

# Programmierpraktikum

Sommersemester 2007 - 18.05.2007

Björn Wilmsmann, B.A.  
Sprachwissenschaftliches Institut  
Ruhr-Universität Bochum  
[wilmsmann@linguistics.rub.de](mailto:wilmsmann@linguistics.rub.de)

# Heute

- jede Menge Akronyme...

# XML

- Motivation
  - linguistische Corpora liegen oftmals nicht als ‚plain text‘ vor, sondern sind mit zusätzlichen Informationen (POS Tags, Syntax etc.) versehen
  - Anwendungen, die Informationen mit der Außenwelt austauschen, müssen üblicherweise Markupssprachen verstehen

# XML

- Beispiele
  - TüPP-D/Z (taz Corpus)
  - TIGER (FR Corpus)
  - Verarbeitung von Webressourcen als Corpus (z.B. Wikipedia, aber auch beliebige andere)

# XML

- Anwendungen
- Information Retrieval: Vor der eigentlichen Analyse muss ein Dokument zunächst geholt und geparkt werden
- textuelle Informationen müssen fürs Web mit HTML aufbereitet werden

# XML

- Anwendungen
  - Semantic Web (RDF, OWL)
  - Blogs, Newsfeeds (RSS)

# Markupsprachen

- Präsentations-Markup: Zeilenumbrüche, Leerzeilen etc.
- Prozedurales Markup: TeX, teilweise HTML
- Deskriptives Markup: RDF, OWL, teilweise HTML

# Geschichtliches

- SGML
- HTML, CSS
- XML, DTD
- XHTML, MathML, GraphML, SVG usw.
- XSD, XSL usw.

# SGML

- Standardized Generalized Markup Language
- ging aus der GML (Generalized Markup Language) hervor
- GML wurde 1960 bei IBM von Goldfarb, Mosher und Lorie entwickelt
- dient(e) der Erstellung von spezialisierten Markup-Sprachen wie z.B. HTML

# HTML

- HyperText Markup Language
- SGML-Sprache, erster Entwurf 1993 von Tim Berners-Lee et.al.
- Präsentation von Text (bzw. später auch audiovisuellen Medien) im WWW
- Verknüpfung zwischen Ressourcen

# HTML

- wird seit 1995 vom W3C (World Wide Web Consortium) standardisiert
- existierte in verschiedenen Versionen
  - HTML 2.0 bis aktuell HTML 4.01 (letzteres in den Flavours strict, transitional und frameset)
  - aktuelles Draft: HTML 5.0

# CSS

- Cascading Stylesheets
- definiert die Präsentation eines HTML-, XHTML- oder XML-Dokumentes
- Motivation
  - Verlagerung von Präsentationscode in eine zentrale Datei
  - HTML-Code idealerweise rein deskriptiv

# CSS

```
.greentext {  
  color: #129f12;  
}  
  
.maintext-bold {  
  font-weight: bold;  
  vertical-align: top;  
}  
  
.maintext {  
  vertical-align: top;  
}  
  
h1 {  
  text-decoration: none;  
}
```

# XML

- eXtensible Markup Language
- Weiterentwicklung der SGML
- vereinfachte Untermenge der SGML
- dient der Entwicklung spezialisierter Markup-Sprachen und Datenformate

# XML

- XML-Dokumente müssen zwei Voraussetzungen erfüllen
  - Wohlgeformtheit: Syntaktisch korrekt
  - Validität: Semantisch definiert über XML Schema (XSD) oder Document Type Definition (DTD)

# DTD

- kontextfreie Grammatik für die Semantik eines XML-Datentyps/-Dokuments
- definiert
  - welche Attribute Elemente eines Datentyps haben
  - welche Unterelemente ein Element hat

# Formale Grammatik

- Grammatik
  - Startsymbol
  - Terminalsymbole
  - Non-Terminalsymbole
  - Produktionsregeln

# Chomsky-Hierarchie

- Typ 0
  - Nicht restringierte Regelgrammatiken
  - erzeugen alle Turing-vollständigen Sprachen
  - rekursiv aufzählbar

