Standard XML Query Languages for Natural Language Processing

Ulrich Schäfer

German Research Center for Artificial Intelligence (DFKI)
Language Technology Lab, Saarbrücken

ESSLLI 2009, 2nd week, 09:00–10:30

<slides>

Outline

Outline

Outline

Why this lecture?

- More and more corpora and other linguistic resources are available in XML format
- XML often plays role of abstract syntax (having concrete syntax at the same time)
- Well implemented and established standards for querying XML are ready to be used for
  - offline access to corpora
  - online integration of NLP components
  - combination and transformation of resources
- This is a practical course introducing the main concepts and elements of W3C's XML query languages, focusing on applications in NLP.

Before we start...

Is there anybody who knows

- ... what a recursive program (recursive function) is?
- ... programming languages such as Java, Python?
- ... what well-formed/valid XML means?
- ... what a (linguistically annotated) corpus is?

Why querying corpora or NLP XML output?

- getting statistics such as frequencies etc.
- combining resources
- pre-processing for machine learning (feature extraction)
- visualization (answer to query is visual representation)
- interfacing to applications (QA, search)

There is often no clear distinction between query and transformation.

What makes a good linguistic query language?

- not only data access, but also transformation and integration facilities
- enable composite and pipelined queries
- support relevant relationships within linguistic data
- abstract from low-level representation where possible
- efficient enough for online processing

Query languages tailored for a specific corpus annotation format come and go (die), and are often not portable, efficient or universal enough.

Corpora often survive 'their' query language.

→ use standard (e.g. W3C) languages instead even if they may seem too general

What will this lecture cover?

Introduction to W3C's three standard XML query languages

- XPath 1.0 and parts of 2.0
- XSLT 1.0 and parts of 2.0
- XQuery 1.0

with various NLP-related examples.

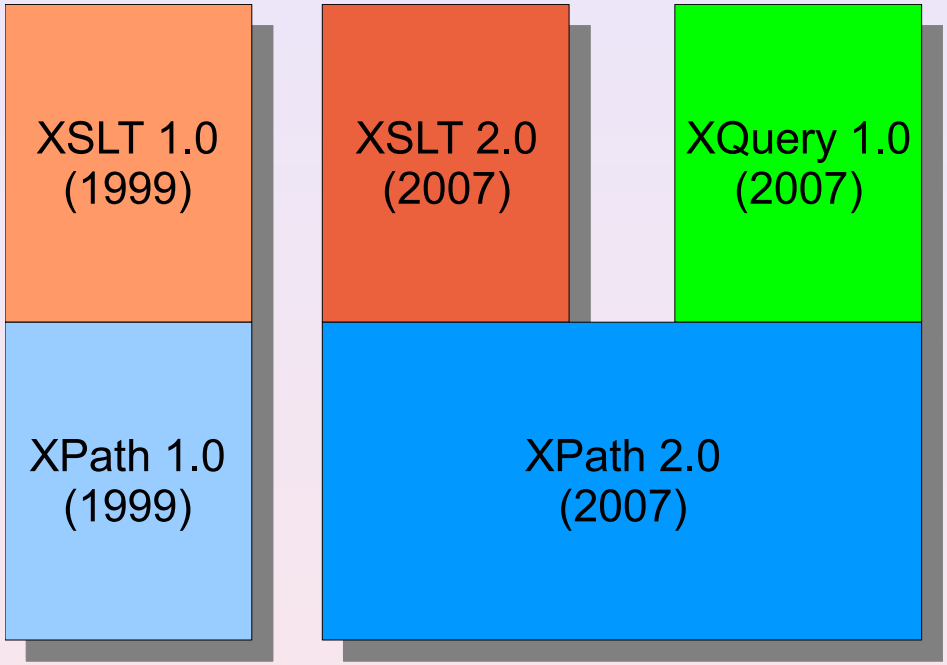


Figure: W3C XML query standards

More about this course

- Linguistic examples will be simple, but taken from real systems/data
- The course might also be of interest to non-linguists who want to learn XML query languages for other purposes than computational linguistics (web application development etc.)
- The query languages are typically embedded in other programming languages such as Java, Python or XML databases, examples will be given
- Most examples will work with standalone tools (libxslt, msxsl etc.)
- XSLT 2.0 and XPath 2.0 are only partially covered in favor of more in-depth introduction to the 1.0 versions

Course material online

Course material, e.g.

- slides
- source code
- online documentation
- bibliography
- links to useful software tools

is/will be made available at

http://www.dfki.de/~uschaefer/esslli09/

About DFKI LT Lab and myself

http://www.dfki.de/lt → Jobs!

http://www.dfki.de/~uschaefer

Outline

Text Markup Idea

Idea: enrich plain text with additional information for

- document structure (document semantics)
- meta-information (author, version, ...)
- linguistic annotation
- layout information

→ 'semi-structured' data

Text Markup Example

DocBook Example

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book lang="en">
  <bookinfo>
    <title>Booktitle</title>
  </bookinfo>
  <chapter>
    <title>Introduction</title>
    <para>This is <emphasis>running</emphasis> text.</para>
  </chapter>
</book>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Combining NLP Markup – An Extreme Case

- Tokenization
- Morphological analysis
- PoS tagging
- Chunking
- Named entity recognition (NER)
- Sentence boundary recognition

```
<S id="S0">
  <CHUNK id="H0" type="NP">
    <W id="W0" PoS="PPOSAT" mclass="971" tc="first_capital_word">Ihre</W>
    <W id="W1" PoS="NN" mclass="60" tc="first_capital_word">Lieblinge</W>
  </CHUNK>
  <CHUNK id="H1" type="VF_MODV_FIN">
    <W id="W2" PoS="VMFIN" mclass="16" tc="lowercase_word">will</W>
  </CHUNK>
  <CHUNK id="H2" type="NP">
    <NE id="N0" type="person" subtype="untitled">
      <W id="W3" tc="first_capital_word">Karin</W>
      <W id="W4" tc="first_capital_word">Schmittmann</W>
    </NE>
  </CHUNK>
  <W id="W5" PoS="ADJD" mclass="66" tc="lowercase_word">künftig</W>
  <W id="W6" PoS="ADV" mclass="102" tc="lowercase_word">noch</W>
  <W id="W7" PoS="ADJD" mclass="44" tc="lowercase_word">naturgetreuer</W>
  <CHUNK id="H3" type="PP">
    <W id="W8" PoS="APPR" mclass="264" tc="lowercase_word">auf</W>
    <W id="W9" PoS="NN" mclass="5" tc="first_capital_word">Seide</W>
  </CHUNK>
  <CHUNK id="H4" type="VF_V_INF">
    <W id="W10" PoS="VVINF" mclass="20" tc="lowercase_word">fixieren</W>
  </CHUNK>
  <W id="W11" tc="separator_symbol">.</W>
</S>
```

Outline

XML History

30 years from GML to XML (eXtensible Markup Language)

- 1969: GML=Generalized Markup Language
  pioneer: IBM (Goldfarb, Mosher, Lorie)
- 1986: ISO Standard 8879: SGML (=Standard GML)
- 1992: HTML (an SGML DTD instance)
- 1994: W3C founded (industry consortium)
- 1996: XML W3C working draft (SGML DTD)
- 1998: W3C XML 1.0 Standard ('recommendation')

From SGML to XML

SGML is/has been

- predecessor of XML
- mainly promoted by IBM
- comprehensive, hard to (fully) implement

→ Motivation for XML:

- WWW (early HTML mixes content and layout)
- support structured data
- SGML too complicated

From SGML to XML

XML can be formulated by a SGML DTD instance

XML inherits from SGML

- element/attribute syntax (but enclosing " mandatory for attribute values),
- a subset of the DTD Syntax

Comparison of SGML and XML:
http://www.w3.org/TR/NOTE-sgml-xml-971215.html

In addition to marking up text (“semi-structured data”), XML also targets at regular data (address book, lexicon,. . .)

