Data Querying, Extraction and Integration: XPath and XSLT

Recuperación de Información
2007

Lecture 4.

Overview

- Motivation: Data Extraction & Integration using XSLT
- Xpath
- XSLT

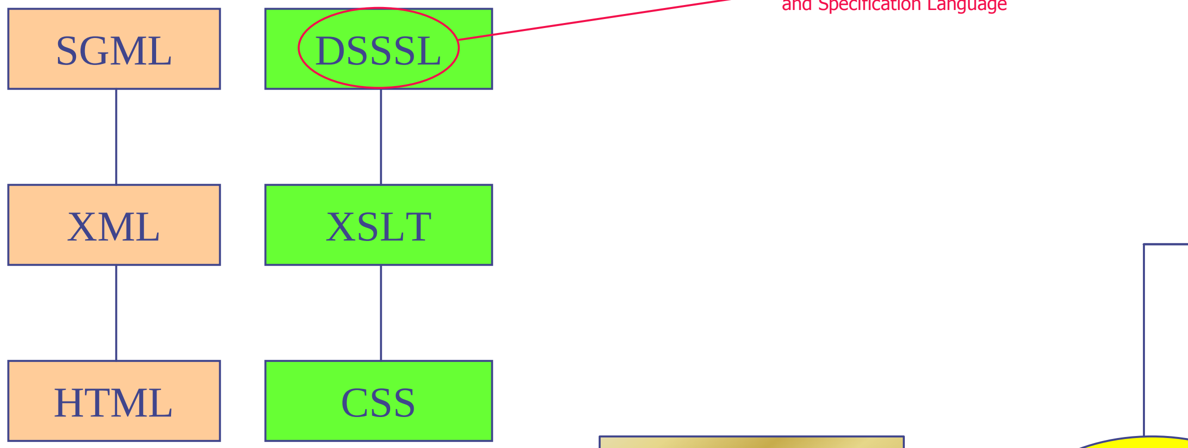
Data Extraction & Integration for XML (and for the Web):

- XPath: A Language for navigating through and querying/selecting parts of the XML tree
- XSL(T): XML Stylesheet Language, define rules for transforming a source tree into a result tree, using XPath to select the parts to transform, also allows combining XML documents

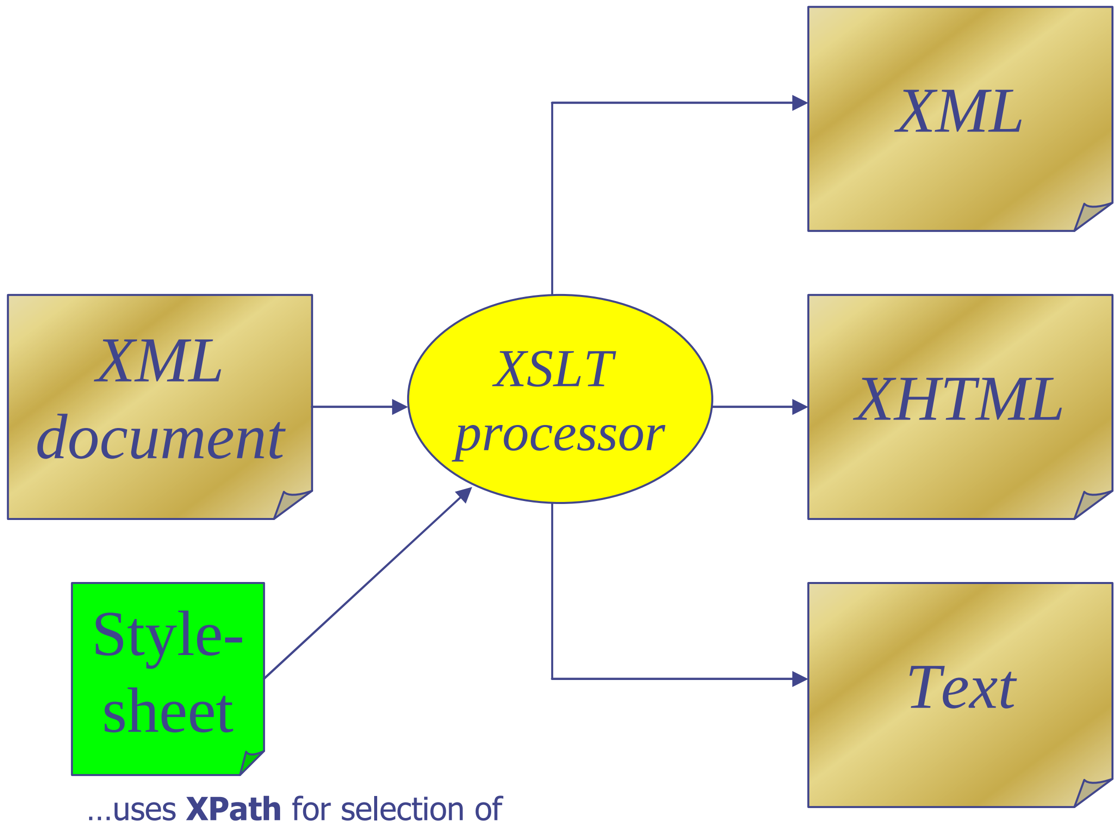
- An application domain on the Web: For the (current) Web we can use tools for generating XML out of existing HTML content to extract information→ *Wrappers*

