Java XML Processing API (JAXP) II:

XML Document Access and

Transformation with

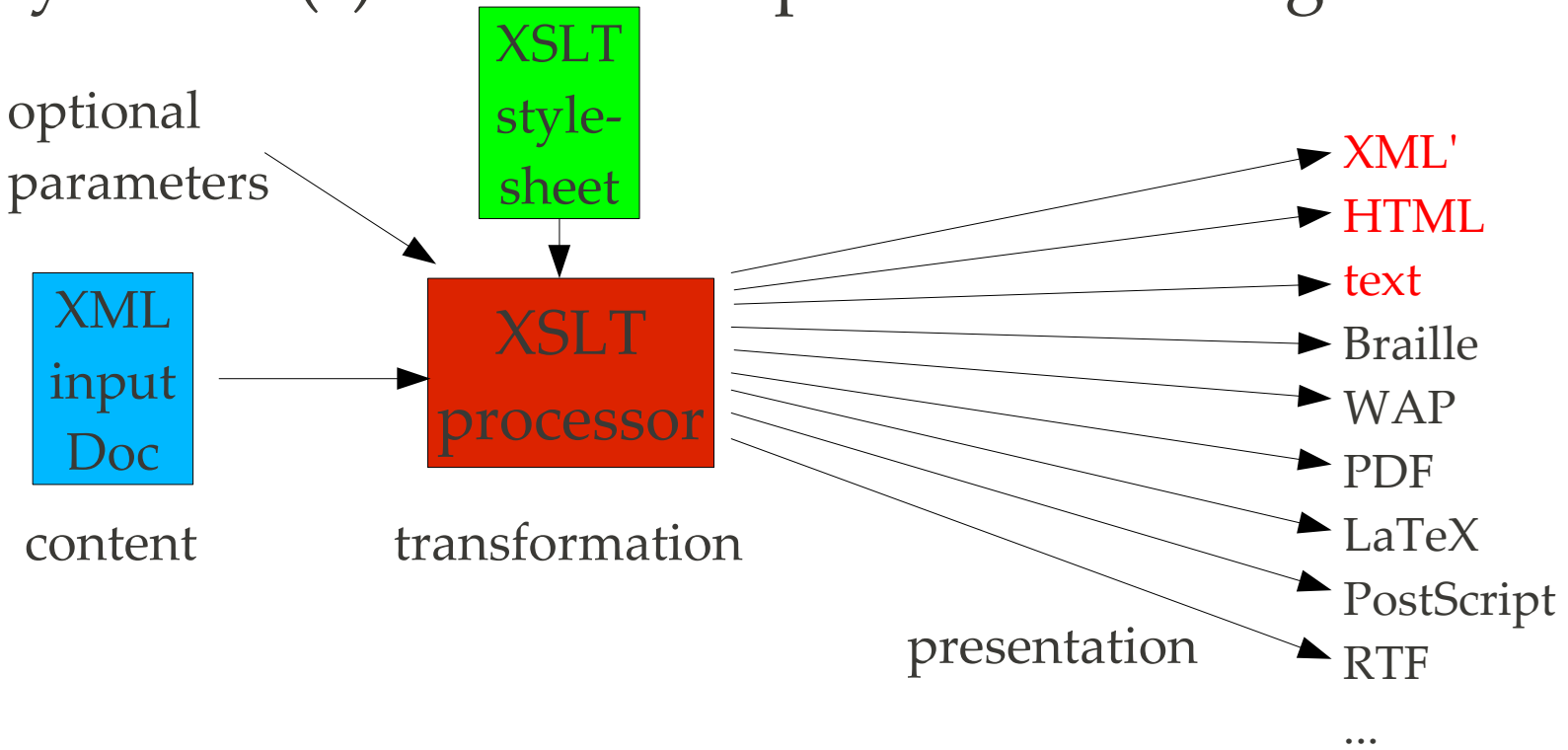
XPath and XSLT

Ulrich.Schaefer@dfki.de

Motivation

- Why XSLT in a Java for NLP lecture?
  - XML to XML transformation interesting for NLP (e.g. tree transformation, annotated corpora)
  - XML, XSLT becomes more and more important
  - XSLT part of JDK since 1.4
  - XPath now also available for DOM access (JDK 5)
  - XML parsing and manipulation in Java is time-consuming and inflexible with SAX and DOM
  - JDK 5: XSLT is much faster (thanks to XSLTC)

- general motivation and idea:
  - Web/document content is (should be) device-independent XML (e.g., DocBook DTD)
  - Stylesheet(s) for device-specific rendering:



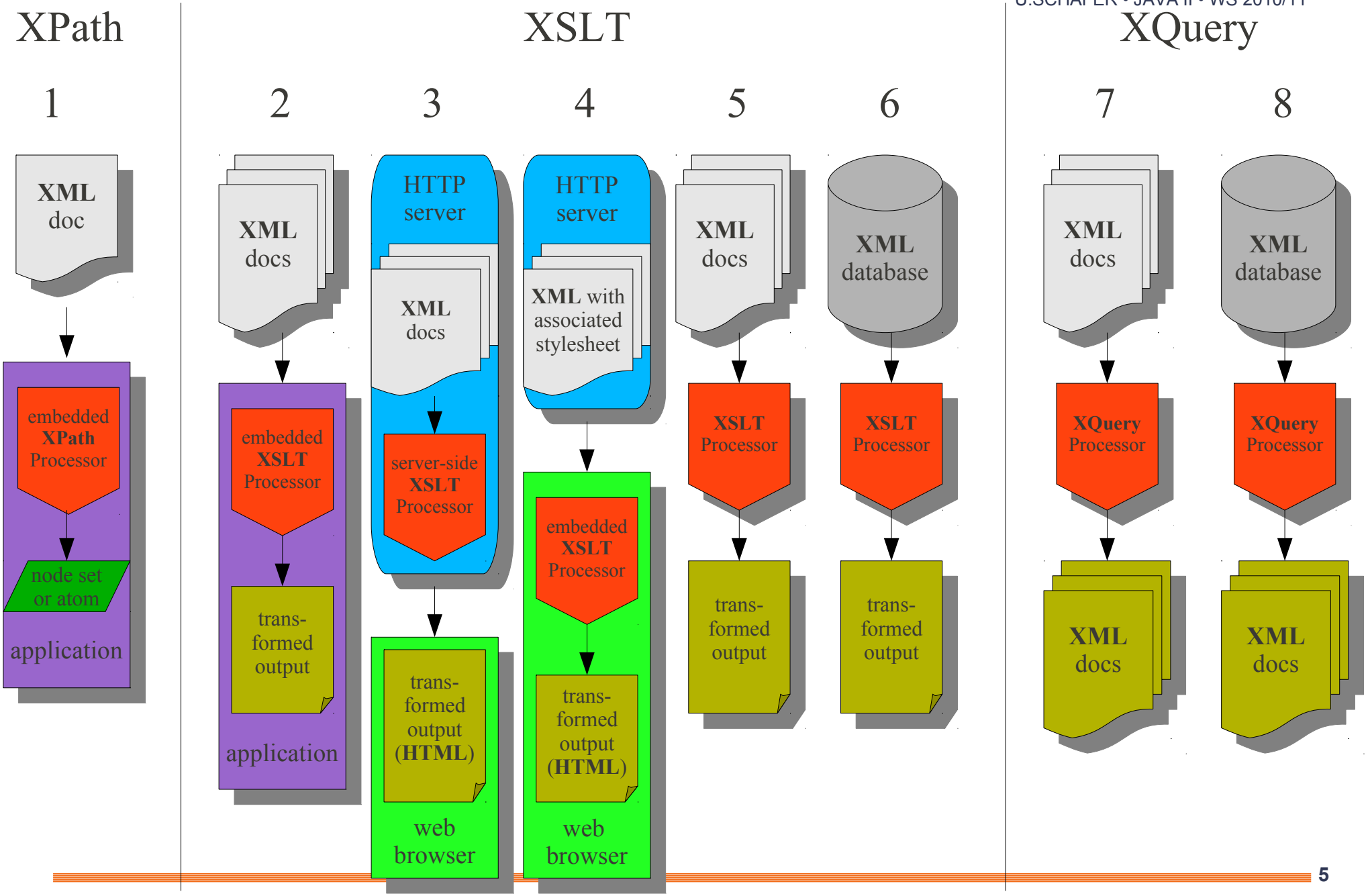
XSL Family

- XSL = XSLT (XSL transformations)
  + XPath (XML Path Language, also used in XQuery and DOM Level 3)
  + XSL-FO (XSL formatting objects)

- W3C recommendation in http://www.w3.org/Style/XSL/

- Related: XQuery (SQL-like queries on XML data)

XML querying in (Java) applications



XSLT (eXtensible Stylesheet Transformations)

- A language for transforming XML documents (tree transformation)
- W3C standard V1.0 (1999)
- XPath for e.g. finding nodes in XML tree, computations etc.
- similar to, but less powerful than DSSSL (SGML Document Style Semantics and Specification Language)
- XML syntax

