Sven Naumann

XML als Beschreibungssprache syntaktisch annotierter Korpora

In den letzten Jahren ist die Zahl der verfügbaren linguistisch annotierten Korpora ständig gewachsen. Zu den bekanntesten gehören das Brown-Korpus, das Susanne-Korpus, die Penn-Treebank, das Negra-Korpus, das Tiger-Korpus und die im Zusammenhang mit dem Verbmobil-Projekt entstandenen Korpora.

Neben den bekannten Differenzen in den zur Annotierung verwendeten Vokabularen (morpho-syntaktische Tags und Merkmale, Bezeichner für syntaktische Kategorien und semantische Rollen, etc.) werden Analysen, die Daten aus mehreren dieser Korpora nutzen wollen, vor allem durch die Tatsache erschwert, dass fast jedes dieser Korpora ein eigenes Annotierungsschemata verwendet.

Im Folgenden soll es darum gehen, (a) die Vor- und Nachteile von XML als Korpusbeschreibungssprache anhand eines sich im Aufbau befindenden Korpus zu diskutieren, (b) die für dieses Korpus gewählte XML-Struktur zu beschreiben und (c) die zur Pflege und Entwicklung des Korpus konzipierte Software vorzustellen.

1 Korpusbeschreibungssprachen

Im Zusammenhang mit der quantitativen Analyse syntaktischer Strukturen verschiedener europäischer Sprachen begann vor etwa 20 Monaten an der Universität Trier der Aufbau eines auf Texten der Berliner Tageszeitung TAZ basierenden Korpus. Eine der ersten Entscheidungen, die in diesem Zusammenhang getroffen werden mussten, betraf das für die Rohdaten zu verwendende Annotierungsschema. Genau genommen ging es dabei um zwei Fragen:

(1) Welches Format ist für die Annotierung der Daten zu wählen?
(2) Ist es sinnvoll, Daten und Annotierungen strikt zu trennen und in jeweils separaten Dateien (eine für die Daten und jeweils eine weitere für jede Beschreibungsebene) zu verwalten oder sollten sie gemeinsam in einer Datei gespeichert werden?

1.1 *Tabelle* oder *Baum*

Eine einfache Möglichkeit, die in einem annotierten Korpus enthaltenen Daten zu organisieren, besteht darin, sie durch eine Tabelle zu repräsentieren, in der jede Spalte Informationen eines bestimmten Typs (Wortform/Token, Lemma, Wortart, …) enthält.

Diese Art der Darstellung ist, wie ein Blick in die oben erwähnten Korpora schnell zeigt, sehr verbreitet:

Beispiel (1-1)

(a) Susanna-Korpus

A01:0170g	-	AT	The		the	[O[S[Ns:s.
A01:0170h	-	JJ	grand		grand	.
A01:0170i	-	NN1c	jury		jury	.Ns:s]
A01:0170j	-	VVDi	commented		comment	[Vd.Vd]
A01:0170k	-	II	on		on	[P:u.
A01:0170m	-	AT1	a		a	[Ns.
A01:0170n	-	NN1c	number	number	.	
A01:0180a	-	IO	of		of	[Po.
A01:0180b	-	JBo	other		other	[Np.
A01:0180c	-	NN2	topics		topic	.Np]Po]Ns]P:u]

(b) MALI-Korpus

i	81	i			0010253
za	774	za			0010254
projev	117214	projev	41_06		0010255
solidarity	110312	solidarita	31_01		0010256
z	772	z			0010257
vietnamsk,	22 312	vietnamskì	31+01		0010258
strany	117312	strana	31_03		0010259
byli	5255 21	bìti	22	912	0010260
jste			_01		0010261

(c) Negra-Korpus

die	ART	Def.Fem.Nom.Sg	NK	507
Zukunft	NN	Fem.Nom.Sg.*	NK	507
der	ART	Def.Fem.Gen.Sg	NK	502
Musik	NN	Fem.Gen.Sg.*	NK	502
liegt	VVFIN	3.Sg.Pres.Ind	HD	509
für	APPR	Akk	AC	503
viele	PIDAT	*.Akk.Pl	NK	503
junge	ADJA	Pos.*.Akk.Pl.St	NK	503
Komponisten	NN	Masc.Akk.Pl.*	NK	503
im	APPRART	Dat.Masc	AC	504
Crossover-Stil	NN	Masc.Dat.Sg.*	NK	504
.	$.	--	--	0

Die Vorteile dieser Art der Datenrepräsentation liegen auf der Hand:

- Sie sichert eine gute Lesarbeit und ermöglicht damit auch prinzipiell direkte manuelle Korrekturen.
- Sie ist sehr kompakt.
- Da es sich um einfache Textdateien handelt, können sie prinzipiell mit jedem Editor bearbeitet werden.