Outline

XML document structure

- Elements (tags) that may have attributes with associated values – `<element attribute="value">`
- Text content – `text content`
- Comments – `<!-- comment -->`
- Entity References – `&title;`
  predefined: `'&amp;'` | `'&lt;'` | `'&gt;'` | `'&quot;'` | `'&apos;'`
  & < > " '
- Miscellaneous
  - Encoding and namespace information
  - Document type declarations
  - Processing instructions

XML documents must be well-formed

- at least one element (may be empty: `<x/>` is well-formed)
- exactly one root element
- elements may embed other elements (tree structure)
- start and end tags balanced (end tags repeat start tag name at same level: `<element>...</element>`)
- empty elements may be abbreviated: `<element/>`
- start tags (and only start tags) may bear attributes
- attributes must have values in single or double quotes
- attribute names must be unique per element
- content must contain legal XML characters

Encodings of XML documents

- 1st line in XML document:

```
<?xml version='1.0'encoding='utf-8'?>
```

- Every XML processor is obliged to support Unicode.
- Most other encodings including ISO-8859-1 (Latin 1) are optional, i.e. depending on implementation.
- XML processors can/try to determine encoding by parsing first line.
- (even without `encoding=`) – example:

```
UTF-8:                  3C    3F    78    6D    6C for  <?xml
UTF-16: FE FF 00 3C 00 3F 00 78 00 6D 00 6C (big-endian)
        FF FE 3C 00 3F 00 78 00 6D 00 6C 00 (little-endian)
```

- + further variants, see next slides resp. Appendix F in XML recommendation

Outline

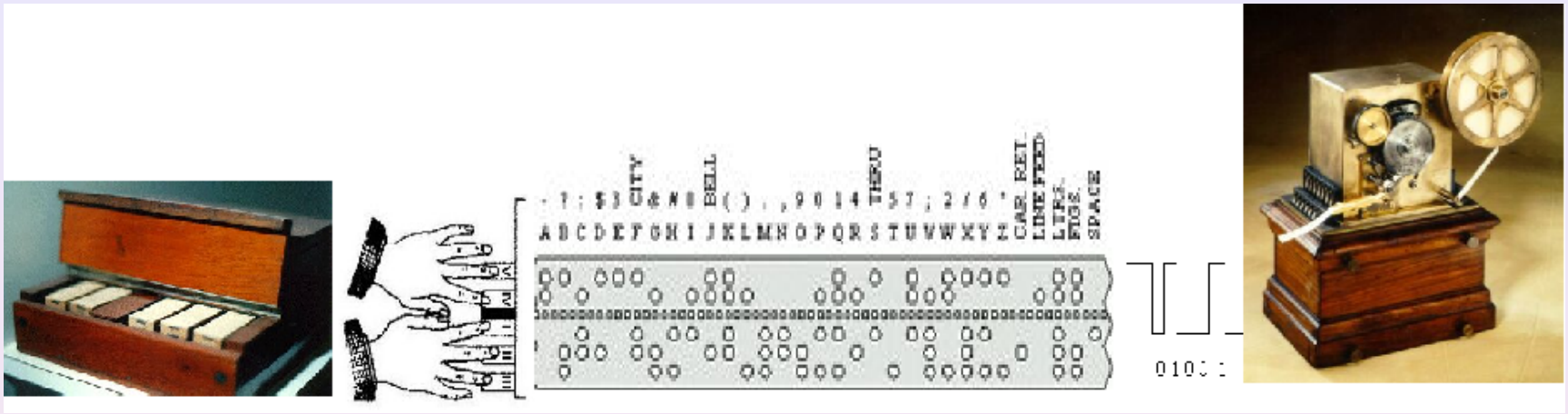
Motivation for this little excursus to encodings

- It is important to properly design NLP from the very bottom
- For text-based NLP: from characters and tokens
- This is especially important if you
  - use and combine different resources or components
  - intend to support multilinguality (recommended)
- $\longrightarrow$ make language and encoding explicit and configurable everywhere

Encodings

- Character Code: maps a character to a number (e.g. A: 65)
- Character Set: set of character codes (e.g. ASCII)
- Encoding: specific mapping (algorithm) between character codes and their digital representations (e.g. UTF-8, UCS2, EUC-JP)
- In case of ASCII, ISO-8859-X, there is (only) a 1:1 mapping

History



- First long-distance teletype 1877: Bordeaux–Paris!
- 5 bits Baudot code (Jean-Maurice-Émile Baudot, later: CCITT-2)
- 2 switching codes for letters vs. numbers/punctuation
- No distinction between upper/lower case (2^5=32 codes)
- Still today: news agencies and sea wheather forecasts (shortwave transmissions) at 45/50/75 Bd (6-11 characters/second)

From ASCII (1968) to Unicode (1993)

- 1968: ASCII (7 bit, 96 US keyboard characters and 32 control codes; no accent characters, umlauts etc.)
- 1984: ISO-8859-1 (8 bit, extension of ASCII, 192 most frequent Western European characters + 64 control codes)
- 1993: Unicode (ISO/IEC 10646)
  - Unicode = unique codes
  - each character has a unique numerical representation (code)
  - motivation: multilingual documents, uniform processing
  - BUT: multiple encodings (variable multi-byte, 16 bit, 32 bit, CPU architecture-dependent variants)

“The” Unicode: UCS

- UCS = Universal Multiple-Octet Coded Character Set
- 2 or 4 bytes for a character (UCS-2, UCS-4)
- UCS-2: 2^{16} = 65536 codes
- UCS-4: 2^{31} = 2.147.483.648 codes

Undefined characters (Unicode code positions) are illegal in XML!

UCS, BMP and ISO-8859

- UCS-2 (2 byte encoding) contains most important characters from all over the world → Java datatype `char`, Python Unicode character
- UCS-2 subset aka BMP (Basic Multilingual Plane)
- codes 0-255 identical with ISO-8859-1:
- `ISO8859-1: 'U'= 01010101 (8 bits)`
- `UCS-2: 'U'= 00000000 01010101 (16 bits)`
- `UCS-4: 'U'= 0000000 00000000 00000000 01010101 (31 bits)`
- other ISO 8859-X codes follow at higher positions

UTF = UCS Transformation Formats

- same codes, different digital representations!
- idea (for supporting legacy programming languages and applications):
  use multi-byte eight-bit sequences with variable length (similar to EUC-JP for Kanji)
- UTF-8: codes 0-127 identical with ASCII

```
Unicode Char (hex.)    UTF-8 representation (binary)
#x00000000-#x0000007F: 0xxxxxxx (=ASCII)
#x00000080-#x000007FF: 110xxxxx 10xxxxxx
#x00000800-#x0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
#x00010000-#x001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
#x00200000-#x03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
#x04000000-#x7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

- `xxx`=payload bits in UCS-2/4
- 4,5,6-byte long UTF codes are for UCS-4

UTF = UCS Transformation Formats

- Example: character ü

```
ISO-8859-1:              11111100 (#xFC)
UCS:          00000000 11111100 (#x00, #xFC)
UTF-8:        11000011 10111100 (#xC3, #xBC)
```

- all characters in Latin1 extension block (ISO-8859-1-equivalent codes $>$ 127) of the BMP already need two bytes in UTF-8!

Further variants

- UTF-16: extension of UCS-2 for characters beyond #xFFFF (UCS-4)
- UTF-32: same as UCS-4 with less characters

Why do Unicode/XML files sometimes start with FFFE or FEFF?
Answer: Endian byte order mark (BOM) (`#xFEFF = ZERO WIDTH NO-BREAK SPACE`)
Purpose: autodetection of byte order in files

Endian variants in memory and on file

- Big (SPARC, PowerPC) vs. Little (Intel, AMD) Endian processor architecture variants:
- ü in UCS-2LE: `11111100 00000000` (little endian)
- ü in UCS-2BE: `00000000 11111100` (big endian)

→ 13 file variants for UCS and UTF encodings: UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4LE, UCS-4BE, UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE

XML names and XML Character Entities

- XML standard says: XML text content may be full Unicode, but all XML names (elements, attributes) must comprise Unicode BMP (the 16 bit Unicode 'subset')
- Character entities can be used for transcription
- Syntax: either decimal or hexadecimal
- decimal: `&#2049;` = hexadecimal: `&#x0801;`
- `#xYYYY` always specifies UCS code ("the" Unicode), not the UTF encoding!
- DO NOT USE `&aacute;` etc. - this is HTML ONLY!
- Using XML character entities, any Unicode text can be encoded by only using ASCII characters

Lessons Learned

- All XML-processing software must support Unicode
- In most cases, choosing the UTF-8 encoding is a good idea
- Try to handle (count) Unicode characters as the least unit, i.e. Do not mix with byte-wise counting e.g. in C or Java byte arrays

Real world examples:

- XML parser error with `encoding="UTF-8"` in latin1-encoded file
- XML files with inconsistent encodings (parts collected from different sources)

Outline

XML Validation

XML standard says

- XML documents *must* be *well-formed* (see above)
- XML documents *may* be *valid*, i.e. validation against a schema is optional

Document structure and content follows rules specified by a grammar, e.g.

- DTD (Document Type Definition), or
- XML Schema, or
- Relax NG, or
- others

Most XML parsers offer at least DTD validation.

Document Type Definitions

(subset inherited from SGML)
`?xml` header (first line; optional) may be followed by a reference to a DTD (optional; may also be specified in-line):

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE nlp SYSTEM 'annotation.dtd'>

<nlp>
  ...
</nlp>
```

Document Type Definitions

A DTD defines in a BNF-like (easy to read) non-XML syntax

- admissible and required elements
- element nesting, sequence, choice, optionality
- required and optional (”implied”) attributes
- default values for optional attributes

- no classic data types are available
- document structure in focus
- similar to SGML DTD, but less powerful