XSLT advantages

- Easy navigation in document within a few lines of code (compare to DOM navigation in Java!)
- Quasi-declarative (as long as input structure is preserved)
- Specialized language for specific purpose (e.g., poor arithmetics)

An XSLT stylesheet is an XML document:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" encoding="utf-8"/>
    <xsl:param name="arg1">defaultvalue</xsl:param>
    <xsl:variable name="x">value</xsl:variable>
    here: stylesheet body with template definitions
</xsl:stylesheet>
```

<xsl:output>

- `method="xml"|"html"|"text"`: output format type
  - use xml for XML, html for HTML output
  - text for anything else (e.g., text, LaTeX, Javascript,...)
- `encoding="..."`: name of encoding in output XML header `<?xml version="1.0" encoding="..."?>` (only specifiy if really needed, e.g. if you write transformation result to a latin1 file).

Templates are subroutines (like java methods)

- matching templates are applied when they match a specified XPath pattern that describes a set of nodes in the XML input document.
- the output of a matching templates is part of the output of the stylesheet.
- matching templates are called implicitly by the XSLT processor (which may be triggered by an explicit `<xsl:apply-templates select="..."/>` call)
- most specific match wins in case of multiple matches

<xsl:template name="...">

- named templates can only be called explicitly, e.g., from other templates, using `<xsl:call-template name="...">`. They may return a value (e.g., number, String or node set).

- both matching and named templates may receive values as parameters from the caller
- variables defined within templates are local to the template, but global variables may be accessed

Empty Stylesheet: Default Processing

- An empty stylesheet (with no templates) copies only text elements to the output, i.e., there is an implicit template

```
<xsl:template match="text()">   <!-- match any text node -->
  <xsl:value-of select="."/>    <!-- output the match (=text) -->
</xsl:template>
```

- Define this if you don't want text to be copied:

```
<xsl:template match="text()"/>    <!-- empty template congesting
                                       text nodes -->
```

Hello World-Stylesheet


```
<xsl:stylesheet version="1.0"

  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/">  <!-- match root el. -->
     <greeting>Hello world!</greeting>
  </xsl:template>
</xsl:stylesheet>
```

input: an empty XML document

output:

```
<?xml version="1.0" encoding="utf-8"?>
<greeting>Hello world!</greeting>
```

- W3C standard (part of XSL family together with XSLT and formatting objects/XSL-FO)
- Part of/used in e.g.
  - XSLT, XPointer, XML:DB, DOM Level 3 (JDK 1.5!)
- purpose:
  - select nodes in an XML tree
  - perform computations on numbers, strings, nodes

XPath data types

- node set: set of nodes, e.g., "`sentence[2]/w`"
- boolean: true/false, e.g., "`starts-with('hello','h')`"
- number: floating-point number, e.g. "`2.4`"
- string: sequence of UCS characters, e.g. "`'hello'`"

- data types are not declared explicitly
- data types are converted automatically (or using `string()`, `boolean()`, `number()` functions)

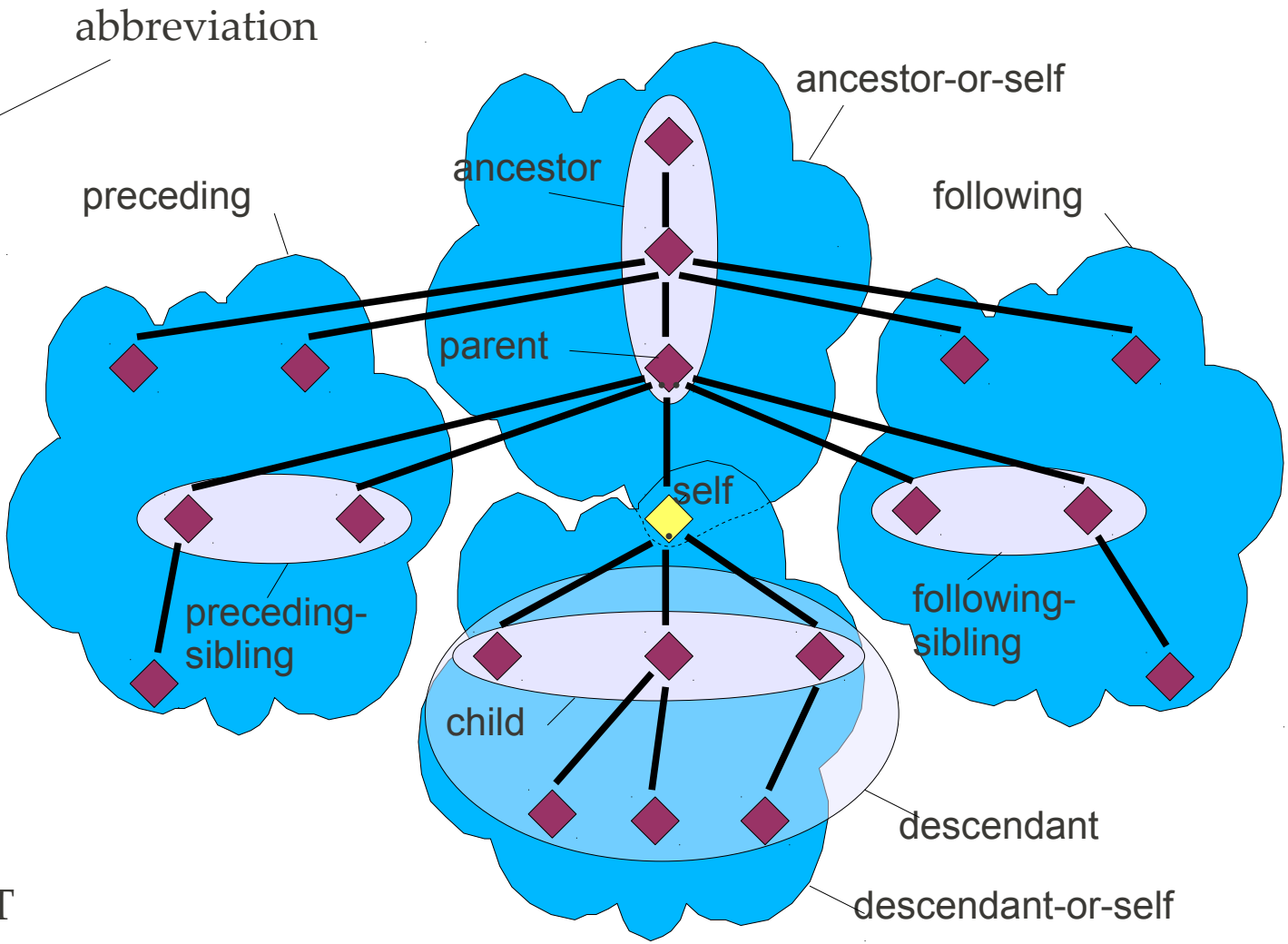
XPath Patterns (for matches)

Pattern	matches
/	root element node
sentence	sentence elements (sentence=sample element name)
*	any element node
text()	any text node
comment()	any comment node
@*	any attribute node
w[@pos='VFIN']	any w element with pos attribute = 'VFIN'
w[@pos and @cat]	only w elements that have pos and cat attribute
w|chunk	w or chunk elements

XPath axes: navigation in XML document tree

AxisName ::= 'ancestor'
| 'ancestor-or-self'
| 'attribute' (@)
| 'child' (/)
| 'descendant'
| 'descendant-or-self'
| 'following'
| 'following-sibling'
| 'namespace'
| 'parent' (..)
| 'preceding'
| 'preceding-sibling'
| 'self' (.)

Examples: parent::FSLIST
preceding-sibling::F
following-sibling::F



Axis examples

- `/book/chapter[1]/author/@name` returns value of the attribute name of the author element of 1st chapter
- `/book/chapter/author[@name='Smith']/../position()` returns chapter number(s) with author Smith
- `sentence/w[last()]` selects the last w child of each sentence element
- `ancestor::chunk` returns all ancestor elements up to the root node named chunk
- `../@cat` returns value of cat attribute of parent element

Node Set Functions

- position(): own position (child number) of node
- count(*nodeset*): number of nodes in nodeset
- last(): returns position of last node
- name(*nodeset*): name of element, attribute etc.

String functions

- string(*object*): translates object to a string
- concat(*string, string, ...*): concatenates strings
- string-length(*string*): returns length of string
- contains(*str1, str2*): true if str2 is part of str1
- starts-with(*str1, str2*): true if str1 starts with str2
- substring(*string, start* [,*length*]): returns substring of length length, starting from start

- not(*object*): negates argument which may be a nodeset: not(*nodeset*) is true if nodeset is empty
- lang(*string*): true if node or ancestor has lang attribute equal to string
- Boolean constants: true(), false()
- Boolean operators: and, or, =, !=, <, >, <=, >=

Number functions

- number(*object*): converts object to number
- sum(*nodelist*): adds values of nodelist (e.g. attributes)
- round(*number*): rounds number
- numeric operators: +, -, *, div,mod,floor(),ceiling()

- e.g. "`string-length('hello') + 1`" is 6

Back to XSLT: Parameters

- to stylesheet (from outside: Java, command line)
- to templates (both named and matching)
- must be 'declared':