- *More generally:* Transformations from one XML format into another (XML) format.

XML Stylesheets:



SGML
DSSSL
XML
XSLT
HTML
CSS
ISO/IEC 10179:1996 Document Style Semantics and Specification Language



…uses **XPath** for selection of parts of the original XML

The Extensible Stylesheet Language Family (XSL)

Definitions from http://www.w3.org/Style/XSL/

- XSL Transformations (XSLT)
  - a language for transforming XML
- the XML Path Language (XPath)
  - an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML Linking specification)
- XSL Formatting Objects (XSL-FO)
  - an XML vocabulary for specifying formatting semantics, rather for output formats/rendering than for XML to XML transformations.

Examples:

- In our examples, we will use XSLT mainly to extract from/generate XHTML, but there are lots of more general applications to transform from/to aribtrary (not necessarily XML) formats!

A first example:

Input:

```
<?xml version="1.0" encoding="UTF-8"?>
<lehre>
   <veranstaltung>Telecooperation</veranstaltung>
</lehre>
```

Stylesheet: stylesheets consist of templates:

```
…
<xsl:template match="/lehre/veranstaltung">
   <i><xsl:value-of select="."/></i>
</xsl:template>
…
```

XPath expressions

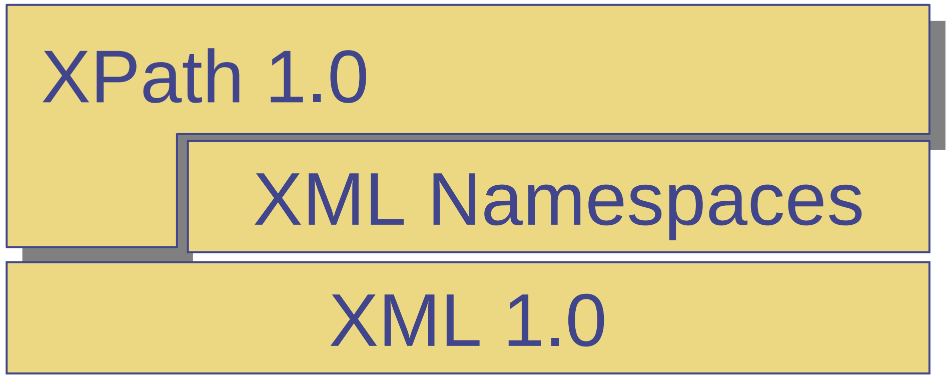
Result:

```
<i>Telecooperation</i>
```

→ Files: exa1.xml, exa1.xsl

Before we get to Stylesheets…

XPath:



XPath is a major element in the W3C XSLT standard. Without XPath knowledge you will not be able to create XSLT documents.

XPath:

- XPath is the basis for most XML query languages
  - Selection of document parts
  - Search context: ordered set of nodes
- XPath is being used extensively in XSLT
  - Disadvantage: XPath itself is not XML notation
  - Also basis of XPointer und XQuery
- Navigate through the XML Tree
  - Similar to a filesystem ("/", "../", " ./", etc.)
  - Query result is the final search context, usually a set of nodes
  - Filters can modify the search context
  - Selection of nodes by
    element names, attribut names, type, content, value, relations
  - several pre-defined functions

