rule, it has to be overridden:

```
<xsl:template match="text()|@*"/>
```

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- **Modes**
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Modes

```
<?xml version="1.0" encoding="UTF-8"?>
<recipes>
	<recipe>
		<title>Spaghetti Carbonara</title>
		<ingredients>
			<item weight="250g">Spaghetti</item>
			...
		</info>
	</recipe>
	<recipe>
		<title>Spaghetti Bolognese</title>
		<ingredients>
		...
		</info>
	</recipe>
</recipes>
```

Modes

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

  <xsl:template match="recipes">
    <head><title>recipes</title></head>
    <body>
      <h1>Overview</h1>
        <ul>
          <xsl:apply-templates mode="overview" select="recipe"/>
        </ul>
      <h1>Preparation</h1>
        <xsl:apply-templates mode="preparation" select="recipe"/>
    </body>
  </xsl:template>
```

Modes

```
  <xsl:template match="recipe" mode="overview">
    <li>
      <xsl:value-of select="titel"/>
      <xsl:text>: Difficulty</xsl:text>
      <xsl:value-of select="info/difficulty"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="infos/duration[@work=,total']/@min"/>
      <xsl:text> minutes duration</xsl:text>
    </li>
  </xsl:template>

  <xsl:template match="recipe" mode="preparation">
    <h2><xsl:value-of select="title"/></h2>
    <p><xsl:value-of select="preparation"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Modes


Rezepte - Opera
Rezepte
file://localhost/D:/js/projects/fau-xslt-kurs/eclipse_ws/Rezepte/js/r
Rezepte Übersicht
Spaghetti Carbonara: Schwierigkeit 2, 20 Min. Zubereitungszeit
Spaghetti Bolognese: Schwierigkeit 3, 60 Min. Zubereitungszeit
Zubereitung
Spaghetti Carbonara
Spaghetti Carbonara ist die klassische Zubereitung von Spaghetti mit frischen rohen Eie
Eier nur ganz leicht stocken dürfen - anders als Rührei - und fast roh gegessen werden.
Eine größere Zehe Knoblauch pellen und im Ganzen zum Bacon in die Pfanne geben. V
g vom Parmesan verquirlen. Da der Parmesan würzig genug ist, werden die Eier nur mi
auffangen. Die gut abgetropften Spaghetti in die Pfanne geben und mit dem Bacon misc
bis die Sauce dicklich wird. Spaghetti untermischen und die Spaghetti Carbonara mit d
Spaghetti Bolognese
Spaghetti Bolognese schmeckt umso aromatischer, je länger die Sauce köchelt. Man s
auf niedrigster Stufe langsam einkochen lässt, bis sie sämig wird. Am besten eignet sich
die Temperatur auf mittlere Stufe zurückschalten und Butter in der Pfanne auslassen. S
einen hohen Kochtopf umfüllen. In der Bratpfanne Olivenöl erhitzen und Hack darin kr

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- **Conditionals**
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Conditionals

- if

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

- switch – corresponds to the if/else construct of programming languages:

```
<xsl:choose>
  <xsl:when test='expression'> ... </xsl:when>
  <xsl:when ..
    ...
  <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
```

Conditionals

- The attribute **`test`** evaluates to a boolean value
- **`<xsl:if test="count(chapter)>=2">`**
- **`<xsl:if test="$x">`**
  - If x is a string: test=true if the length of x > 0
  - If x is a nodeset: test=true if x contains at least one node
  - If x is a number: test=true, if x != 0
  - If x is a boolean: test uses this value

Conditionals

Are these expressions true or false?