```
<xsl:template name="format">
  <xsl:param name="arg1">default value</xsl:param>
  ...
</xsl:template>
```

call:

```
<xsl:call-template name="format">
  <xsl:with-param name="arg1" select="'hello world'"/>
<xsl:call-template>
```

Variables

- are untyped, i.e., may contain nodeset, string, ...
- are defined globally or within a template using

```
<xsl:variable name="lang" select="'en'"/>
```

- are referenced within expressions with $

```
<xsl:value-of select="$lang"/>
```

- cannot change their value, i.e. loops with count variables etc. must be defined recursively!
  - cf. named template 'repeat' later on

- Text:
  - verbatim in a template body (may be confusing)
  - `<xsl:text>This output text</xsl:text>`
    - raw text output without variables, expressions etc.
    - important for exact text output formatting (e.g. spaces in non-XML output)
  - `<xsl:value-of select="expression"/>`
    - text output from variables, computed XPath expressions etc.
    - example: `<xsl:value-of select="@cat"/>` outputs value of attribute 'cat'

- XML:
  - verbatim in a template body: `<FS type="avm"/>`
    Here, attribute values may contain XPath expressions enclosed in curly brackets:
    `<FS type="{concat('a', 'v' ,'m')}"/>`
  - by copying the current input element (without attributes): `<xsl:copy> ... </xsl:copy>`
  - Deep copy of input elements with expression:
    `<xsl:copy-of select="FS"/>`

- XML (continued):
  - by explicitly generating nodes:

```
<xsl:element name="FS">
  <xsl:attribute name="type">
     <xsl:value-of select="'avm'"/>
  </xsl:attribute>
</xls:element>
```

Conditionals

- If statement (no else alternative!)

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

- Switch statement (if... then... else)

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

Iteration

- For-each loops (with optional sorting)

```
<xsl:for-each select='expr'>
   [<xsl:sort select='expr' order='ascending|
    descending' data-type='text|number|...'/>]
  ...
</xsl:for-each>
```

- Apply matching templates with optional sorting

```
<xsl:apply-templates select='expr'>
   [<xsl:with-param name='...' select='expr'/>]*
   [<xsl:sort select='expr' order='ascending|
    descending' data-type='text|number|...'/>]
</xsl:apply-templates>
```

Sort Example: Output in Reverse Order

- Input document:

```
<alphabet>
  <a/><b/><c/><d/><e/><f/><g/><h/><i/><j/><k/><l/><m/>
  <n/><o/><p/><q/><r/><s/><t/><u/><v/><w/><x/><y/><z/>
</alphabet>
```

- Stylesheet (body):

```
<xsl:output method="text"/>
<xsl:template match="/alphabet">
  <xsl:for-each select="*">
    <xsl:sort select="position()" order="descending" data-type="number"/>
    <xsl:value-of select="concat(name(.), ' ')"/>
  </xsl:for-each>
</xsl:template>
```

- Output: z y x w v u t s r q p o n m l k j i h g f e d c b a

Example: print paths of atoms in TFS-XML

- Input: sample.xml from Ex. (TFS "Katzen...")
- Output: list of feature paths to FS nodes with atomic values, paths sorted alphabetically per feature structure in FSLIST. Example:

```
value(PRED|AGR|NUMBER) = pl
value(PRED|AGR|PERSON) = 3rd
value(PRED|STEM) = schnurren
value(SUBJ|AGR|GENDER) = fem
value(SUBJ|AGR|NUMBER) = pl
value(SUBJ|STEM) = Katzen
```

Example: print paths of atoms in TFS-XML


```
<FS type="avm">
   <F name="SUBJ">
     <FS type="infl">
        <F name="STEM">
           <FS type="Katzen"/>
        </F>
        <F name="AGR">
           <FS type="agr">
              <F name="NUMBER">
                 <FS type="pl"/>
              </F>
              <F name="GENDER">
                 <FS type="fem"/>
              </F>
           </FS>
        </F>
     </FS>
   </F>
```

```
<F name="PRED">
  <FS type="infl">
     <F name="STEM">
        <FS type="schnurren"/>
     </F>
     <F name="AGR">
        <FS type="agr">
           <F name="NUMBER">
              <FS type="pl"/>
           </F>
           <F name="PERSON">
              <FS type="3rd"/>
           </F>
        </FS>
     </F>
  </FS>
</F>
</FS>
```

Example: print paths of atoms in TFS-XML