- Version 1.0 Recommendation, Version 2.0 Working Draft

XPath:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<lehre>
  <responsible>
    <title>Dr.</title>
    <name>Ying Ding</name>
  </responsible>
  <veranstaltung sine-tempore="no" year="2003" type="lecture">
    <title>Telecooperation</title>
    <lecturer>Dieter Fensel</lecturer>
    <exam>
      <date>2.7.</date>
      <room>HS A</room>
      <max>150</max>
    </exam>
    <exam>
      <room>SR 13</room>
      <date>2.10.</date>
      <max>20</max>
    </exam>
  </veranstaltung>
  <veranstaltung sine-tempore="no" year="2004" type="lecture">
    <title>Telecooperation</title>
    <lecturer>Axel Polleres</lecturer>
    <exam>
      <date>1.7.</date>
      <room>HS A</room>
      <max>150</max>
    </exam>
  </veranstaltung>
  <veranstaltung sine-tempore="yes" year="2004" type="seminar">
    <title>Next Web Generation</title>
    <lecturer>Dieter Fensel</lecturer>
  </veranstaltung>
</lehre>
```

Select particular parts of this document, some possible queries:

- **Give me all exams of lectures in 2003**
- **All titles**
- **All exams which allow for more than 100 participants**

etc.

All this is (and some more) is expressible in XPath!

XPath: How to reference paths

- “`/`” … document root (before doc element)
- “`/lehre`” … selects the document element.

- **Absolute Paths**:
  “`/lehre/veranstaltung/title`” … returns a set of nodes
  (all `title` nodes which are children of `veranstaltung` nodes directly below `lehre`)

- Arbitrary descendands, wildcards:

- “`/lehre//datum`” … all datum nodes at any depth below lehre
- “`/lehre/*/title`“ … all title nodes at depth 2 below lehre (any node in between).

XPath: How to reference paths

- **Relative paths:** if you do not start the path with "/", the path is viewed relative to the current search context (important for instance in stylesheets, the context where the template is applied):

`"./"`	…	refers to the current search context
`"../"`	…	refers to the parent node oft the current search context.
`title/..`	…	selects the parents of title (relative path)
`//date/../*`	…	selects all elements on the same level as date, in the document (absolute path).

Filters: use Predicates in []

Filters allow us to specify conditions under which a certain part is selected.

Examples:

- `//veranstaltung/exam[2]` … selects only the second `exam` for each `veranstaltung`

- `//veranstaltung[@sine-tempore="yes"]` …
  selects all `veranstaltung` nodes with the attribute `sine-tempore` set true.

- `//veranstaltung[@sine-tempore]` …
  selects all `veranstaltung` nodes with the attribute `sine-tempore` defined.

Can be combined with relative paths and comparison operators:

- `//veranstaltung[./exam/max>100]` …
  selects only the veranstaltung nodes which have an exam with more than 100 participants

XPath Axes

The XPath syntax for a location step is:

`axisname::nodetest[predicate]`

- We have used axes already (but hidden):

Abbr.	Meaning	Example
none	child::	`veranstaltung` is short for `child::veranstaltung`
@	attribute::	`veranstaltung[@type="lecture"]` is short for `veranstaltung ::node()[attribute::type="lecture"]`
.	self::node()	`.//exam` is short for `self::node()/descendant-or-self::node()/child::exam`
..	parent::node()	`../exam` is short for `parent::node()/child::exam`
//	/descendant-or-self::node()/	`//exam` is short for `/descendant-or-self::node()/child::exam`

Other axes:

AxisName	**Description**
ancestor	Contains all ancestors (parent, grandparent, etc.) of the current node
ancestor-or self	Like above, but also contains the current node
attribute	Contains all attributes of the current node
child	Contains children of the current node
descendant, descentant-or self	analogous ancestor, but contains all descendants instead.
following following-sibling	Everything in the document after the current node (in depth-first parsing of the tree) All siblings after the current node
preceding, preceding-sibling	analogous following, following-sibling
self	The current node
namespace	All namespace nodes of the current node

Test your examples:

→Files:

teaching.xml, teaching2.xml,
xpathtest.xsl

Node-Tests:

The XPath syntax for a location step is:

```
axisname::nodetest[predicate]
```

- Nodetests:

Function	**Description**	**Examples**
text()	returns all textnodes (could be multiple in case of mixed content)	//title/text() /descendant::text()
node()	Returns the node itself	//exam/self::node()
*	Returns all element nodes	//exam/date/* *(empty set!)* /*/*/self::node() (all nodes at depth 2)

Node Set functions (usable in filter predicates):

The XPath syntax for a location step is:

```
        axisname::nodetest[predicate]
```

- In predicates you can use relative paths and node set functions, some of which are:

position()	Position in the node set, more precisely: position wrt. to its sibling!	//@*[position()=2] //exam[position()=1] ***short**:* //exam[1]
count(nodeset)	Counts number of elements of the node set	//*[count(./child::node())>3] ***short**:* //*[count(./*)>3]
sum(noteset)	Sum of numerical value-node set	//veranstaltung[sum(.//max) > 150]
last()	Returns the position of the last element in the node set	//veranstaltung[title="Telecooperation"]/exam[position()=last()]
local-name(), name(), namespace-uri()	Returns the local part of a node name Returns the full node-name Resolves and returns the namespace-uri	//*[namespace-uri()="http://www.xyz.org"]

Some arithmetics, Boolean & String Functions:

+,-, *, div, mod	Simple Arithmetics	//veranstaltung[@year - 3 = 2000]
=, !=, <, >, <=, >=	Comparison operators, <, >, >=, <= always numberic (no lexicographic comparison)!	//*[count(./child::node())>3]
and, or, not(expression)	Returns the position of the last element in the node set	//veranstaltung[title="Telecooperation" and @year=2004]
floor(number), ceiling(number), number(value), round(number),		… some more arthimetic functions

Some String functions:

string(value)	String conversion	
normalize-space()	Like trim() in JAVA	
String-length(string)	Returns length	//veranstaltung[string-length(title) >15]
Starts-with(string, substr)	String starts with substring	//veranstaltung[starts-with(title,"Tele")]

… and many more functions, check Standard for the details!

Filter-lists: If multiple filters are specified the semantics depends on order!

//exam[max > 50][1]

choose the first among those exams which have max > 50 for each veranstaltung.*

//exam[1][max > 50]

choose the first exam for each veranstaltung*, but only if max > 50.

Remark: *The position is viewed wrt. the siblings in the tree not wrt. all elements, cf. slide 14. In order to select **the first among ALL exam nodes which have max>50**, you have to write something like:* /descendant::exam[max>50][1]

-> Let's try this out!

XPath: Summary

- Simple but powerful query & navigation language for XML trees
- Allows to almost arbitraryly select parts of the XML Tree.
- Many useful built-in functions!

Some more examples!

XML Stylesheet Language

- Stylesheet consists of a list of templates, example:

```
<xsl:template match="/lehre/veranstaltung">
  <i><xsl:value-of select="."/></i>
</xsl:template>
```

- Output not necessarily HTML or XML, could be any text!

- XSLT *Recommendation: Version 1.0, Working Draft: V2.0*

Stylesheets: Building Blocks:

- A list of templates:

```
<xsl:template match="XPATH pattern">
    substitution part
</xsl:template>
```

- A pattern is matched with `match` and
- in the **`substitution part`** it is described how it is processed:
- The **`substitution part`** contains Markup, Text and XSLT elements.

Stylesheets: Building Blocks

- Important elements in the **`substitution part`**:
  - `</xsl:value-of select="XPath expression">`
    chooses textual value of the XPath expression, left-to-right concatenation.
  - `</xsl:apply-templates [select="XPath expression"]>`
    determines whether and which templates will be further applied at the current position:
    - w/o `select` : all child nodes with all templates
    - with `select` : all matching descendants with all templates
    - only some templates: by use of `call-templates`

Details: see below, slide 32f.

Stylesheets: How to use

Stylesheet overall structure:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">


          <xsl:template …>
                    …
          </xsl:template>
          …
          List of templates
          …
</xsl:stylesheet>
```

Link the stylesheet in a (source) document:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="lehre.xsl"?>
<lehre>
          …
<lehre>
```

Stylesheets: Processing

- Documents are processed depth-first
- Check whether current node matches a template
- Execute template (specific templates first)
- If “apply-templates” then process child nodes, otherwise backtrack

A simple example:

```
<?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" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
  <testoutput>
    <xsl:apply-templates/>
  </testoutput>
  </xsl:template>

<xsl:template match="test1">
    <test>
       <att1>
          <xsl:value-of select="@att1"/>
       </att1>
       <xsl:apply-templates/>
    </test>
</xsl:template>

<xsl:template match="test2">
   <xsl:element name="test123">
     <xsl:attribute name="att2"><xsl:value-of select="."/></xsl:attribute>
   </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Create new document element