Document Type Definitions - inline variant example

```
<?xml version='1.0' encoding='UTF-8'?>
  <!DOCTYPE nlp [
     <!ELEMENT nlp (sentence)* >
     <!ELEMENT sentence (w|chunk)* >
     <!ATTLIST sentence id ID #REQUIRED >
     <!ELEMENT chunk (w|chunk)* >
     <!ATTLIST chunk id   ID      #REQUIRED
                  cat    CDATA   #IMPLIED
                  length NMTOKEN #REQUIRED >
     <!ELEMENT w (#PCDATA) >
     <!ATTLIST w ttype "ccase|ucase|lcase|number">
     ...
     <!ENTITY % finiteVerb "VVFIN" >
  ]>
  <nlp>
    <sentence id="S0">
      <w id="W0" length="3" offset="0" ttype="ccase" pos="PPER">Wir</w>
      <w id="W1" length="6" offset="4" ttype="lcase" pos="&finiteVerb;">fuhren</w>
      <chunk id="C0" cat="PP" length="11" offset="11">
        <w id="W2" ...
```

Document Type Definitions - external variants

Alternative declaration places:

File: `DOCTYPE SYSTEM 'nlp.dtd'`

URL: `DOCTYPE PUBLIC 'http://www.purl.org/nlp.dtd'`

Element declaration: ELEMENT

```
<!ELEMENT name (RHS) >
```

name: element to declare, RHS (in parentheses): admissible daughters

Element declarations: RHS syntax

- `element` – daughter element name
- `,` – sequence
- `*` – zero or more occurrences
- `+` – one ore more occurrences
- `|` – alternative occurrence
- `#PCDATA` – text node
- `()` – grouping, prioritization

Element declaration: Example

The following DTD only defines the element skeleton (element structure plus textual content)

DTD element declarations

```
<!DOCTYPE article [
<!ELEMENT article (title, section+) >
<!ELEMENT section (title, paragraph+) >
<!ELEMENT paragraph (#PCDATA) >
<!ELEMENT title   (#PCDATA) >
]>
```

Attribute declarations: ATTLIST

```
<!ATTLIST element-name attribute-name value-type flag >
```

- element-name: name of element for which following attributes are defined
- attribute-name: name of attributes to be defined
- value-types: `NUMBER`, `ID`, `IDREFS`, `CDATA`, `NMTOKEN`, `NMTOKENS` or |-separated enumeration of possible string values
- flag: `#IMPLIED` (means optional) or `#REQUIRED`

Attribute declarations: Example

The following DTD defines elements with attributes

DTD attribute declarations

```
<!DOCTYPE article [
<!ELEMENT article (title, section+) >
<!ATTLIST article artno NUMBER #IMPLIED >
<!ELEMENT section (title, para+) >
<!ATTLIST section secid ID #REQUIRED >
<!ELEMENT para (#PCDATA) >
<!ELEMENT title (#PCDATA) >
]>
```

XML Entity declarations: ENTITY

Entities: abbreviation identifier for repeated text

Entity definition in DTD

```
<!ENTITY nlp "Natural Language Processing">
```

Usage

```
&nlp;
```

DTD design: Elements vs. Attributes

When you design a DTD (or schema, ...), ask yourself

- what should be elements: structure
- what should be attributes: atomic ”modifiers”
- what is text (attribute values or text elements)

Elements are extensible (by adding child elements or attributes), attributes are not.

Do not encode structure in attribute values!

Partial DTD declarations

`ID/IDREF` attribute declarations

- are required to indicate that attribute values bear unique keys for efficient storage and random retrieval
- can be declared standalone in DTD fragment
- can be dereferenced in XPath `id()` function

NLP application example for `id()` will be in XPath section.

Partial DTD declarations: Example

Partial DTD for ID/IDREF

```
<?xml version="1.0"?>
<!DOCTYPE standoff [
    <!ATTLIST w id ID #REQUIRED >
    <!ATTLIST ne parts IDREFS #IMPLICIT >
]>
<standoff>
  <words>
    <w id="W0">Harry</w>
    <w id="W1">Potter</w>
  </words>
  <named_entities>
   <ne id="NE0" parts="W0 W1"/>
  </named_entities>
</standoff>
```

XML Schema

- W3C Candidate Recommendation (http://w3.org/TR/xmlschema11-1)
- replacement for DTD
- finer-grained constraints than in DTD
- rich datatype repository predefined
- user-definable data types (including complex types)
- XML Syntax
- basis for XQuery 1.0/XPath 2.0/XSLT 2.0 data types

XML Schema: Example

XSD Example: Number interval

```
<xsd:simpleType name='NineteenthCentury'>
  <xsd:restriction base='xsd:integer'>
    <xsd:minExclusive value='1800'/>
    <xsd:maxInclusive value='1899'/>
  </xsd:restriction>
</xsd:simpleType>
```

DTD vs. XML Schema: Example

Application: order with items and prices

DTD fits on one slide

```
<!ELEMENT order (item)+>
<!ELEMENT item (name,price)>
<!ATTLIST item code NMTOKEN #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency NMTOKEN 'USD'>
```

DTD vs. XML Schema: Example

Equivalent schema: part 1/4

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

  <xsd:element name='order' type='Order'/>
  <xsd:element name='item' type='Item'/>
  <xsd:element name='name' type='Name'/>
  <xsd:element name='price' type='Price'/>

  <!-- <!ELEMENT order (item)+> -->
  <xsd:complexType name='Order'>
    <xsd:sequence>
      <xsd:element ref='item' minOccurs='1' maxOccurs='unbounded'/>
    </xsd:sequence>
  </xsd:complexType>
```

DTD vs. XML Schema: Example

Equivalent schema: part 2/4

```
<!-- <!ELEMENT item (name,price)> -->
<xsd:complexType name='Item'>
  <xsd:sequence>
    <xsd:element ref='name'/>
    <xsd:element ref='price'/>
  </xsd:sequence>
  <!-- <!ATTLIST item code NMTOKEN #REQUIRED> -->
  <xsd:attribute name='code'>
    <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
        <xsd:pattern value='[A-Z]{2}\d{3}'/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

DTD vs. XML Schema: Example

Equivalent schema: part 3/4

```
<!-- <!ELEMENT price (#PCDATA)> -->
<xsd:complexType name='Price'>
  <xsd:simpleContent>
    <xsd:extension base='NonNegativeDouble'>
      <!-- <!ATTLIST price currency NMTOKEN 'USD'> -->
      <xsd:attribute name='currency' default='USD'>
        <xsd:simpleType>
          <xsd:restriction base='xsd:string'>
            <xsd:pattern value='[A-Z]{3}'/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

DTD vs. XML Schema: Example

Equivalent schema: part 4/4

```
  <!-- <!ELEMENT name (#PCDATA)> -->
  <xsd:simpleType name='Name'>
    <xsd:restriction base='xsd:string'/>
  </xsd:simpleType>

  <xsd:simpleType name='NonNegativeDouble'>
    <xsd:restriction base='xsd:double'>
      <xsd:minInclusive value='0.00'/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Outline

XML Standards for Corpora and NLP

- TEI – Text Encoding Initiative
- ISO TC 37 SC 4
- OLAC – Open Language Archives Community
- OLIF – Open Lexicon Interchange Format
- OASIS – Advancing Open Standards for the Information Society: XLIFF (localization), DocBook, UIMA, ebXML, WS,...
- VoiceXML – Voice Extensible Markup Language
- ...

Querying XML

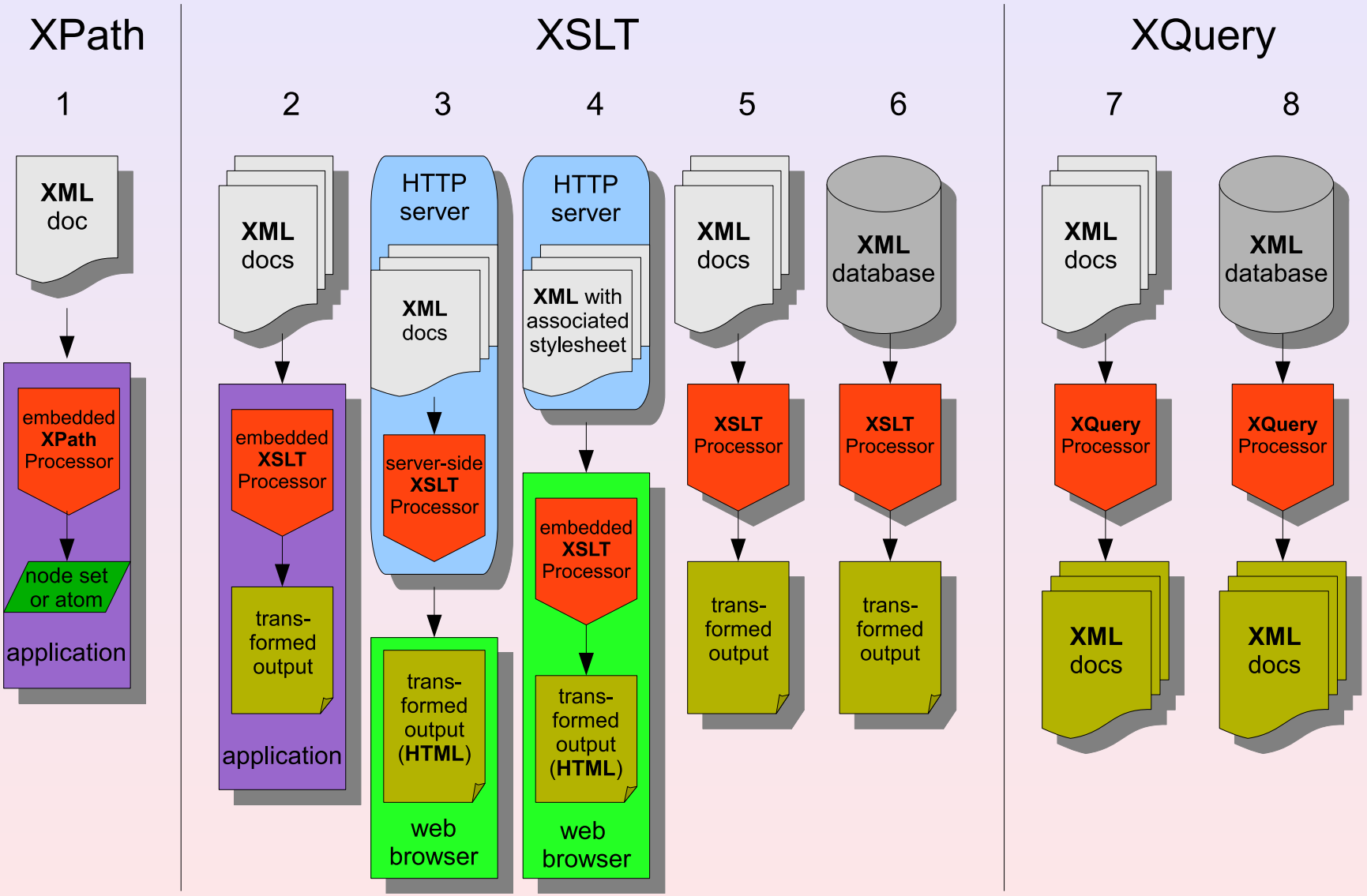
To access or merge markup, there are alternatives to parsing XML documents 'manually' (which is even worse for NLP markup as there a no commonly accepted standards).