```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:strip-space elements="*"/>  <!-- delete newlines etc. in input -->
  <xsl:output method="text"/>

  <xsl:variable name="newline">    <!-- constant for inserting newlines -->
    <xsl:text>
</xsl:text>
  </xsl:variable>

  <xsl:template match="F">         <!-- append feature name to path -->
    <xsl:param name="path"/>
    <xsl:apply-templates select="FS">
      <xsl:with-param name="path" select="concat($path, @name)"/>
    </xsl:apply-templates>
  </xsl:template>
```

Example: print paths of atoms in TFS-XML


```
<xsl:template match="FS">
  <xsl:param name="path"/>                    <!-- contains concatenated path-->
  <xsl:variable name="sep">
    <xsl:if test="not(parent::FSLIST)">  <!-- append | only if not root -->
      <xsl:value-of select="'|'"/>
    </xsl:if>
  </xsl:variable>
  <xsl:choose>                                <!-- print value if no features-->
    <xsl:when test="count(F) = 0">
      <xsl:value-of select="concat('value(',$path,') = ',@type,$newline)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="F">   <!-- sort according to features-->
        <xsl:sort select="@name"/>
        <xsl:with-param name="path" select="concat($path, $sep)"/>
      </xsl:apply-templates>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Recursion example: print a character n times


```
<xsl:template name="repeat">                <!-- print $char $times x -->
 <xsl:param name="times" select="1"/>       <!-- number of chars to repeat -->
 <xsl:param name="char"  select="' '"/>     <!-- character to repeat -->
 <xsl:if test="$times > 0">                 <!-- stop recursion if $times=0 -->
  <xsl:value-of select="$char"/>            <!-- return single character -->
  <xsl:call-template name="repeat">         <!-- recursion -->
   <xsl:with-param name="times" select="$times - 1"/>   <!-- pass params -->
   <xsl:with-param name="char" select="$char"/>
  </xsl:call-template>
 </xsl:if>
</xsl:template>
```

XSLT debugging with <xsl:message>


```
<xsl:message>
  <xsl:text>Value of variable x=</xsl:text>
  <xsl:value-of select="$x"/>
</xsl:message>
```

prints a message during processing to the screen
(not to the generated output document)

Referencing and Merging NLP markup

- Method 1 (referencing):
  - use unique attributes (marked as ID in source DTD)
  - use IDREF in same(!!) document, already shown, advantage: no transformation necessary
  - or use id() function to incorporate values (XPath/XSLT)
- Method 2 (merging):
  - use document() function to incorporate parts of source document in target document (XPath/XSLT)

Method 1: Referencing Markup (example)

- source annotation (id *must* be a DTD-declared ID attribute):

```
<chunk id="C0" cat="PP" length="11" offset="11">
  <w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
  <w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
</chunk>
```

- reference in stylesheet using id() function:

```
<xsl:value-of select="id('W2')/@pos"/>
```

  -> APPR

Method 2: Merging Markup (example)

- source annotation:

```
<chunk id="C0" cat="PP" length="11" offset="11">
  <w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
  <w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
</chunk>
```

- merging e.g. via attribute-value-based selection:

  `<xsl:for-each select="document(chunks.xml)//w[(@offset > 10) and (@offset < 23)]">` loops over

```
<w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
<w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
```

Walking through IDREFS (part 1, input)


```
<!DOCTYPE standoff [<!ATTLIST W id   ID     #REQUIRED > ]>
...
<W id="W0" length="4" offset="0">Jean</W>
<W id="W1" length="5" offset="5">Marie</W>
<W id="W2" length="6" offset="11">Lucien</W>
<W id="W3" length="6" offset="18">Pierre</W>
<W id="W4" length="7" offset="25">Anouilh</W>
<W id="W5" length="3" offset="33">was</W>
<W id="W6" length="4" offset="37">born</W>
<W id="W7" length="2" offset="42">on</W>
<W id="W8" length="4" offset="45">June</W>
<W id="W9" length="2" offset="50">23</W>
<W id="W10" length="1" offset="53">,</W>
<W id="W11" length="4" offset="55">1910</W>
<W id="W12" length="2" offset="60">in</W>
<W id="W13" length="8" offset="63">Bordeaux</W>
<W id="W14" length="1" offset="72">.</W>
```