Allerdings zeigt dieses Beispiel auch, das sich mit der Wahl einer tabellarischen Struktur die Gemeinsamkeiten in der Datenrepräsentation erschöpfen: Jedes Korpus verwendet sein eigenes, proprietäres Format, das sich sowohl in Form (Spaltenzahl, Spaltentrenner, etc.) als auch im Inhalt[1] von allen übrigen unterscheidet. Daraus folgt, dass die zur Validierung, zur Visualisierung, zum Datenaustausch mit anderen Systemen und zur Datenextraktion erforderlichen Softwarewerkzeuge an die spezifische Struktur angepasst bzw. vollständig neu geschrieben werden müssen.

XML ist eine inzwischen weit verbreitete SGML-basierte (Meta-)Markupsprache , die es anders als HTML ermöglicht, die Menge der für das Markup verwendeten Tags und Attribute frei zu definieren.

XML-Dokumente sind, was ihre Struktur betrifft, Bäume: In jedem Dokument gibt es genau einen Wurzelknoten, von dem aus allen anderen Knoten des Dokuments erreichbar sind und es gibt keine sich kreuzenden Kanten.

Die Attraktivität von XML zur Datenmodellierung erklärt sich aus folgenden Gründen:

- Mit den DTDs und den XML-Schemata sind Mechanismen verfügbar, durch die sehr genau spezifiziert werden kann, welche Bedingungen ein XML-Dokument erfüllen muss, um nicht nur – im Sinne der allgemeine XML-Spezifikation – *wohlgeformt*, sondern auch *gültig* zu sein.
- Software, die die Validität von XML-Dokumenten überprüft (XML-Parser) ist frei verfügbar.
- XML-Dateien können mit jedem Editor, der Unicode unterstützt, bearbeitet werden.
- Einige Browser (wie z.B. Microsofts Internet-Explorer) sind in der Lage, XML-Dokumente direkt darzustellen.
- Mit XSLT steht ein mächtiges Instrument zur Konvertierung von XML-Dokumenten in andere Formate (HTML, PDF, …) zur Verfügung. Auf diese Weise ist es einfach, selektiv auf Daten zuzugreifen, sie zu exportieren oder zu visualisieren (SVG).

Auf der anderen Seite ist XML kein sehr kompaktes Datenrepräsentationsformat: Eine explizite Darstellung der in den tabellarisch organisierten Daten der Korpora aus Bei-

[1] So enthält z.B. nur das Negra-Korpus neben den morpho-syntaktischen Tags weitere Merkmale, verzichtet aber die Angaben von Grundformen. Syntaktische Strukturen werden im Susanna-Korpus durch Klammerstrukturen, im Mali-Korpus dagegen durch numerisch repräsentierte Pointer dargestellt.

spiel (1-1) implizit enthaltenen Struktur durch ein XML-Dokument lässt die Größe der Korpusdatei deutlich wachsen. Außerdem sind komplexe XML-Strukturen – selbst wenn die Einbettungstiefe durch Einrückung visualisiert wird – für menschliche Betrachter nur schwer lesbar.

Einen für die Erstellung syntaktisch annotierter Korpora auf den ersten Blick gravierenden Nachteil bildet die Baumstruktur von XML-Dokumenten: Da sich Kanten nicht schneiden dürfen[2], ist eine direkte Abbildung diskontinuierlicher Strukturen, etwa durch einfache Grafen wie sie im Negra-Korpus verwendet werden, nicht möglich.