We will discuss XPath, XSLT, XQuery in this course as standard languages also used in other fields than Computational Linguistics/Language Technology/Natural Language Processing.

Standard XML Query Languages

- **XPath**: embedded in applications, e.g. via DOM 3, and part of XSLT/XQuery
  output: node set or atom (string/number/bool) in V1.0
- **XSLT**: standalone, embedded in applications, web browser, web servers, as XML database interface
  output: XML, HTML, text
  - Variants: XSL-FO (formatting objects)
  - XPath + XSLT + XSL-FO form the Extensible Stylesheet Language Family (XSL)
- **XQuery**: usage similar to XSLT (except browser embedding), but mostly as XML database interface
  output: XML, ...?
  - Variants: XQuery-Fulltext

Typical XPath, XSLT, XQuery workflows



Outline

XML Data Model: Tree + X

Everything is a node

- Root node
- Element nodes
- Attribute nodes
- Text nodes
- Comment nodes
- Processing instruction nodes
- Namespace nodes

XPath 1.0 Data types

Attribute values, variables, parameters and return values of built-in functions

Data Type

Data types are not declared explicitly.
Data types are converted automatically (or using string(), boolean(), number() conversion functions).

- bool `"true()"`
- number `"2.4"`
- string `"'hello'"`
- node(set) `"chunk[2]/w"`

Conversion Function

- `bool()`
- `number()`
- `string()`
- `node()`

XPath Axes

XPath axes

```
                                  Abbrev.
AxisName ::=
  'ancestor'
| 'ancestor-or-self'
| 'attribute'                       @
| 'child'                           /
| 'descendant'
| 'descendant-or-self'             //
| 'following'
| 'following-sibling'
| 'namespace'
| 'parent'                          ..
| 'preceding'
| 'preceding-sibling'
| 'self'                            .
Examples: parent::SENTENCE
  preceding-sibling::CHUNK
  following-sibling::CHUNK
```

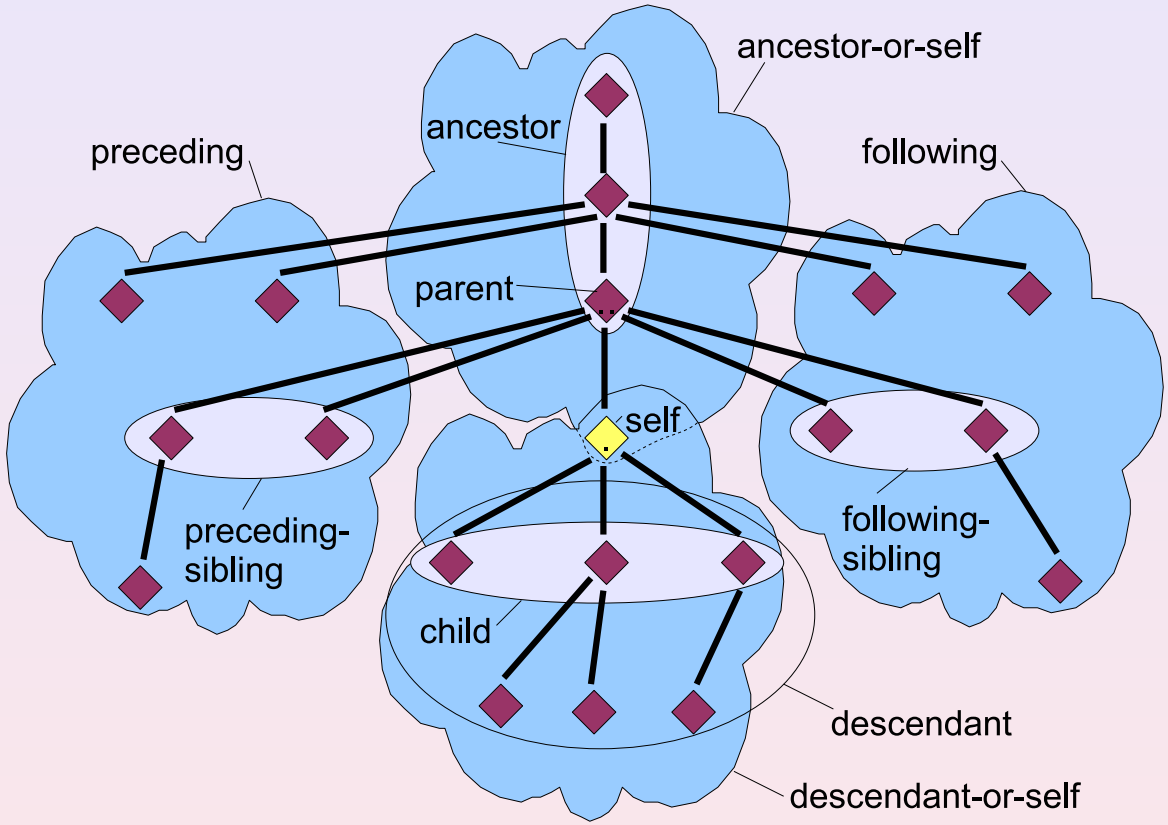


Figure: Axes in XML tree

Intuitive Path Notation

cf. Unix/Windows file system paths

Path Syntax

- absolute path: `/book/chapter/title`
- relative path: `chapter/title`
- current context: `.`
- one level up: `..`
- sibling: `../chapter`
- node wildcard: `*/title`
- path wildcard (at any depth): `//title`

XPath Operators

The usual syntax, except that `/` is path separator, not division

Path Syntax

- arithmetic: `+`, `-`, `*`, `div`, `mod`
  integer or floating point notation: `42`, `1.29`
- comparison: `=`, `!=`, `<`, `>`, `<=`, `>=`
- logical: `and`, `or`, `not()`
  `true()`, `false()` are constants (looking like functions)
- priority: `(`, `)`

XPath Examples

XPath expr.	Description
`/`	root element node
`sentence`	all daughter elements named `sentence`
`*`	any element node
`text()`	any text node
`comment()`	any comment node
`@*`	any attribute node
`w[@pos="VFIN"]`	any `w` element with Attribute `pos="VFIN"`
`w[@pos and @cat]`	only `w` elements that have `pos` and `cat` attributes
`w | chunk`	`w` or `chunk` elements

XPath 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 functions

XPath function	Description
`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

XPath function	Description
`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 [,len])`	returns substring of length `len`, starting from `start`

Boolean functions and operators

XPath function	Description
`not(object)`	negates argument which may be a nodeset: `not(nodeset)` is `true` if `nodeset` is empty
`lang(string)`	`true` if node or descendant has `lang` attribute equal to `string`
`true()`	Boolean constant
`false()`	Boolean constant

Number functions

XPath function	Description
`number(object)`	converts object to `number`
`sum(nodelist)`	sums over values in `nodelist`, e.g. attributes
`round(number)`	rounds `number`
`floor(number)`	integer $<=$ `number`
`ceiling(number)`	integer $=>$ `number`

Outline

Playing with XPath

Both tools require Java Runtime Engine

- JEdit + XSLT plugin
  - good result data type view
  - install via Linux distr. or from jedit.sourceforge.net
  - install XSLT plugin as explained
  - activate XPath sidebar
- BaseX
  - install from baseX webpage
  - interactive XPath 2.0 and XQuery editing
  - nice XML visualization options