- **<xsl:if test="true()">**
- **<xsl:if test="true">**
- **<xsl:if test="'false'">**
- **<xsl:if test="not(3)">**
- **<xsl:if test="section/section">**

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- **Repetitions**
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Repetitions

- for statement

```
<xsl:for-each select='expression'>
  <xsl:sort select='expression'/>
  …
</xsl:for-each>
```

Repetitions

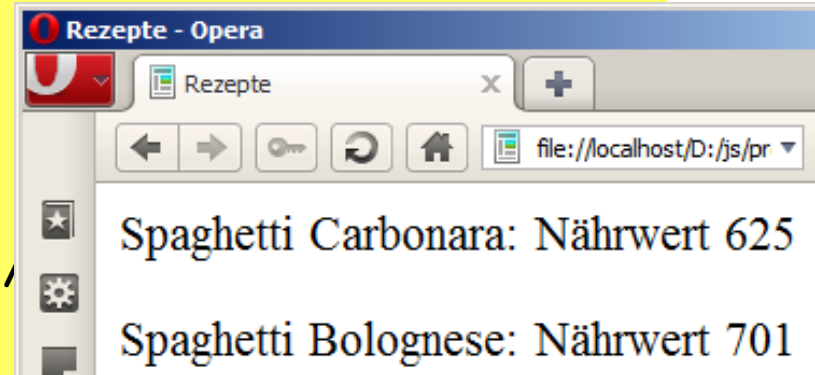
- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- **Copying and sorting nodes**
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Sorting

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet ...>
  <xsl:template match="/">
    <html><xsl:apply-templates/></html>
  </xsl:template>

  <xsl:template match="receips">
    <head><title>recipes</title></head>
    <body>
      <xsl:apply-templates select="recipe">
        <xsl:sort select="info/kcal" data-type="number"
                                        order="ascending"/>
      </xsl:apply-templates>
    </body>
  </xsl:template>

  <xsl:template match="recipe">
    <p><xsl:value-of select="title"/
...
```



Copying Nodes

There are two instructions to copy nodes:

- **`copy-of`**: produces a “deep” copy
  - the element and all its subelements are copied
- **`copy`**: produces a “shallow” copy

An example:

```
<xsl:template  match="ol|ul">
  <xsl:copy>
    <xsl:attribute name="style"
                   select="'list-style-type:  square;'"/>
    <xsl:copy-of select="*"/>
  </xsl:copy>
</xsl:template>
```

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- **Template invocation and named templates**
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Named Templates

- Templates can be called by their names
- Parameters are supplied by **<xsl:with-param>**

```
<xsl:call-template name ="draw-box">
  <xsl:with-param name="startX" select="50"/>
  <xsl:with-param name="startY" select="50"/>
  <xsl:with-param name="endX" select="100"/>
  <xsl:with-param name="endY" select="100"/>
</xsl:template>
```

```
<xsl:template name ="draw-box">
       <xsl:param name="startX"/>
       <xsl:param name="startY" select="'0'"/>
       <xsl:param name="endX">10</xsl:param>
       <xsl:param name="endY"/>
... $startX ... $startY ...
```

Passing Template Parameters

- **<xsl:param>** have to be the first elements of a template.
- Parameters are passed by **<xsl:with-param>**
- The default value of a parameter ist the empty string (see also **xsl:variable**)
- Names associate parameters in template definitions and calls.
- Parameters may be omitted in template calls.

Parameter Example

```
<xsl:template match="/">
  <table>
  <xsl:for-each select="//number">
  <tr><th>
    <xsl:choose>
      <xsl:when test='text() mod 2'>
        <xsl:apply-templates select=".">
          <xsl:with-param name="type">odd</xsl:with-param>
        </xsl:apply-templates>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </th></tr>
  </xsl:for-each>
  </table>
</xsl:template>
```

```
  <xsl:template match="number">
    <xsl:param name="type">even</xsl:para
    <xsl:value-of select="."/>
    <xsl:text> (</xsl:text>
    <xsl:value-of select='$type'/>
    <xsl:text> )</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- **XSLT functions**
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