1.2 *Einsam* oder *Gemeinsam*

Eine distribuierte Repräsentation von Korpora[3], in der Daten und Annotationen getrennt verwaltet werden, ist solange nicht möglich, wie es an einen Referenzmechanismus fehlt, der es ermöglicht, die systematische Beziehung zwischen den Entitäten der Daten- und Annotationsebenen zu modellieren. Mit XLINK und XPOINTER besitzt XML einen derartigen Mechanismus und insofern ist es nur konsequent, dass mit XCES eine XML-Instanz des *Corpus Encoding Standards* CES entwickelt wurde, der diese Idee konsequent umsetzt (vgl. (Ide und Romary 2000) und (Ide und Romary 2001)).

Außer strukturellen Tags, die Satzgrenzen und die Grenzen anderer Texteinheiten markieren, enthält die Korpusdatei keine weiteren Annotationen. Die Annotationsdateien können auf einzelne Sätze der Korpusdatei mit Hilfe eines Identitätsattribut des Satzgrenzen kennzeichnenden Tags referieren und die Token eines Satzes lassen sich dann durch die ***substring***-Funktion ansprechen, die auf dem Textknoten operiert, der den ganzen Satz enthält:

Beispiel (1-2)

(a) Korpusdatei

```
…
<satz id="s43">
  Er wurde 1981 von Mobutu inhaftiert und erst nach einer internationalen
  Kampagne wieder freigelassen.
</satz>
```

(b) morpho-syntaktische Annotation

```
…
```

[2] Anders formuliert: Ein XML-Tag darf nicht geschlossen werden, wenn nicht alle der in ihn eingebetteten Tags geschlossen wurden.

[3] In der Literatur oft auch als ‚*Stand-off* Annotation' bezeichnet (vgl. (Lezius 2002), S. 22).

```
    <satz xlink:href="xptr(substring(//satz[43]))">
        <token      id="t1" xlink:href="xptr(substring(//satz[43]/text(),1,2)"
                    wclass="PPER" lemma="er">
        <token      id="t2" xlink:href="xptr(substring(//satz[43]/text(),4,8)"
                    wclass="VAFIN" lemma="sein">
        …
    </satz>
```

(c) syntaktische Annotation

```
    …
    <struktur id="s0" cat="S">
        <struktur id="s1" cat="NP"
                    xlink:href="xptr(substring(//satz[43]/text(),1,2)/>
        <struktur id="s2" cat="VP"
                    xlink:href="xptr(substring(//satz[43]/text(),4,101)/>
        …
    </struktur>
```

Der große Vorteil einer distribuierten Repräsentation liegt darin, dass nun (zumindest prinzipiell) Änderungen auf einer Beschreibungsebene keine direkten Auswirkungen auf die Annotationen der anderen Ebenen haben. Es ist daher zu erwarten, dass dieser Ansatz zunehmend beim Aufbau neuer Korpora Verwendung findet.

2 Das Trierer TAZ-Korpus

Die Erstellung von großen Mengen von syntaktisch annotierten Sprachdaten, wie sie für quantitative Untersuchungen benötigt werden, ist trotz z. T. frei verfügbarer Tools wie Tokenizier, Tagger und Parser ein zeit- und arbeitsintensiver Prozess; denn um die Fehlerquote in tolerablen Grenzen zu halten, ist es erforderlich, über eine angemessene Menge von bereits vollständig und korrekt annotierten Daten (Trainingskorpus) zu verfügen, anhand der diese Tools trainiert werden können (*supervised learning*). Ein Verzicht auf das Trainieren von Tagger und Parser bzw. die Verwendung von anhand anderer Korpora trainierter Tagger und Parser scheint wenig sinnvoll, da bereits eine geringe Zahl von Annotierungsfehlern die Qualität von den auf diesen Annotierungen basierenden Analysen erheblich beeinträchtigen können: So verschlechtert sich z.B. durch Fehler bei der Identifikation von Abkürzungen die Quote der korrekt als Satzendemarkierungen identifizierten Punkte deutlich (vgl. Mikheev (2002)) und Taggingfehler blockieren häufig korrekte syntaktische Analysen. Man steht also vor der auf

den ersten Blick paradox erscheinenden Situation, über annotierte Daten verfügen zu müssen, bevor man Daten mit hoher Genauigkeit semiautomatisch annotieren kann.