Outline

XPath in Java

Querying DOM with XPath in Java

```
1  import java.io.File;
2  import javax.xml.parsers.DocumentBuilderFactory;
3  import javax.xml.parsers.DocumentBuilder;
4  import org.w3c.dom.Document;
5  import javax.xml.xpath.XPathFactory;
6  import javax.xml.xpath.XPath;
7  public class DomXpathTest {
8    public static void main(String [] args) {
9      try { // args[0]=xml filename, args[1]=XPath expression
10       DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
11       DocumentBuilder builder = domFactory.newDocumentBuilder();
12       Document dom = builder.parse(new File(args[0]));
13       XPathFactory xpathFactory = XPathFactory.newInstance();
14       XPath xpath = xpathFactory.newXPath();
15       System.out.print(xpath.evaluate(args[1], dom)); }
16     catch (Exception e) { System.err.print(e.toString()); }
17   }
18 }
```

XPath in Python

(tested with Python 2.5; lxml binding required)

Python libxslt example

```
1 from lxml import etree
2 from StringIO import StringIO
3
4 xmldoc = etree.parse(StringIO("<text>Inline XML document</text>"))
5 result = xmldoc.xpath('string-length(/text/text()) + $var', var=u"1000")
6 print result
7
8 # prints 1019.0
```

Outline

Introducing XSLT

- Language for transforming XML (tree transducer)
- Transforming = re-arranging, e.g.
  - Sorting
  - Grouping
  - Merging
- XML syntax
- XPath embedded for e.g. finding nodes in XML tree, computations etc.
- similar to, but less powerful than SGML's DSSSL (Document Style Semantics and Specification Language)

XSLT processing schema

- Easy navigation in document within a few lines of code (compare to DOM navigation!)
- Quasi-declarative as long as input structure is preserved BUT XSLT in general is **not** a declarative language as some have claimed
- Specialized language for specific purpose (e.g. poor arithmetics in version 1.0)

XSLT processing schema

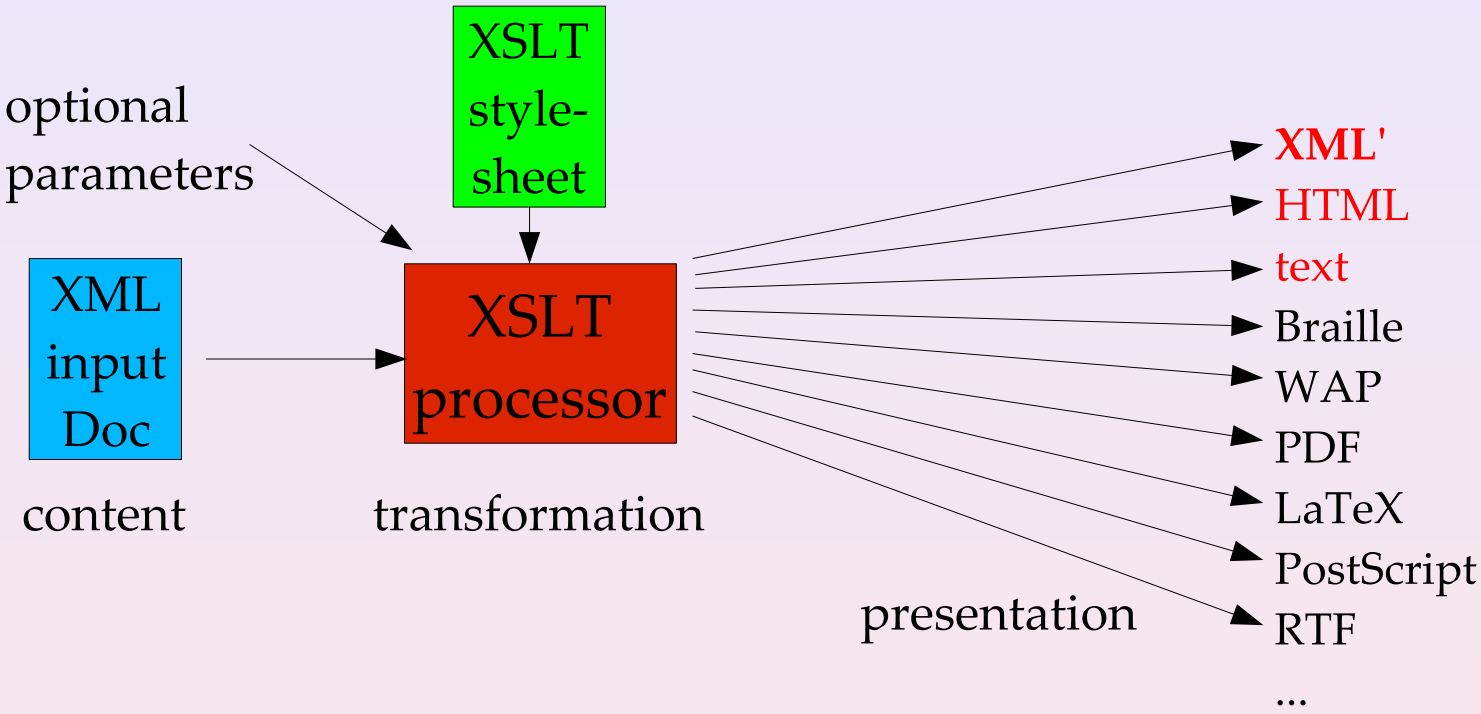


Figure: XSLT processing model

Running XSLT via command line

Unix (with libxslt)

```
xsltproc stylesheet.xsl input.xml > output.xml
```

Windows (with msxsl)

```
msxsl.exe input.xml stylesheet.xsl > output.xml
```

Saxon

```
saxon input.xml stylesheet.xsl -o output.xml
```

XalanJ

```
xalan -IN input.xml -XSL stylesheet.xsl -OUT output.xml
```

Running XSLT via processing instruction in XML file

By specifying a reference to a stylesheet within the XML file:

```
<?xml-stylesheet type='text/xsl' href='style.xsl'?>
```

The transformation will be performed by the XSLT processor that is part of modern web browsers. The transformation result will be rendered by the browser engine (typically (X)HTML, plain text or SVG.

NB1 `type='text/css'` for CSS formatting of XML documents (HTML in a web browser)

NB2 Generating JavaScript will not always deliver expected results

Outline

A first stylesheet

Code displays message and outputs message
Note: <xsl:message> is only allowed inside <xsl:template>

helloworld.xsl

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet version="1.0"
3       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5    <xsl:output method="text" encoding="utf-8"/>
6
7    <xsl:template match="/">
8      <xsl:message>Hello world!</xsl:message>
9      Hello World!
10   </xsl:template>
11
12 </xsl:stylesheet>
```

Passing Parameters from outside

Parameter `$who` defaults to `'world'`, but may be overwritten by externally specified value.

`<xsl:message>` is only allowed inside `<xsl:template>`

helloworldp.xsl

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet version="1.0"
3       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5    <xsl:output method="text" encoding="utf-8"/>
6    <xsl:param name="who" select="'world'"/>
7
8    <xsl:template match="/">
9      <xsl:message>Hello <xsl:value-of select="$who"/>!</xsl:message>
10     Hello <xsl:value-of select="$who"/>!
11   </xsl:template>
12
13 </xsl:stylesheet>
```

Transducing the tree

- <xsl:template match="..."> define transductions based on incoming elements
- <xsl:apply-templates> trigger other matching templates
- <xsl:call-template> explicitly call templates from within another template
- <xsl:for-each select="..."> loop through uniform structures
- <xsl:sort select="..."> rearrange/sort subtree (only as daughter element of <xsl:apply-templates> and <xsl:for-each>)
- <xsl:copy>, <xsl:copy-of>, <xsl:value-of>, <xsl:text> and others generate output

Generating output text

- Verbatim in a template body (may be confusing)
- `<xsl:text>This output text</xsl:text>`
  for 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'

Generating XML elements etc.

- Verbatim in a template body: `<FS type="avm"/>`
  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"/>`
- by explicitly generating nodes:

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

Conditionals

If statement (no else alternative!)

```
<xsl:if test='xpath-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>
```

Templates

Templates are subroutines (like java methods)

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

Matching Templates

- Syntax: `<xsl:template match="pattern">` **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
- empty templates can be used to suppress output
- matching templates are called implicitly by the XSLT processor
- or may be triggered by an explicit `<xsl:apply-templates match="pattern"/>` call
- most specific match wins in case of multiple matches

Named Templates

- Syntax: `<xsl:template name="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).