Make element from attribute

Make attribute from element

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="test.xslt"?>
<document>
  <test1 att1="AnAttribute">
      <test2>AnElement</test2>
  </test1>
</document>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<testoutput>
  <test>
   <att1>AnAttribute</att1>
   <test123 att2="AnElement"/>
  </test>
</testoutput>
```

XSLT: How to match patterns

<xsl:template match="**XPATH pattern**">

- a pattern is a set of XPath location paths separated by **|** (union)
- restrictions: only the **child** (default) and **attribute** (**@**) axes are allowed here
- extensions: the location paths may start with **id(..)** or **key(..)**

XSLT: Template

There are many different kinds of template constructs:

- literal result fragments
- recursive processing
- computed result fragments
- conditional processing
- sorting
- numbering
- variables and parameters
- keys

Literal result fragments

A literal result fragment is:

- a text constant (character data)
- an element not belonging to the XSL namespace
- **`<xsl:text ...> ... </...>`** as raw text, but with keeps white-spaces and character escaping control, e.g.:

  ```
  <xsl:text disable-output-escaping="yes">    write &lt; </xsl:text>
  ```
- **`<xsl:comment> ... </...>`** (inserts a comment **<!-- …-->**)

Recursive processing of templates:

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

apply pattern matching and template instantiation on selected nodes (default: all children)

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

invoke template by name (where xsl:template has name="..." attribute, a named template does not need a match condition)

```
<xsl:for-each select="XPath expression"> template </xsl:for-each >
```

instantiate inlined template for each node in node-set (document order by default)

```
<xsl:copy> template </xsl:copy>
```

copy current node to output and apply template,

(shallow copy, without child nodes, attributes or other content!)

```
<xsl:copy-of select="..."/>
```

copy selected nodes to output (deep copy, includes descendants)

Computed result fragments

Result fragments can be computed using XPath expressions:

`<xsl:element name="..." namespace="..."> ... </...>`
construct an element with the given name, attributes, and contents

`<xsl:attribute name="..." namespace="..."> ... </...>`
construct an attribute (This has to occur inside xsl:element !!!)

`<xsl:value-of select="..."/>`
construct character data or attribute value (expression converted to string)

`<xsl:processing-instruction name="..."> ... </...>`
construct a processing instruction

The attributes may contain {*expression*}: XPath expressions which are

evaluated (and converted to string) on instantiation. Example (from slide 7):

```
<xsl:template match="veranstaltung">
 <xsl:element name="{@type}{@year}">
   <xsl:attribute name="titel">
     <xsl:value-of select="title"/>
   </xsl:attribute>
 </xsl:element>
</xsl:template>
```

→

```
<lecture2003 titel="Telecooperation"/>
<lecture2004 titel="Telecooperation"/>
<seminar2004 titel="Next Web Generation"/>
```

Conditional processing

Processing can be conditional:

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

apply template if expression (converted to boolean) evaluates to true (like in XPath predicates and filters)

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

test conditions in turn, apply template for the first that is true

```
<xsl:template match="veranstaltung">
 <course>
   <xsl:if test="@year = 2004">
     <xsl:attribute name="thisyear">yes</xsl:attribute>
   </xsl:if>
 </course>
</xsl:template>
```

→

```
<course/>
<course thisYear="yes"/>
<course thisYear="yes"/>
```

Sorting:

`<xsl:sort select="`*`expression`*`" .../>`
a sequence of `xsl:sort` elements placed inside `xsl:apply-templates` or `xsl:for-each` defines a lexicographic order

(default: document order)

- Some extra attributes:
  `order="ascending/descending"`
  `lang="..."`
  `data-type="text/number"`
  `case-order="upper-first/lower-first"`

Example: Sort by lecturer name

```
<xsl:template match="/">
  <teaching>
  <xsl:apply-templates select="./*/veranstaltung">
    <xsl:sort select="lecturer"/>
  </xsl:apply-templates>/>
  </teaching>
</xsl:template>
```

Numbering

- For numbering of lists, sections, items,etc.:

```
<xsl:number     value="expression"          converted to number
                format="... "               (as ol in HTML, default: "1. ")
                level="..."                 any / single / multiple
                count="expression"          select what to count
                from="...“                  select where to start counting
                lang="..." letter-value="..."
                grouping-separator="..." grouping-size="..."/>
```

level="single"	(default) numbering with respect to sibling nodes
level="all"	all nodes of the same type in the tree
level="multiple"	like "all", but numbering respects depth (1.1, 1.1.2, 2.3.4)