XPath Functions

- Functions for nodesets
  - position(), last(), local-name(), etc…
- Functions for strings
  - string(), contains(), substring(), etc…
- Boolean Functions
  - boolean(), not(), etc…
- Functions for numbers
  - number(), sum(), round(), etc…

XSLT Functions

- **`current()`**: returns a nodeset consisting of the current node
- **`document()`**: includes external files
- **`format-number()`**: formats numbers
- **`generate-id()`**: generates an unique ID
- **`key()`**: refers to a <xsl:key> ID
- **`element-available()`**: checks if the XSLT processor provides a specific element
- **`function-available()`**: checks if the XSLT processor provides a specific function

Example for "document"

- Multiple input documents
- **<xsl:apply-templates select=document('external/external.xml')/order"/>**

```
<report>

 <title>Bestellungen</title>

 <po filename="po1.xml">

 <po filename="po2.xml">

 <po filename="po2.xml">

</report>
```

```
<xsl:template match ="/">

 <xsl:for-each select=/report/po"/>

 <xsl:apply-templates
 select="document(@filename)"/>

 <xsl:for-each/>

</xsl:template>
```

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- **Text handling and regular expressions**
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- User defined functions

Handling unstructured text

- unparsed-text() function
  - reads a text file into a string
- tokenize() function
  - splits a string into substrings
- xsl:analyze-string
  - parses a string and generates markup

Regular expression functions

- matches()
  - test if a string matches a regex
  - **`if (matches($in, '[A-Z]{3}[0-9]{3}')`**
- tokenize()
  - split a string into substrings
  - regex matches the separator
  - **`for $s in tokenize($in, ',\s?') ...`**
- replace()
  - replace every occurrence of a match
  - **`replace($in, '\s', '%20')`**

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- **xsl:variable, xsl:number**
- Grouping (and xsl:key)
- User defined functions

xsl:variable

- Defines a variable
- The variable may be accessed by $variablename
- If <xsl:variable> is a toplevel element, it defines a global variable.
- The values of variables may not be modified
- Local variables are only valid within the element that contains the declaration

```
<xsl:variable name="inRow" select='4'/>
<xsl:variable name="color">red</xsl:variable>
<xsl:variable name="totalChapters"> <xsl:value-of select="count(//chapter)"/> </xsl:variab
```

xsl:number

- Associates nodes to sequences of Numbers or other ordered enumerations.
- Is used to provide a numbering for chapters in a document.
- Attributes:
  - level (the method of counting)
  - format (the format of the displayed numbers or characters)
  - count (determines which nodes to count)
- Example: http://www.xm.co.nz/ShoXS/xslnumber.htm

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- **Grouping (and xsl:key)**
- User defined functions

xsl:key

- XSLT keys are an efficient method to extract values from a tree with the help of other values.
- Defines an index for the current document
- xsl:key works in combination with the functions key() and generate-id().

Syntax

xsl:key

Attri-but	Description
name	The name of the index
match	An xPath expression, that identifies the nodes contained in the index.
use	That part of the match node which identifies the node within the index. This parameter is used to access the node of the document vie the index.

The function key()

Para-meter	Description
name	The name of the index
index	The key that identifies a node in the index

generate-id() associates any node in the document with an unique id.

Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<recipies>
	<recipe>
		<titel>Spaghetti Carbonara</titel>
		<incredients>
			<item weight="250g">Spaghetti</item>
			...
		</infos>
	</recipe>
	<recipe>
		<titel>Spaghetti Bolognese</titel>
		<incredients>
		...
		</infos>
	</recipe>
</recipies>
```

Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" indent="yes"/>
 <xsl:key name="keyRecipies" match=„recipe" use="titel"/>

 <xsl:template match="/">
  <Match>
   <xsl:value-of
    select="key('keyRecipies','Spaghetti Carbonara')/incredients"/>
  </Match>
 </xsl:template>

</xsl:stylesheet>
```

Explanation

xsl:kex defines an index of the following form:

Index (unique)	Node
`Spaghetti Carbonara`	The **`recipe`** node with the title **`Spaghetti Carbonara`**
`Spaghetti Bolognese`	The **`recipe`** node with the title **`Spaghetti Bolognese`**

The key() function may be used to access the index.