Built-in, implicit template rules

```
1  <!-- recursively descent elements -->
2  <xsl:template match="* | /">
3    <xsl:apply-templates/>
4  </xsl:template>
5
6  <!-- propagate mode where specified -->
7  <xsl:template match="* | /" mode="m">
8    <xsl:apply-templates mode="m"/>
9  </xsl:template>
10
11 <!-- copy text nodes and attributes -->
12 <xsl:template match="text() | @*">
13   <xsl:value-of select="."/>
14 </xsl:template>
15
16 <!-- suppress comments and processing instruction -->
17 <xsl:template match="processing-instruction() | comment()"/>
```

Built-in template rules: example

Input XML with elements, text nodes and comment

```
<book>
  <!-- chapter 1 starts here -->
  <chapter num="ch1">This is chapter 1</chapter>
</book>
```

Empty Stylesheet

```
1 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
2   <xsl:output method="text"/>
3 </xsl:stylesheet>
```

Output: text only

```
This is chapter 1
```

Variant 1: Copy comments

Same XML input as on previous slide

Empty Stylesheet, but copy comments

```
1 <xsl:stylesheet version="1.0"
2      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:output method="xml"/>
4   <xsl:template match="comment()">
5     <xsl:copy-of select="."/>
6   </xsl:template>
7 </xsl:stylesheet>
```

Output: text with comments (not well-formed)

```
1 <?xml version="1.0"?>
2   <!-- chapter 1 starts here -->
3     This is chapter 1
```

Variant 2: Copy comments

Same XML input as on previous slide; comment is converted to text

Empty Stylesheet, but copy comments

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="comment()">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Output: text with comments

```
  chapter 1 starts here
    This is chapter 1
```

Variant 3: Convert comments to text

Same XML input as before

Empty Stylesheet, but copy comments

```
1 <xsl:stylesheet version="1.0"
2      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3   <xsl:output method="xml"/>
4   <xsl:template match="comment()">
5     <xsl:value-of select="."/>
6   </xsl:template>
7 </xsl:stylesheet>
```

Output: text with comments (not well-formed)

```
<?xml version="1.0"?>
  chapter 1 starts here
    This is chapter 1
```

Sorting

xsl:sort must be an immediate child of either

- <xsl:for-each> or
- <xsl:apply-templates>

Example: Input XML with elements and text nodes

```
<corefs>
  For <coref cluster="1277">many NLP tasks</coref>, however,
  <coref cluster="603">we</coref> are confronted with
  <coref cluster="1100">new domains</coref> in which labeled
  <coref cluster="193">data</coref> is scarce or non-existent.
</corefs>
```

Sorting in for-each loop

Sorting in for-each loop

```
<?xml version="1.0"?>
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output method='xml'/>
  <xsl:template match="/corefs">
    <xsl:copy>
      <xsl:for-each select="coref">
        <xsl:sort select="@cluster" data-type="number"/>
        <xsl:copy-of select="."/>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="text()"/>
</xsl:stylesheet>
```

Sorting

Output sorted according to cluster attribute values

```
<?xml version="1.0"?>
<corefs>
  <coref cluster="193">data</coref>
  <coref cluster="603">we</coref>
  <coref cluster="1100">new domains</coref>
  <coref cluster="1277">many NLP tasks</coref>
</corefs>
```

Another Sorting Example: Reverse Output

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

Variables aren't really variables

XSLT variables...

- are untyped, i.e., may contain nodeset, string, number, boolean
- 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

Generic Stylesheets

XSLT stylesheets do not stick to a DTD or a Schema. Stylesheets may be written in a way that is general/generic and works independent of e.g. element names by using `name()` and $*$ matches, cf. example on next page.

On the other hand, stylesheets may also be used to check XML document rigidly and in a more finegrained way than DTD or even XML Schema processors can do (Schematron approach).

Generic Stylesheets with * match and name()

Transform any XML to Graphviz tree (dot file format)

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

  <xsl:template match="/"> <!-- output wrapping dot code -->
    digraph G { <xsl:apply-templates select="*"/> }
  </xsl:template>

  <xsl:template match="*">
    <xsl:variable name="node" select="concat(local-name(), position())"/>
    <xsl:for-each select="*">
      &quot;<xsl:value-of select='$node'/>&quot; -&gt;
      &quot;<xsl:value-of select="concat(local-name(), position())"/>&quot;;
    </xsl:for-each>
    <xsl:apply-templates select="*"/>
  </xsl:template>

</xsl:stylesheet>
```

Generic Stylesheet Example

```
xsltproc xml2dot.xsl xml2dot.xsl > graph && dot -Tps graph && gv graph.ps
```

Transformation result as input for Graphviz

```
digraph G {
 "stylesheet1" -> "output1";
 "stylesheet1" -> "template2";
 "stylesheet1" -> "template3";
 "template2" -> "apply-templates1";
 "template3" -> "variable1";
 "template3" -> "for-each2";
 "template3" -> "apply-templates3";
 "for-each2" -> "value-of1";
 "for-each2" -> "value-of2";
}
```

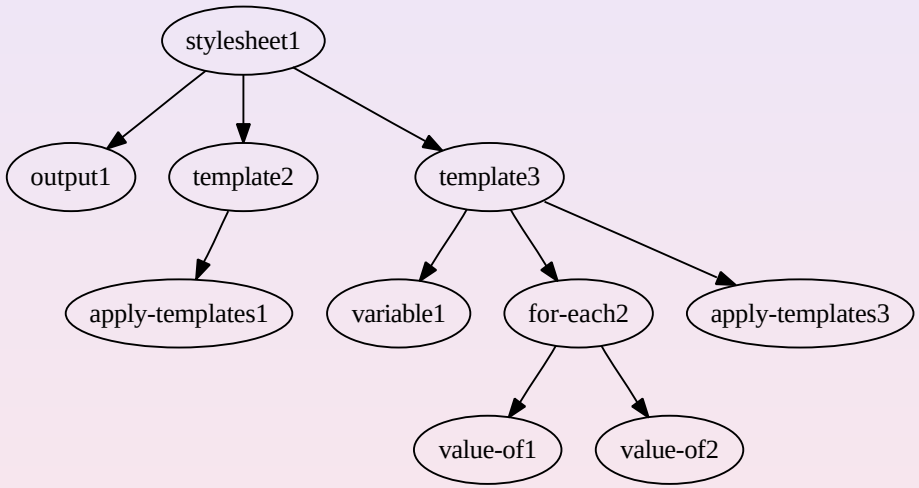


Figure: Graphviz output

Standoff NLP Markup Example: Input Words

```
<!DOCTYPE standoff [<!ATTLIST W id   ID    #REQUIRED
                                wref IDREF #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>
```

Standoff NLP Markup Example: Tokens

```
<!DOCTYPE standoff [<!ATTLIST T id   ID    #REQUIRED
                                wref IDREF #REQUIRED > ]>
...
<T id="T0"  wref="W0"  tc="first_capital_word"/> <!-- Jean     -->
<T id="T1"  wref="W1"  tc="first_capital_word"/> <!-- Marie    -->
<T id="T2"  wref="W2"  tc="first_capital_word"/> <!-- Lucien   -->
<T id="T3"  wref="W3"  tc="first_capital_word"/> <!-- Pierre   -->
<T id="T4"  wref="W4"  tc="first_capital_word"/> <!-- Anouilh  -->
<T id="T5"  wref="W5"  tc="lowercase_word"/>     <!-- was      -->
<T id="T6"  wref="W6"  tc="lowercase_word"/>     <!-- born     -->
<T id="T7"  wref="W7"  tc="lowecase_word"/>      <!-- on       -->
<T id="T8"  wref="W8"  tc="first_capital_word"/> <!-- June     -->
<T id="T9"  wref="W9"  tc="two_digit_number"/>   <!-- 23       -->
<T id="T10" wref="W10" tc="separator_symbol"/>   <!-- ,        -->
<T id="T11" wref="W11" tc="four_digit_number"/>  <!-- 1910     -->
<T id="T12" wref="W12" tc="lowercase_word"/>     <!-- in       -->
<T id="T13" wref="W13" tc="first_capital_word"/> <!-- Bordeaux -->
<T id="T14" wref="W14" tc="separator_symbol"/>   <!-- .        -->
```

Standoff NLP Markup Example: PoS Tags

```
<!DOCTYPE standoff [<!ATTLIST P id   ID    #REQUIRED
                                wref IDREF #REQUIRED > ]>
...
<P id="P0"  wref="W0"  pos="NNP"/> <!-- Jean     -->
<P id="P1"  wref="W1"  pos="NNP"/> <!-- Marie    -->
<P id="P2"  wref="W2"  pos="NNP"/> <!-- Lucien   -->
<P id="P3"  wref="W3"  pos="NNP"/> <!-- Pierre   -->
<P id="P4"  wref="W4"  pos="NNP"/> <!-- Anouilh  -->
<P id="P5"  wref="W5"  pos="VBD"/> <!-- was      -->
<P id="P6"  wref="W6"  pos="VBN"/> <!-- born     -->
<P id="P7"  wref="W7"  pos="IN"/>  <!-- on       -->
<P id="P8"  wref="W8"  pos="NNP"/> <!-- June     -->
<P id="P9"  wref="W9"  pos="CD"/>  <!-- 23       -->
<P id="P10" wref="W10" pos=","/>   <!-- ,        -->
<P id="P11" wref="W11" pos="CD"/>  <!-- 1910     -->
<P id="P12" wref="W12" pos="IN"/>  <!-- in       -->
<P id="P13" wref="W13" pos="NNP"/> <!-- Bordeaux -->
<P id="P14" wref="W14" pos="."/>   <!-- .        -->
```

Standoff NLP Markup Example: Chunks