Diese Ausgangssituation führt dazu, dass bei der Entwicklung (homogener) Korpora wie folgt verfahren wird:

- Es wird eine geringe Menge von Texten ausgewählt, die mit nicht an diese Daten adaptierten Tools analysiert und annotiert wird.
- Nach umfangreichen manuellen Korrekturen dieser Daten werden sie dazu verwendet, um Tagger und Parser zu trainieren, die dann anschließend verwendet werden können, um weitere Texte aus derselben Quelle mit hoher Genauigkeit zu annotieren.

Diesen Vorgehen folgend wurden aus den knapp 600 000 Artikel der elektronischen Edition der Jahrgänge 1986 bis 1998 der Berliner Tageszeitung **TAZ** 200 Artikel aus verschiedenen Ressorts (s. Abbildung (1)) als Ausgangskorpus ausgewählt. Aus methodologischen Gründen handelte es sich bei den ausgewählten Texten um solche, die das Suchmodul der TAZ-Datenbank der Kategorie **[lang]** zuordnet (Textlänge > 8250 Zeichen). Das auf diese Weise gebildete Rohkorpus besteht aus 21296 Sätzen und hat eine Größe von 2,5 MB. Die Aufbereitung der Rohdaten erfolgte in mehreren Schritten:

1. Zunächst wurden mit Hilfe eines einfachen Tokenizers (a) Satzgrenzen, Abkürzungen und häufige Eigennamen identifiziert; (b) Interpunktionszeichen klassifiziert und (c) Tags zur Markierung elementarer Textstrukturen eingeführt. Die Eigennamenerkennung stützte sich auf eine 10.000 Einträge umfassende Liste von gebräuchlichen Vor- und Eigennamen. Die Bestimmung der Satzgrenzen und die Disambiguierung der Interpunktionszeichen erfolgten anhand einer kleinen Zahl von Heuristiken. Die nicht sehr hohe Erfolgsquote (sie lag je nach Text zwischen 65% und 81%) dieser Heuristiken ist zumindest partiell durch die Qualität der Rohdaten zu erklären. Wie das Satzende markiert ist, scheint bei der TAZ stark von den persönlichen Vorlieben der einzelnen Autoren abzuhängen: Oft fehlen satzfinale Interpunktionszeichen oder werden durch eine Leerzeile substituiert. Allerdings darf umgekehrt nicht einfach jede Leerzeile als Satzendemarkierung interpretiert werden.
2. Anschließend wurden die Texte mit dem Stuttgarter ***TreeTagger*** (Schmid (1994) und Schmid (1995)) getaggt. Dieser Tagger wurde verwendet, da er (i). frei verfügbar ist; (ii). die publizierten Resultate einen geringen Nachbereitungsaufwand wahrscheinlich schienen ließen; (iii). die Token nicht nur getaggt, sondern auch lemmatisiert werden und (iv). der Tagger das weit verbreitete Stuttgarter-Tübinger-Tagset (STTS) verwendet und so komparative Untersuchungen erleichtert werden.

3. Abschließend wurden zunächst die Nominal- und Präpositionalphrasen im Korpus mit einem nicht-probabilistischen Parser analysiert[4].

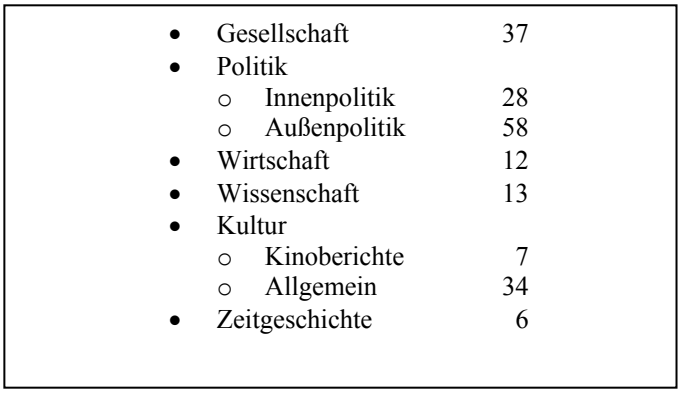


Abb. 1: Textauswahl.