Grouping

- Takes any sequence as input
- Divides the items into groups
- Applies processing to each group

group-by:
items with a common value for a grouping key
group-adjacent:
adjacent items with a common grouping key
group-starting-with:
pattern to match first item in each group
group-ending-with:
pattern to match last item in each group

Grouping by Value

```
<xsl:for-each-group select="book"
                                    group-by="publisher">
   <xsl:sort select="current-grouping-key()"/>
   <h2>Publisher:
          <xsl:value-of select="current-grouping-key"/>
   </h2>
   <xsl:for-each select="current-group()"/>
      <xsl:sort select="title"/>
      <p>author: <xsl:value-of select="author"/></p>
      <p>title: <xsl:value-of select="title"/></p>
   </xsl:for-each>
</xsl:for-each-group>
```

xsl:key remarks

- Attributes may be used as keys as well as other nodes.
- Keys may be used efficiently if the match and use parameters are selected carefully.
- Keys may also be used for efficiency reasons.

Sequence Constructors

- Text constructors
- Element- and attribute constructors
- Recursion
- Modes
- Conditionals
- Repetitions
- Copying and sorting nodes
- Template invocation
- XSLT functions
- Text handling and regular expressions
- xsl:variable, xsl:number
- Grouping (and xsl:key)
- **User defined functions**

User-defined Functions

- Written like named templates
- Called from XPath
- Return a result

```
<xsl:function name="ged:date-to-ISO" as="xs:date">
  <xsl:param name="in" as="ged:date"/>
  <xsl:sequence select="xs:date(concat(
     substring($in, 8, 4), '-'
     format-number(index-of(("JAN", "FEB", ...),
                            substring($in, 4, 3)), '00'),
     '-', substring($in, 1, 2)))"/>
</xsl:function>

<xsl:sort select="ged:date-to-ISO(@birth-date)"/>
```

XSLT 2.0

Michael Kay, Prague 2005

What's new in XSLT 2.0

- Major Features
  - grouping
  - regular expressions
  - functions
  - schema support
- Many "minor" features

Some “minor” features

XSLT 2.0

- Temporary trees
- Multiple Output Files
- Format date/time
- Tunnel parameters
- Declared variable types
- Multi-mode templates
- xsl:next-match
- conditional compilation
- XHTML serialization
- xsl:namespace
- separator=“,”
- character maps

XPath 2.0

- Sequences
- if..then..else
- for $x in X return f($x)
- some/every
- except/intersect
- $n is $m

Function library

- String functions
- Regex functions
- Date/time arithmetic
- URI handling
- min(), max(), avg()

Hands-on Section

Recipes

- Convert the recipes to HTML to view it in a browser
- Use modes to generate a table of contents for the recipes
- Use grouping to generate an index of the recipes by ingredient

Resources and Links

Online

- XSLT Online Tutorial von ZVON (Gruppe aus Tschechien)
  http://www.zvon.org/xxl/XSLTutorial/Output/index.html
- XSLT Online Tutorial von W3Schools (eine norwegische Firma: Refsnes Data)
  http://www.w3schools.com/xsl/default.asp
- XSLT Spezifikation des WWW Consortiums
  http://www.w3.org/TR/xslt
- XSLT Präsentation des W3C
  http://www.w3.org/Consortium/Offices/Presentations/XSLT_XPATH/#(1)
- XSLT Unterlagen im XML-Kurs von Erik Wilde
  http://dret.net/lectures/xml-fall08/
- Quelle für Fragen zur XSLT-Programmierung (wie macht man was / FAQ)
  http://www.dpawson.co.uk/xsl/