Walking through IDREFS (part 2)


```
<!DOCTYPE standoff [<!ATTLIST NE id    ID      #REQUIRED
                                wrefs IDREFS  #REQUIRED
                                nt    NMTOKEN #REQUIRED > ]>
...
<NE id="NE0" wrefs="W0 W1 W2 W3 W4"   nt="PNAME"/> <!-- Jean ... Anouilh -->
<NE id="NE1" wrefs="W7 W8 W9 W10 W11" nt="DATE"/>  <!-- on June ... 1910 -->
<NE id="NE2" wrefs="W12 W13"          nt="LOC"/>   <!-- in Bordeaux.     -->
```

Walking through IDREFS with xsl:for-each and id():

```
<xsl:for-each select="id(id('NE0')/@wrefs)">
   <xsl:value-of select="text()"/><xsl:text> </xsl:text>
</xsl:for-each>
```

will return the original text of the W nodes referenced by Named Entity NE0, i.e. `Jean Marie Lucien Pierre Anouilh`.

Numbering without variables: <xsl:number>

<xsl:number> inserts a customizable numbering as text

Attributes:

- **count**: node to number
- **level**: element level, i.e. single, multiple or any
- **format**: select pre-defined numbering schema, e.g. decimal (1, 01), roman (i, I), letters (a, A)
- **from**: start number at given value
- **value**: calculating expression for count value
- **grouping-separator**: separator symbol in large numbers: , or .
- **grouping-size**: typically 3 for decimal numbers

Numbering without variables: <xsl:number>

Example: XML input

```
<infl>
  <number num="singular">
    <person/>
    <person/>
    <person/>
  </number>
  <number num="plural">
    <person/>
    <person/>
    <person/>
  </number>
</infl>
```

Numbering without variables: <xsl:number>


```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/infl">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="person">
    <xsl:copy>
      <xsl:attribute name="infl">
        <xsl:value-of select="../@num"/>
        <xsl:text> </xsl:text>
        <xsl:number count="person"/>
      </xsl:attribute>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Example: XML output

```
<?xml version="1.0"?>
<infl>
  <person infl="singular 1"/>
  <person infl="singular 2"/>
  <person infl="singular 3"/>
  <person infl="plural 1"/>
  <person infl="plural 2"/>
  <person infl="plural 3"/>
</infl>
```

Numbering Variant with Roman numbering

In previous stylesheet, add format attribute:

```
<xsl:number count="person" format="i"/>
```

Output:

```
<?xml version="1.0"?>
<infl>
  <person infl="singular i"/>
  <person infl="singular ii"/>
  <person infl="singular iii"/>
  <person infl="plural i"/>
  <person infl="plural ii"/>
  <person infl="plural iii"/>
</infl>
```

Numbering Variant with Level Attribute

In previous stylesheet, add level attribute:

```
<xsl:number count="person" level="any"/>
```

Output:

```
<?xml version="1.0"?>
<infl>
  <person infl="singular 1"/>
  <person infl="singular 2"/>
  <person infl="singular 3"/>
  <person infl="plural 4"/>
  <person infl="plural 5"/>
  <person infl="plural 6"/>
</infl>
```

Java XSLT processing (JAXP)

- package java.xml.transform
- DOMSource: input is DOM tree
- DOMResult: output is DOM tree
- StreamSource: read from Stream (File, String,...)
- StreamResult: write to Stream (File, String,...)
- TransformerFactory creates Transformer
- Transformer.transform(Source, Result)

- Example: transform XML File to String output

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
...
TransformerFactory tFactory = TransformerFactory.newInstance();
StringWriter resultStringWriter = new StringWriter();
StreamSource source = new StreamSource("input.xml");
StreamResult result = new StreamResult(resultStringWriter);
StreamSource stylesheet = new StreamSource("stylesheet.xsl");
Transformer transformer = tFactory.newTransformer(stylesheet);
transformer.transform(source, result);
String output = resultStringWriter.toString();
...
```

XSLT processors (1): JDK via command line

- JDK 1.4: `java org.apache.xalan.xslt.Process -IN input.xml -XSL stylesheet.xsl -OUT outputfile`
- This doesn't work in JDK 5 and 6 (Xalan XSLTC):
  http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=5099865
  reason: main method has been renamed to _main
- Workaround (src: xalanProcess.java):

```
public class xalanProcess {
   public static void main(String[] args) {
     com.sun.org.apache.xalan.internal.xslt.Process._main(args);
   }
}
```

XSLT processors (2)

- `msxsl.exe` (commandline tool, MS Windows only)
  download unless already installed (check first)
- `xsltproc` (Gnome libxslt + commandline)
- Apache ant `<xsl>` task (uses XSLT processor of underlying JDK)
- libs/extensions for Python, Perl, Lisp, C, C++, Javascript, PHP, ...
- Web browser! - inline `<?xml-stylesheet?>` in XML document (next slide)