Da das TAZ-Korpus sehr einfach strukturiert ist, wurde zunächst auf die Verwendung einer distribuierten Repräsentation, wie sie durch XCES nahe gelegt wird, verzichtet[5]. Es gilt:

1. Jede Korpusdatei besteht aus einer Folge von Artikeln.
2. Jeder Artikel besteht aus einem *Header*, der Angaben zum Copyright, den Autoren, dem Veröffentlichungsort und Erscheinungsdatum, etc. enthält, sowie dem Text selbst.
3. Der Text besteht aus einer Folge von Sätzen.
4. Jeder Satz wiederum ist eine Folge von Token.

Von einigen Details abgesehen, kann die Struktur des Korpus durch folgende Syntax beschrieben werden:

KORPUS	→	<corpus> ARTIKEL^{+} </corpus>
ARTIKEL	→	<article> HEADER TEXT </article>
HEADER	→	<header> …[6] </header>

[4] Die weitere syntaktische Analyse des Korpus ist zurzeit in Arbeit.
[5] Es ist geplant, zu einem späteren Zeitpunkt durch Verwendung eines XSL-Skripts das Korpus in dieses Format zu konvertieren.

TEXT	→	<text> SATZ^{+} </text>
SATZ	→	<clause> TOKEN^{+} </clause>
TOKEN	→	<token> *token*[7] </token>

Beispiel (2-1)
Die morpho-syntaktischen Annotationen für den Satz aus Beispiel (1-2) werden durch folgende Struktur repräsentiert:

```
<clause>
 <syntax cat="NP" position="(0 1)">
        <syntax cat="PPER">
                <token wclass="PPER" lemma="er">Er</token>
        </syntax>
 </syntax>
 …
 <syntax cat="PP" position="(8 12)">
        <syntax cat="P">
                <token wclass="APPR" lemma="nach">nach</token>
        </syntax>
        <syntax cat="NP">
                <syntax cat="DET">
                        <token wclass="ART" lemma="ein">einer</token>
                </syntax>
                <syntax cat="N1">
                        <syntax cat="A">
          <token wclass="ADJA" lemma="international">internationalen</token>
                        </syntax>
                        <syntax cat="N">
                                <token wclass="NN" lemma="Kampagne">
                                        Kampagne
                                </token>
                        </syntax>
                </syntax>
        </syntax>
 </syntax>
 …
</clause>
```

Diskontinuierliche Strukturen, wie sie sich im Bereich der Syntax fast aller natürlicher Sprachen finden lassen, werden in den meisten der heute verwendeten *constraint-basierten* Grammatikformalismen durch Strukturbeschreibungen repräsentiert, in de-

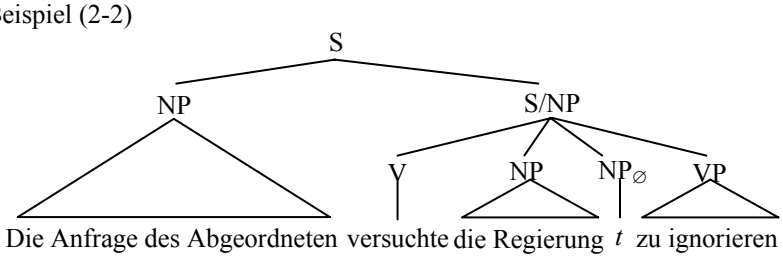
---

[6] Auf eine genaue Spezifikation der Tags zur Repräsentation der *Header*-Informationen kann hier verzichtet werden.

[7] Bei den *Token* handelt es sich um die Wortformen, Interpunktionszeichen, etc., aus denen die Rohdaten bestehen. Aus der Sicht von XML handelt es sich bei ihnen um Objekte vom Typ PCDATA.

nen via Unifikation Merkmale dislozierter Konstituenten ihren syntaktischen Mitspielern verfügbar gemacht werden (*structure sharing*). In XML können solche nichtlokalen Bezüge durch Verwendung zusätzlicher Attribute innerhalb des SYNTAX-Tags vom Typ ID und IDREF bzw. IDREFS nachgebildet werden.