```
<!DOCTYPE standoff [<!ATTLIST C id    ID     #REQUIRED
                                wrefs IDREFS #REQUIRED > ]>
...
<C id="C0" wrefs="W0 W1 W2 W3 W4"   ct="NP"/> <!-- Jean ... Anouilh -->
<C id="C1" wrefs="W5 W6"            ct="VP"/> <!-- was born         -->
<C id="C2" wrefs="W7 W8 W9 W10 W11" ct="PP"/> <!-- on June ... 1910 -->
<C id="C3" wrefs="W12 W13"          ct="PP"/> <!-- in Bordeaux.     -->
```

Standoff NLP Markup Example: Named Entities

```
<!DOCTYPE standoff [<!ATTLIST N id    ID     #REQUIRED
                                wrefs IDREFS #REQUIRED > ]>
...
<N id="N0" wrefs="W0 W1 W2 W3 W4"   nt="PNAME"/> <!-- Jean ... Anouilh -->
<N id="N1" wrefs="W7 W8 W9 W10 W11" nt="DATE"/>  <!-- on June ... 1910 -->
<N id="N2" wrefs="W12 W13"          nt="LOC"/>   <!-- in Bordeaux.     -->
```

Random Access to elements using id()

Function id() works independently of the current context node!

Accessing IDs

```
<xsl:value-of select="id('P5')/@pos" />
```

on PoS markup will return VBD (PoS tag of token P5)

Walking through ID nodes

Walking through nodes via IDREFS attribute

```
<xsl:for-each select="id(id('NO')/@wrefs)">
   <xsl:value-of select="text()"/><xsl:text> </xsl:text>
</xsl:for-each>
```

on chunks will return the original text of the `W` nodes referenced by Named Entity `NO`, i.e. `Jean Marie Lucien Pierre Anouilh`.

Numbering with xsl:number

<xsl:number> inserts a customizable numbering (as text)
Attributes:

- `count`: element to number
- `level`: `single`, `multiple` or `any`
- `format`: select pre-defined numbering schema (e.g. decimal, roman, letters)
- `from`: start number at given value
- `value`: expression for counting
- `grouping-separator`: separator symbol in large numbers
- `grouping-size`: (typically 3 for decimal numbers)

Numbering Elements

Numbering Example: Stylesheet

```
<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>
```

Numbering Elements

Numbering Example: XML input

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

Numbering Elements

Numbering Example: 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"/>
```

Numbering with format=”i”

```
<?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

In previous stylesheet, add level attribute:

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

Numbering with level=”any”

```
<?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>
```

Named Template with Parameters

Both named and matching Templates may take parameters. They are passed by the calling template using <xsl:with-param>.
In previous stylesheet, replace <xsl:number count="person"/> by

Passing parameters to a named template

```
<xsl:call-template name="ordinal">
  <xsl:with-param name="num">
    <xsl:number count="person"/>
  </xsl:with-param>
</xsl:call-template>
```

Template Parameters Example: Computing Ordinals

Passing Parameters to named and matching templates

```
<xsl:template name="ordinal">
  <xsl:param name="num" select="0"/>
  <xsl:variable name="m" select="$num mod 10"/>
  <xsl:choose>
    <xsl:when test="$m = 1">
      <xsl:value-of select="concat($num, 'st')"/>
    </xsl:when>
    <xsl:when test="$m = 2">
      <xsl:value-of select="concat($num, 'nd')"/>
    </xsl:when>
    <xsl:when test="$m = 3">
      <xsl:value-of select="concat($num, 'rd')"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="concat($num, 'th')"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Computing Ordinals Variant without xsl:choose

Passing Parameters to named and matching templates

```
<xsl:template name="ordinal">
  <xsl:param name="num" select="0"/>
  <xsl:variable name="m" select="$num mod 10"/>
  <xsl:value-of select='concat($num,
       translate($m,"1234567890","snrttttttt"),
       translate($m,"1234567890","tddhhhhhhh"))'/>
</xsl:template>
```

Recursive markup: TFS DTD

A recursive DTD

```
<!DOCTYPE FSLIST [
  <!ELEMENT FSLIST ( FS )* >
  <!ELEMENT FS ( F )* >
  <!ATTLIST FS type  NMTOKEN #IMPLIED >
  <!ELEMENT F  ( FS ) >
  <!ATTLIST F  name  NMTOKEN #REQUIRED >
]>
```

cf. Rationale for TEI feature structure markup (Langendoen and Simons 1995)

Similar representations also used in MAF, ISO TC 37 SC 4

Recursive markup: Input TFS example

AVM for 'Cats purr'

```
<FSLIST>
  <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>
</FSLIST>
```

Recursive markup: Output example

Output: indented text AVM

```
avm [SUBJ infl [STEM Katzen,
                AGR agr [NUMBER pl,
                         GENDER fem]],
     PRED infl [STEM schnurren,
                AGR agr [NUMBER pl,
                         PERSON 3rd]]]
```

Recursive markup: Stylesheet

Part 1/4: header

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- print TFS-XML in text AVM format -->

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

  <!-- generate non-XML output -->
  <xsl:output method="text"/>

  <!-- store newline character in variable -->
  <xsl:variable name="newline">
    <xsl:text>&#10;</xsl:text>
  </xsl:variable>
```

Recursive markup: Stylesheet

Part 2/4: FS match

```
<!-- match typed feature structures -->
<xsl:template match="FS">
  <xsl:param name="indent" select="0"/> <!-- # of spaces to insert -->
  <xsl:value-of select="@type"/> <!-- print type name -->
  <xsl:if test="count(F) > 0">   <!-- skip [] if no features -->
    <xsl:text> [</xsl:text> <!-- print opening bracket -->
    <!-- recursively print feature value pairs -->
    <!-- indentation is the string-length + 2, for ' [' -->
    <xsl:apply-templates select="F">
      <xsl:with-param name="indent"
                      select="$indent + string-length(@type) + 2"/>
    </xsl:apply-templates>
    <xsl:text>]</xsl:text> <!-- print closing brackets -->
  </xsl:if>
  <!-- start a new line for following tfs -->
  <xsl:if test="parent::FSLIST and following-sibling::FS">
    <xsl:value-of select="$newline"/>
  </xsl:if>
</xsl:template>
```

Recursive markup: Stylesheet

Part 3/4: F match

```
<xsl:template match="F"> <!-- match feature-value pairs -->
  <xsl:param name="indent"/> <!-- # of spaces to insert -->
  <!-- indent unless we are in the first feature -->
  <xsl:if test="preceding-sibling::F">
    <xsl:call-template name="repeat">
      <xsl:with-param name="times" select="$indent"/>
    </xsl:call-template>
  </xsl:if>
  <xsl:value-of select="concat(@name, ' ')"/> <!-- print F name -->
  <!-- recursively print value -->
  <!-- indentation is string-length + 1 (+1 for space char) -->
  <xsl:apply-templates select="FS">
    <xsl:with-param name="indent"
                    select="string-length(@name) + $indent + 1"/>
  </xsl:apply-templates>
  <!-- insert comma and newline unless we are in last feature -->
  <xsl:if test="following-sibling::F">
    <xsl:value-of select="concat(',', $newline)"/>
  </xsl:if>
</xsl:template>
```

Recursive markup: Stylesheet

Part 4/4: auxiliary template

```
<!-- auxiliary subroutine to print $char $times times -->
<xsl:template name="repeat">
  <xsl:param name="times"/>  <!-- number of chars to repeat -->
  <xsl:param name="char" select="' '"/> <!-- char to repeat -->

  <!-- stop recursion if $times = 0 -->
  <xsl:if test="$times > 0">
    <!-- print single character -->
    <xsl:value-of select="$char"/>
    <!-- tail recursion -->
    <xsl:call-template name="repeat">
      <xsl:with-param name="times" select="$times - 1"/>
      <xsl:with-param name="char" select="$char"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

Outline

xsl:include and xsl:import

Include definitions from another, external stylesheet file.
Syntax: only allowed at top-level of stylesheet files (at same level as xsl:template)
Two variants:

- `xsl:include`: local definitions take precedence over external definitions
- `xsl:import`: all stylesheets are treated equal (macro-like import)
- `xsl:apply-imports`: similar to `xsl:apply-templates`: apply templates imported with `xsl:import` at position where `xsl:apply-imports` occurs

Application: Modularization of stylesheets, re-use of templates

Outline

Merging multiple XML files using document()

`document()` as initial path component includes external document.