```
<xsl:template match="exam">
        <xsl:number count="exam" format="1."/> Exam: <xsl:value-of select="date"/>
</xsl:template>
```

Variables & Parameters

- Rudimentary variable definitions are possible (cannot be updated)
- Global or local in templates

Declaration:

```
<xsl:variable name="..." select="expression"/>
```

variable declaration, value given by XPath expression

or

```
<xsl:variable name="...">
     template
</ xsl:variable >
```

variable declaration, where the value is a result tree fragment.

Refer to variables by `$`*name*

Very similar: `xsl:param`

`xsl:with-param` allows to pass parameters in `xsl:call-template` and `xsl:apply-templates`

Keys

`<xsl:key match="`*`pattern`*`" name="..." use="`*`node set expression`*`"/>`

declares set of keys - one for each node matching the pattern and for each node in the node set

Extra XPath function:

`key(`*`name expression`*`, `*`value expression`*`)`

returns nodes with given key name and value

Eample: `key(‘courseKey’,’t’)` returns all `veranstaltung` nodes which title starts with a ‘T’, or ‘t’, with the following key definition:

```
<xsl:key name="courseKey" match="veranstaltung"
use="translate(substring(titel, 1, 1),
               'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
               'abcdefghijklmnopqrstuvwxyz')"/>
```

Another example: Combining XML-documents:

- document() … another special XPAth function:

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

<xsl:variable name="emps" select="document('merge2.xml')"/>

<xsl:template match="/">
   <employees>
   <xsl:for-each select="employees/employee">
      <xsl:copy-of select="."/>
   </xsl:for-each>

   <xsl:for-each select="$emps/employees/employee">
      <xsl:copy-of select="."/>
   </xsl:for-each>
   </employees>
</xsl:template>

</xsl:stylesheet>
```

Attribute-Sets

Globally defined set of attributes:

```
<xsl:attribute-set name="myAtts">
          <xsl:attribute name="href">
                    <xsl:value-of select="@link"/>
          </xsl:attribute>
          <xsl:attribute name="font-size">
                    +1
          </xsl:attribute>
</xsl:attribute-set>

<xsl:template match="/url">
          <a xsl:use-attribute-set="myAtts">
                    <xsl:apply-templates/>
          </a>
</xsl:template>
```

Overview over XSL elements:

xsl:attribute	produces an attribute with a given name
xsl:use-attribute-set	for attribute-sets
xsl:value-of	returns text (extraction of source tree or variables)
xsl:element	produces an element with a given name (e.g. useful if element name produced by variable name)
xsl:text	produces text node
xsl:processing-instruction	produce PI
xsl:comment	produce a comment
xsl:copy-of	copy the subtree at a certain node
xsl:copy	copy current context tag
xsl:if	conditional
xsl:choose / xsl:when	alternatives
xsl:number	produce a formated number
xsl:key	set a key, usable with key(...) "extra" XPath function.
xsl:variable	define variables
xsl:param,xsl:with-param	define, use parameters

Default-Templates:

The following templates are pre-defined:

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

Guarantees that per default, all children are recursively processed without producing a child elements.

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

In general, for text nodes write the text out to the result document.

If you want to override these you have to create an empty template, e.g.:

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

or you can particularly suppress recursive processing of certain elements by empty templates:

```
<xsl:template match=„younot | youneither" />
```

XSLT & Browsers & Tools

- All current browsers should contain an XSLT processor.
- Other tools, e.g.
  - XMLSpy
    http://www.altova.com/
  - Sablotron
    http://www.gingerall.com/charlie/ga/xml/p_sab.xml
- APIs: e.g. Apache XALAN
  http://xml.apache.org

Other query languages: XQuery

- XML Query (XQuery): More powerful than XPath, W3C recommendation since 23 January 2007 only!
  - derived from an XML query language called Quilt [Quilt], which in turn
  - borrowed features from several other languages, including XPath 1.0 [XPath 1.0], XQL [XQL], XML-QL [XML-QL], SQL [SQL], and OQL [ODMG].
- XCerpt - a more academic one http://www.xcerpt.org/