Beispiel (2-2)



NP1 : <syntax position="(0 4)" cat="NP" identifier="@1">
$NP_{\varnothing}$: <syntax … source="@1">

3 Korpus-Software

Das zur Pflege und Entwicklung des Korpus konzipierte System besteht aus vier Modulen:

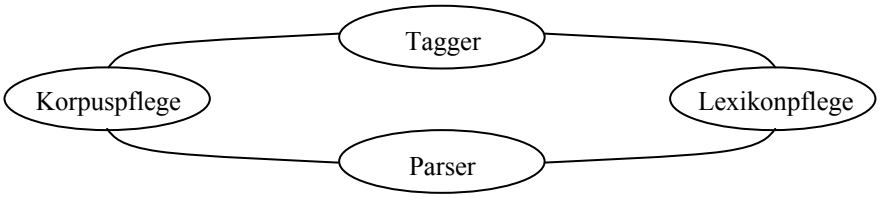


Abb. 2: Module des Systems.

Sie verwenden weitgehend Algorithmen, die auch in anderen, ähnlich konzipierten Systemen Verwendung finden. Der entscheidende Unterschied zu ihnen liegt in der Verwendung von XML als Korpusbeschreibungssprache und die Nutzung der damit verbundenen Möglichkeiten der Datenaufbereitung und Auswertung.

Eine Adaption des Systems an andere Sprachen als das Deutsche ist grundsätzlich möglich, setzt aber die Verfügbarkeit geeigneter Lexika und annotierter Trainingsdaten voraus.

3.1 Morphosyntaktische Analyse

Die morphosyntaktische Analyse kann als ein Prozess betrachtet werden, der folgende drei Phasen durchläuft:

1. **Tokenerkennung**
   Da die Tag-Disambiguierung (s.u.) auf Sätzen operiert, identifiziert der *Tokenizer* mittels Disambiguierung der potentiellen Satzzeichen Satzgrenzen und annotiert gleichzeitig erkannte Abkürzungen.
2. **Annotierung**
   Nun wird versucht, jedem lexikalischen Token ein bzw. mehrere Lesarten zuzuweisen. Findet sich für ein Token in keinem der konsultierten Lexika (Eigennamen-, Idiom-, Lemma- und Wortformenlexikon) ein Eintrag und liefert auch keine der verfügbaren Heuristiken (*morphologische Experte*) eine Interpretation, wird ihm die Menge aller für offene Wortklassen verfügbaren STTS-Tags zugewiesen.
3. **Tag-Disambiguierung**
   Die Auswahl eines Tags aus der Menge der potentiellen Tags eines Tokens erfolgt durch einen HMM-basierten Tagger, der Bi- und Trigramme miteinander verrechnet (*deleted interpolation*), mit Hilfe des Viterbi-Algorithmus (Verweis).

Wie erste Tests gezeigt haben, liefert der Tagger für Systeme dieses Typs übliche *Precision*-Werte (≥ 95%).

3.2 Syntaktische Analyse

Die syntaktische Analyse eines Textkorpus ist eine komplexe Aufgabe, zu deren Lösung sich – wenn kein geeigneter probabilistischer Parser zur Verfügung steht – ganz unterschiedliche Strategien verfolgen lassen. Ordnet man diese nach dem Umfang in dem sie den Einsatz menschlicher Experten erfordern, so ergibt sich ein Spektrum, das durch die folgenden beiden Ansätze begrenzt wird:

- Das Korpus wird vollständig manuell – eventuell unter Verwendung grafischer Annotierungstools – verarbeitet.
- Das Korpus wird mit einem nicht-probabilistischen Parser geparst und die Strukturbeschreibungen anschließend manuell korrigiert.

Das fundamentale Problem des zweiten Verfahrens, das aus arbeitsökonomischen Gründen sicher vorzuziehen ist, bildet die Syntax, die der Parser verwendet. Selbst wenn man dem weit verbreiteten Trend zur Postulierung flacher Strukturen (s. z.B. die Strukturbeschreibungen in der Penn-Treebank und dem Negra-Korpus) folgt, so ist es kaum möglich, eine Syntax zu entwickeln, die allen Sätzen eines größeren Korpus syntaktische Strukturen zuweist, die linguistisch plausibel erscheinen. Andererseits steigt der Ambiguitätsgrad von Sätzen abhängig von ihrer Länge und der Größe der

Syntax exponentiell. Anders formuliert: Je höher der Abdeckungsgrad einer Syntax, desto höher ist auch der für die strukturelle Disambiguierung erforderliche Aufwand.