Stylesheet application in web browser

- Most current web browsers support instruction `<?xml-stylesheet type="text/xsl" href="abc.xsl"?>` within an XML document. This starts XSLT transformation of the XML document using the specified stylesheet "abc.xsl"
- see http://www.w3.org/TR/xml-stylesheet

xsltxt: XSLT without XML syntax

- For those allergic to angle brackets
- Example:

```
tpl [/table/row] {
  if [@nullable] {
    nullable(#n = `@name`);
  } else {
    notNullable(#n = `@name`);
  }
}
```

- https://xsltxt.dev.java.net

XSLT equivalent:

```
<xsl:template match="/table/row">
  <xsl:choose>
    <xsl:when test="@nullable">
      <xsl:call-template name="nullable">
        <xsl:with-param name="n" select="@name"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="notNullable">
        <xsl:with-param name="n" select="@name"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Future: XSLT 2.0

- based on XPath 2.0 (partly shared with XQuery 1.0)
- user-definable functions
- multiple document output

- XSLT 2.0 W3C recommendation (= standard specification): http://www.w3.org/TR/xslt20

- JAXP XSLT is comparably slow
- Idea: compile XSL stylesheets directly to Java bytecode, called Translets
- implementation in Apache Xalan project
- XSLTC is default transformer since JDK 5 (JAXP 1.3) with automatic Translet compilation
- e.g. 6 times faster in Ambrosoft benchmark using DataPower XSLTMark

XML / XSLT Editing

- Emacs xslide package - http://www.menteith.com/xslide/
- jEdit - http://jedit.sourceforge.net: stable version 4.2 + XML + XmlIndenter + XSLT (+ XQuery) plugins
  Installation: http://www.dfki.de/~uschaefer/JEdit-XSLT.pdf
- Eclipse + Orangevolt XSLT plugin - http://eclipse.org
- XML-SPY (commercial, Windows) - http://xmlspy.com
- XML Cooktop (Windows) - http://www.xmlcooktop.com
- Amaya (XML editor) - http://www.w3.org/Amaya/
- EpcEDIT - http://www.epcedit.com

- W3C recommendation: XQuery 1.0, based on XPath 2.0: http://www.w3.org/TR/xquery/
- Query language for XML documents, mainly in XML databases (BDBXML, Tamino, Oracle)
- like SQL on XML (cf. structured XML documents and relational databases)
- comes with a query language in non-XML format

- preliminary implementations exist (e.g. http://saxon.sourceforge.net)
- seems to be better suited and more efficient for linguistic markup than existing XML corpus query languages (cf. NLPXML-2006 article by the NITE people)
- Structure of a query: **FLOWR** (For-Let-Where-OrderBy-Return)

XQuery example (from W3C use cases)

- For bicycle(s) offered by Tom Jones that have received a bid, list the item number, description, highest bid, and name of the highest bidder, ordered by item number.

```
<result>
  {
    for $seller in doc("users.xml")//user_tuple,
        $buyer in  doc("users.xml")//user_tuple,
        $item in  doc("items.xml")//item_tuple,
        $highbid in  doc("bids.xml")//bid_tuple
    where $seller/name = "Tom Jones"
      and $seller/userid  = $item/offered_by
      and contains($item/description , "Bicycle")
      and $item/itemno  = $highbid/itemno
      and $highbid/userid  = $buyer/userid
      and $highbid/bid =
max(doc("bids.xml")//bid_tuple[itemno = $item/itemno]/bid)
```

XQuery example (cont'd)


```
    order by ($item/itemno)
     return
         <jones_bike>
             { $item/itemno }
             { $item/description }
             <high_bid>{ $highbid/bid }</high_bid>
             <high_bidder>{ $buyer/name }</high_bidder>
         </jones_bike>
   }
</result>
```

XPath/XSLT Literature

- Further XSLT and XPath examples: ESSLLI 2009 slides
- W3C XSLT 1.0 recommendation: http://www.w3.org/TR/xslt
- Book: Michael Kay (author of Saxon): XSLT ('the' reference)
- Quick reference cards: http://www.mulberrytech.com/quickref/
- Sun JAXP Tutorial: Chapter 7 in J2EE 1.4 Tutorial (PDF, HTML): http://java.sun.com/xml/tutorial_intro.html
- http://www.cafeconleche.org/books/bible2/chapters/ch17.html
- W3C XPath 1.0 rec: http://www.w3.org/TR/xpath
- selfhtml: http://de.selfhtml.org/xml/darstellung/index.htm
- XSLTC: http://xml.apache.org/xalan-j/xsltc