Example: merge NLP standoff markup

Part 1/3: Stylesheet header

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

  <!-- Combine multiple XML input documents into one document, -->
  <!-- the parameter filelist should countain a comma-separated -->
  <!-- list of annotation file names -->

  <xsl:output method="xml" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:param name="filelist">file1.xml,file2.xml</xsl:param>

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

Merging multiple XML files using document()

Part 2/3: global template

```
<xsl:template match="/">
  <!-- insert new root element <combined> -->
  <xsl:element name="combined">
    <!-- deep copy of input file -->
    <xsl:copy-of select="."/>
    <!-- insert files specified in filelist -->
    <xsl:call-template name="insert-documents">
      <xsl:with-param name="filelist" select="$filelist"/>
    </xsl:call-template>
  </xsl:element>
  <xsl:apply-templates/>
</xsl:template>
```

Merging multiple XML files using document()

Part 3/3: recursive template

```
  <xsl:template name="insert-documents">
    <xsl:param name="filelist"/>
    <xsl:choose>
      <xsl:when test="contains($filelist,',')">
        <xsl:variable name="first" select="substring-before($filelist,',')"/>
        <xsl:copy-of select="document($first)"/>
        <xsl:call-template name="insert-documents">
          <xsl:with-param name="filelist" select="substring-after($filelist,',')"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy-of select="document($filelist)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Outline

Indexing XML documents with xsl:key

`xsl:key` (in stylesheet preamble) generates a unique key named as specified as value of the attribute `name` (like in databases) based on the element specified as value of the attribute `match` and the values specified in the attribute `use`.
Intuitive example: address book index using surname

Example for xsl:key (RDF)

```
<xsl:key  name="aboutkeys"
         match="rdf:Description"
           use="@rdf:about"/>
```

Index lookup with XSLT-specific XPath function key()

`key(name,value)`

returns node(s) with key name (first arg) and value (second arg)
(pre-indexing with `xsl:key` required as shown on previous slide)

XSLT-specific XPath functions: generate-id()

`generate-id(node)` generates unique IDs for nodes
example: generating unique RDF statements

Sorting and Grouping RDF statements

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!-- Description: sorts and groups rdf:Description elements with
                    same rdf:about values -->
  <!-- Prerequisites: Sesame forward-chaining inferences for full
                      chains of rdfs:subClassOf relations -->
  <!-- Input DTD: RDF (as produced by Sesame) -->
  <!-- Output: RDF (with grouped Descriptions) -->

  <xsl:output method="xml" indent="yes"/>
  <xsl:key name="aboutkeys" match="rdf:Description" use="@rdf:about"/>

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

XSLT-specific XPath functions: generate-id()

Sorting and Grouping RDF statements

```
<xsl:template match="/rdf:RDF">
  <xsl:copy>
    <xsl:copy-of select="@*"/> <!-- copy namespace and other attributes -->
    <!-- walk through rdf:Description elements with rdf:about attributes -->
    <xsl:for-each select="rdf:Description[generate-id(.) =
                          generate-id(key('aboutkeys', @rdf:about)[1])]">
      <xsl:sort select="@rdf:about"/>
      <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:for-each select="key('aboutkeys', @rdf:about)">
          <xsl:copy-of select="*"/>
        </xsl:for-each>
      </xsl:copy>
    </xsl:for-each>
  </xsl:copy>
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

XSLT-specific XPath functions: system-property()

Every XSLT processor is obliged to define at least the following three system properties.

systemprop.xsl

```
1 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
2   <xsl:strip-space elements="*"/>
3   <xsl:output method="text"/>
4   <xsl:template match="/">
5     Supported XSL version: <xsl:value-of select="system-property('xsl:version')"/>
6     Vendor Name: <xsl:value-of select="system-property('xsl:vendor')"/>
7     Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')"/>
8   </xsl:template>
9 </xsl:stylesheet>
```

```
Supported XSL version: 1.0
Vendor Name: libxslt
Vendor URL: http://xmlsoft.org/XSLT/
```

xsltxt: XSLT without XML syntax

https://xsltxt.dev.java.net

For those allergic to angle brackets

xsltxt syntax

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

xsltxt: XSLT without XML syntax

Equivalent XSLT syntax

```
<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>
```

Outline

Java: Xalan via command line

Usage:

```
java XalanProcess -IN xmlfile -XSL xslfile -OUT outputfile
```

Xalan

```
1  // workaround for missing main method in xslt.Process in JDK 1.5
2  // In JDK 1.4, XSLT transformations could be called from the command line:
3  // java org.apache.xalan.xslt.Process -IN in.xml -XSL stylesheet.xsl -OUT out.xml
4  // In JDK 1.5, it was renamed to com.sun.org.apache.xalan.internal.xslt.Process
5  // and the main method has been renamed to _main, i.e. it is no longer
6  // callable from the command line
7  // cf. http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=5099865
8  // This class is a workaround, it makes a main method available from cmd line
9  // invoke as: // java xalanProcess -in file.xml -xsl file.xsl
10 public class xalanProcess {
11   public static void main(String[] args)
12   {
13      com.sun.org.apache.xalan.internal.xslt.Process._main(args);
14   }
15 }
```

Java

JAXP example

```
1  import java.io.StringWriter;
2  import javax.xml.transform.Transformer;
3  import javax.xml.transform.TransformerFactory;
4  import javax.xml.transform.stream.StreamResult;
5  import javax.xml.transform.stream.StreamSource;
6  public class TransformerTest {
7    public static void main(String[] args) {
8      try { // args[0]=stylesheet filename, args[1]=xml input filename
9        TransformerFactory tFactory = TransformerFactory.newInstance();
10       StreamSource stylesheet = new StreamSource(args[0]);
11       Transformer transformer = tFactory.newTransformer(stylesheet);
12       StringWriter resultStringWriter = new StringWriter();
13       StreamSource source = new StreamSource(args[1]);
14       StreamResult result = new StreamResult(resultStringWriter);
15       transformer.transform(source, result);
16       System.out.print(resultStringWriter.toString());
17     } catch (Exception e) {
18       System.err.print(e.toString());
19     }
20   }
21 }
```

Python

(tested with Python 2.5; lxml binding required)

Python libxslt example

```
1 from lxml import etree
2 from StringIO import StringIO
3
4 xmldoc = etree.parse(StringIO("<text>Inline XML document</text>"))
5 transformer = etree.XSLT(etree.parse("stylesheet.xsl"))
6 utf8result = str(transformer(xmldoc, param1=u"'value'"))
7 print utf8result
```

More examples for online NLP integration

http://heartofgold.dfki.deHeart of Gold: XML-(XSLT-)based integration middleware for NLP components

- typical (pipelined) worksflows: shallow pre-processing for deep parser (HPSG)
- robustness (default lexicon entries generated for unknown words guessed by PoS tagger and named entity recognition components
- division of labour between general linguistic modelling and semantics output by deep grammar
- rapid integration of different NLP tools for multiple languages for experimantation
- rapid development of new NLP-based systems (Question Answering, Information Extraction etc.)
- many XSLT examples in xsl/ subdirectory of source tarball

Outline

Alternatives to XSLT

For simply structured XML and simple queries:

- Apache Lucene: Very fast indexing of large simply-tagged XML data sets, customizable schema Easy-to-use interfaces (Java, Python etc.)
- Python ElementTree: Transforming and Accessing XML easily with Python ElementTree
- ...

Otherwise... use XSLT. It's very efficient, portable, standardized.

Outline

XSLT 2.0

- W3C recommendation
- mostly compatible with version 1.0
- based on XPath 2.0
- multiple document output
- user-definable XPath functions ('first class' functions)

Books:

- Michael Kay: XSLT 2.0, 3rd edition
- Sal Mangano: XSLT Cookbook, 2nd edition (covers XSLT 1.0 and 2.0; O'Reilly)

Outline

XPath 2.0

- W3C recommendation
- more than 80 predefined functions
- if-then-else (e.g. `select="if ($x gt 3) then 'big' else 'small'"`)
- sequence data type (similar to Python: `1 to 50, ((1,2,3),4,(5,6))`)
- quantifiers: `some, every, in, satisfies`
- user-definable XML Schema-based data types
- regular expressions for strings and even path matches

XQuery 1.0

- query language mainly intended as interface to XML databases
- based on XPath 2.0 (largely shared with XSLT 2.0)
- non-XML syntax of the query language
- non-XML syntax for user-definable functions

XPath vs. XSLT vs. XQuery

- XPath 1.0: path expressions, comparison and computation, some built-in functions
- XPath 2.0: conditional, arithmetic, quantified expressions, built-in functions and operators, data model
- XSLT 2.0: stylesheets (programs), templates, literal result elements, user-definable functions
- XQuery 1.0: FLWOR expressions, Query prolog, XML constructors, user-definable functions

XQuery Syntax

- Query Prolog: declarations of namespaces, variables, function definitions, processing instructions
- FLWOR (pronounced 'flower'): the query (for-let-where-order-by-return clause)

Xquery example without FLWOR clause

```
1 declare boundary-space preserve;
2 declare namespace nlp = "http://www.purl.org/nlp";
3 declare variable $annotation := doc("input.xml")/sentences;
4
5 count($annotation/sentence)
```

XQuery Syntax: FLWOR

Example with baseX factbook sample database:

Xquery example with FLWOR clause

```
1 declare boundary-space preserve;
2 declare variable $factbook := doc("factbook.xml")/mondial;
3
4 for $country in $factbook/country
5     order by number($country/@population)
6     return $country/name/text()
```

Further reading

- Standard: W3C recommendation
- Many examples can be found in the XQuery Wikibook
- Use baseX to interactively develop and test XQueries.
- Book: Priscilla Walmsley: XQuery (O'Reilly)

</slides>