# Chomsky-Hierarchie

- Typ I
  - Kontextsensitive Grammatiken
  - Vorkommen von Symbolen hängt vom Auftreten von (Non-)Terminalsymbolen ab
  - $\alpha A \beta \rightarrow \alpha \gamma \beta$  mit  $A$  Non-Terminalsymbol und  $\alpha, \gamma, \beta$  (Non-)Terminalsymbole

# Chomsky-Hierarchie

- Typ 2
  - Kontextfreie Grammatiken
  - Vorkommen von Symbolen ist frei
  - $A \rightarrow \gamma$  mit  $A$  Non-Terminalsymbol und  $\gamma$  (Non-)Terminalsymbol

# Chomsky-Hierarchie

- Typ3
  - Reguläre Grammatiken (RegEx)
  - Umsetzung einzelner Non-Terminalsymbole in Terminalsymbole (plus optionales Non-Terminalsymbol)
  - $\langle A \rangle ::= [B]\alpha$  mit A, B Non-Terminalsymbol und  $\alpha$  Terminalsymbol

# DTD

```
<!-- entity definitions -->
<!ENTITY % def.ngram.content 'token,frequency'>

<!-- elements -->
<!ELEMENT analysis (result|error)>
<!ELEMENT result (tenMostFrequentWords,mostFrequentWordsAll,keywords)>
<!ELEMENT tenMostFrequentWords (word)+>
<!ELEMENT mostFrequentWordsAll (word)+>
<!ELEMENT word (%def.ngram.content)>
<!ELEMENT keywords (keyword)+>
<!ELEMENT keyword (token)>
<!ELEMENT token (#PCDATA)>
<!ELEMENT frequency (#CDATA)>
<!ELEMENT error (message,url)>
<!ELEMENT message (#PCDATA)>
<!ELEMENT url (#PCDATA)>
```

# XHTML

- Re-write von HTML mittels XML
- HTML besaß ursprünglich keine definierte Grammatik, sondern war anfangs mehr eine Quick-and-Dirty-Lösung
- SGML- bzw. HTML-Parser wie z.B. Browser sind nachsichtig, verzeihen Syntax-Fehler

# XHTML

- XML (und damit XHTML) muss wohlgeformt sein
- ist daher für Parser vorhersehbar
- erleichtert das Parsing von Dokumenten erheblich
- siehe Browser: Korrekte Darstellung von HTML-Code ist Rocket Science!

# XHTML

- XHTML 1.0 war der Nachbau von HTML 4.01 in XML
- existiert ebenfalls in drei Flavours: strict, transitional, frameset
- XHTML 1.1, XHTML 2.0
- XHTML5 wird parallel zum vorgeschlagenen HTML5 entworfen

# XSD

- Anwendung von XML
- Alternative zum (nicht-XML) Standard DTD
- erlaubt im Gegensatz zu DTD typensichere Strukturen

# XSD

```
<xs:complexType name="MyType">
  <xs:sequence>
    <xs:element name="MyChildElement1" type="xs:int" minOccurs="0"/>
    <xs:element name="MyChildElement2" type="xs:string" maxOccurs="10"/>
    <xs:element name="MyChildElement3" type="xs:float"/>
  </xs:sequence>
</xs:complexType>
```

# XSL

- eXtensible Stylesheet Language
- Sprachfamilie, die das Transformieren von einem XML Datentyp in einen anderen ermöglicht
- z.B. XML nach XHTML

# XSL

- XSL Transformations (XSLT)
  - Transformation von XML nach XML
- XSL Formatting Objects (XSL-FO)
  - Visualisierung von XML-Dokumenten
- XML Path Language (XPath)
  - Query-Sprache für XML-Dokumente

# XSLT - Input

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="ND1">
    <name>Nancy</name>
    <family_name>Davolio</family_name>
  </person>
</persons>
```

# XSLT - Transformationsregeln

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

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

<xsl:template match="//person">
    <name username="{@username}">
        <xsl:value-of select="name" />
    </name>
</xsl:template>

</xsl:stylesheet>
```