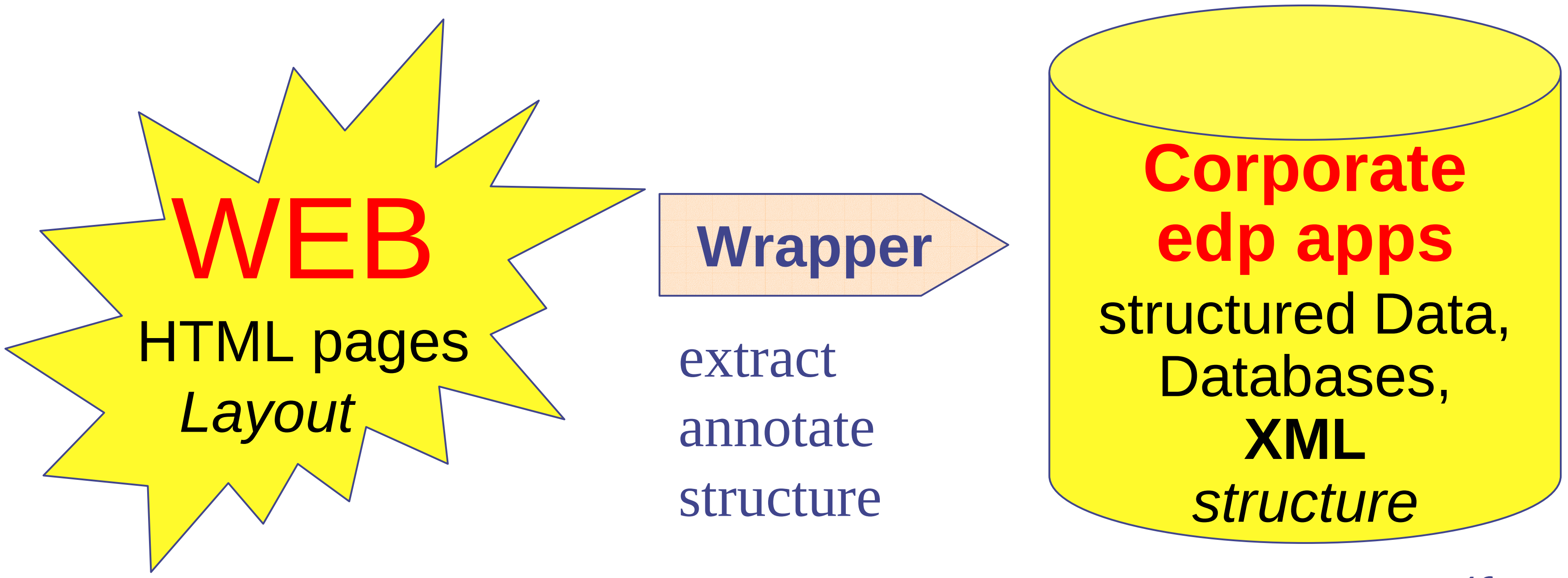
Q: Now how do we get the XML Data to be transformed from current Web or other Data?

A: Wrapper Tools

- A wrapper is an extraction program
- Takes information from Web Pages and turns it into more meaningful structured data, XML.
- Many tools around, cf.
  http://www.wifo.uni-mannheim.de/%7Ekuhlins/wrappertools/

Motivation: Bridge the Gap

Goal: Make Web content accessible for electronic data exchange.



More next lecture…

References

- **XPath:** http://www.w3.org/TR/xpath
- **XSLT:** http://www.w3.org/Style/XSL/

Possible presentation Topics:

OPTION 1: Present another W3C standard related to what we cover in the lecture:

- **XQuery**: XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/xquery/
- The Web Ontology Language (**OWL**)
  http://www.w3.org/2004/OWL/
- Semantic Annotations for WSDL and XML Schema (**SA-WSDL**)
  http://www.w3.org/TR/sawsdl/

OPTION2: Special Topics:

- Translating SPARQL (an RDF query language) to Prolog.
  This is a challenging one, requires some background in SQL and in Prolog, but I can provide a lot of support on this topic:
  http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf
- The relation of OWL and Logics
  I will give you two or three papers on this to read, this is about Logics and might be a nice one, if you attended my last course.
- Own suggestions might be sent in until next week by mail!

You are required to give a presentation and submit a short paper (Spanish or english)

Proyecto Fin de Master!!!!!

- **Beca: 6000 EUR**
- **20h/week, 1 year**

- Goal: Building a conference management system a la www.easychair.com, combining (Semantic) Web Technologies with Multi-Agent-Systems

--> Intelligent retrieval and Integration of Web information.

--> Implementation will need stuff we treated in this lecture!

Programming mostly in JAVA.