# XSLT - Output

```
<?xml version="1.0" encoding="UTF-8"?>  
<root>  
  <name username="JS1">John</name>  
  <name username="ND1">Nancy</name>  
</root>
```

# Vorteile von XML

- Menschen- und maschinenlesbar (im Gegensatz zu den Extremen Prosatext und Binär)
- Unicode-Support
- Datenstrukturen
- Selbsterklärende Feldbezeichnungen möglich

# Vorteile von XML

- Stringente Syntax ermöglicht einfaches und effizientes Parsen
- Plain Text, offenes Format
- Industriestandard, weite Anwendungsbereiche
- robust, erprobt

# Nachteile von XML

- häufig redundant
- erhöht möglicherweise Speicher- und Rechenkosten im Vergleich zu kompakteren Formaten
- begrenzt auf ER- oder OO-Konzepte
- Fuzzy Relationen lassen sich nicht ausdrücken

# XML mit Perl

- XML ist ein Textformat
- Perl ist eine Programmiersprache, die zum Verarbeiten von Text entwickelt wurde
- daher passen XML und Perl ideal zusammen

# XML-Verarbeitung mit Perl

- There is more than one way to do it...
- XML::Simple - einfach zu benutzen
- XML::Parser - vollständiger Parser, baut auf der C Bibliothek Expat auf
- XML::XPath - Query in XML-Dokumenten

# XML-Verarbeitung mit Perl

- XML::SAX (SAX: Simple API for XML) - vollständiger Parser
- 2. Generation von XML Parseern
- sprachunabhängig
- definiert lediglich Interfaces
- konkrete Implementierungen, was bei welchem Ereignis passieren soll, werden vom Entwickler selbst geliefert

# XML::Simple

```
# use XML::Simple
use XML::Simple;

# initialise
my $xml = new XMLIn('in.xml', forcearray => 1);

# iterate over items
foreach my $item (@{$xml->{customer}}) {
    print $item;
}
```

# XML::SAX

- XML::Parser und XML::SAX sind bedeutend komplexer als XML::Simple
- nutzen individuelle EventHandler, die vom Objekt während der Laufzeit aufgerufen werden
- Beispiele für EventHandler: `start_element`, `end_element`, `characters`...

# XML::SAX

- der Entwickler erstellt eine Handler-Klasse, welche die vom SAX-Interface vorgegebenen Methoden implementiert, der SAX-Parser erledigt den Rest
- diese Methoden führen dann die gewünschte Funktionalität zum jeweiligen Zeitpunkt aus

# XML::SAX

```
# use SAX parser factory
use XML::SAX::ParserFactory;

# use SAX handler
use XML::SAX::MyHandler;

# initialise
my $handler = new XML::SAX::MyHandler();
my $parser = XML::SAX::ParserFactory->parser(handler => $handler,);

# parse
$p->parse_uri('in.xml');
```

# XML::SAX

```
# package
package MyHandler;

# constructor
sub new {
    my $type = shift;
    return (bless({}, shift));
}

# start element
sub start_element {
}

# end element
sub end_element {
}

# characters
sub characters {
}

...
```

# Aufgabe

- verbessern Sie Ihre bisherigen Lösungen...

# Referenzen

- Ray, Erik T. & McIntosh, Jason (2002): Perl & XML. O'Reilly. Sebastopol, CA, USA
- [http://en.wikipedia.org/wiki/XSL\\_Transformations](http://en.wikipedia.org/wiki/XSL_Transformations)
- <http://www.sgmlsource.com/history/AnnexA.htm>

# Referenzen

- <http://www.tech-know-ware.com/lmx/xsd-overview.html>
- <http://www.w3.org/>
- <http://www.w3schools.com/dtd/default.asp>
- <http://www.w3schools.com/xml/default.asp>
- <http://www.w3schools.com/xpath/default.asp>

# Referenzen

- [http://www.w3schools.com/xsl/xsl\\_languages.asp](http://www.w3schools.com/xsl/xsl_languages.asp)
- <http://www.w3schools.com/xslfo/default.asp>
- <http://www.w3schools.com/xsl/default.asp>

**Vielen Dank für Ihre  
Aufmerksamkeit!**