Um diese Probleme zu reduzieren, verwendet das Parsingmodul eine Strategie, die als „sequentielles Parsen mit lokalen Syntaxen“ bezeichnet werden kann: Statt einer großen globalen Syntax mit mehreren hundert Regeln wird eine Vielzahl kleiner Syntaxen, die oft nur Regeln für bestimmte Konstituententypen enthalten, in mehreren Iterationszyklen angewendet. Dieses Vorgehen gleicht stark dem von der *Constraint Grammar* (Karlsson 1990) zur Disambiguierung morphosyntaktischer Lesarten verwendete Verfahren. Die Grundidee besteht darin, zu jedem Zeitpunkt nur die (syntaktischen) Entscheidungen zu treffen, deren Korrektheit mit hoher Sicherheit zu gewährleisten ist.

Ergänzt wird dieser Ansatz durch die aus anderen Bereichen bekannte *longest-match* Heuristik: Wenn die Regeln einer lokalen Syntax eine bereits gebildete Konstituente auf unterschiedliche Weise in eine neue Konstituente einbetten können, dann wird die Regel gewählt, die zu der größten Konstituente führt. Sind die neuen Konstituenten gleich groß, wird die erste Regel gewählt.

Kern des Parsingmoduls bildet ein Earley-basierter Chartparser, der eine inkrementelle Verarbeitung von Sätzen erlaubt und sich in folgenden Details von einem einfachen Earley-Parser unterscheidet:

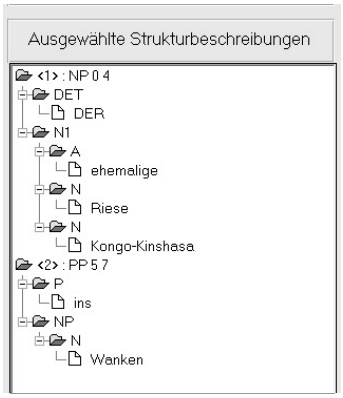
1. Jeder Satz $w = w_1 \ldots w_n$ ($n \geq 1$) wird von links nach rechts verarbeitet. Um zu verhindern, das der Parser terminiert, sobald er auf ein w_i trifft, dass nicht in einen (partiellen) Parse integriert werden kann, wird eine zusätzliche Prozedur verwendet, die neue, bottom-up gebildete Kanten in die Chart einfügen kann: Wenn eine *passive* Kante mit keiner in der Chart gespeicherten *aktiven* Kante kombiniert werden kann, wird für jede Regel, deren linke Ecke (1. Symbol der rechten Regelseite) mit dem Kopf der Kante übereinstimmt, eine neue *aktive* Kante gebildet (*left-corner* Schritt).
2. Außer beim ersten Aufruf des Parsers für einen Satz *w* ist die Chart nicht leer, sondern sie wird mit den dem Parser als Parameter übergebenen Liste von den bereits erkannten Konstituenten (*passive* Kanten) initialisiert.
3. Der Parser generiert eine Menge von (partiellen), sich zum Teil überlappenden Strukturbeschreibungen. Da in dem zur Repräsentation der strukturellen Informationen gewählten XML-Format einem Satz / einer Konstituente nicht mehrere Strukturbeschreibungen zugeordnet werden können und im textuellen Kontext auch nur eine Lesart korrekt ist, wird mit der oben beschriebenen *longest-match* Heuristik eine Folge von maximalen Konstituenten für den Satz ausgewählt.

Die vom Parser verwendeten Parameter und die von ihm generierten Ergebnisse können durch die grafische Oberfläche des Parsermoduls überprüft und modifiziert werden.

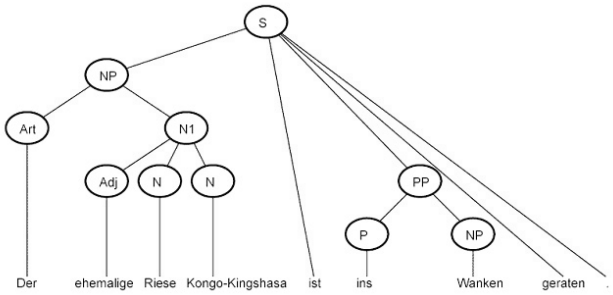
Die vom Parser generierten Strukturen können entweder innerhalb dieser Oberfläche durch so genannte *outline-items* oder mit Hilfe eines XSLT-Skripts via SVG als Bäume repräsentiert werden:

Beispiel (3-1)

(a) syntaktische Strukturen als *outline-items*:



(b) syntaktische Strukturen als SVG-Bäume:[8]



Für die erforderlichen IO-Operationen verwendet das Parsingmodul keinen der frei verfügbaren XML-Parser. Stattdessen wird die zum Parsen geöffnete Korpusdatei vollständig in einen Textbuffer geladen. Eine Validierung der XML-Struktur erfolgt implizit beim Parsen. Obwohl je nach Modus die Daten zwischen bis zu drei Formaten (XML-Strukturen des Korpus/Struktur der Objekte der grafischen Benutzeroberfläche/vom Parser verwendete Datenstruktur) konvertiert werden müssen, ist das Laufzeitverhalten des Sys-

[8] Bei einem partiellen Parse werden alle Token, die nicht Teil einer erkannten Konstituente sind, direkt von S dominiert.

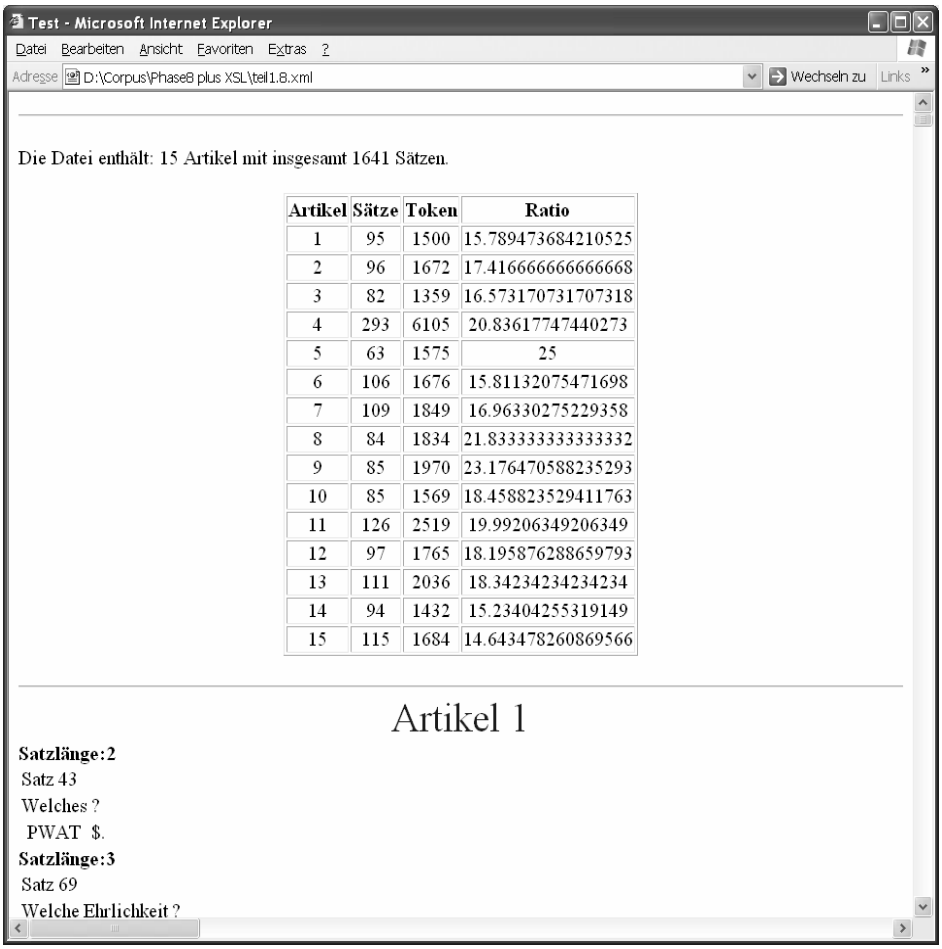
tems überzeugend: Auf gängigen PC-Systemen lassen sich mehr als 100 Sätze pro Minute (partiell) parsen.

3.3 Module zur Lexikon- und Korpuspflege

Die beiden anderen Module des Systems fassen eine Reihe von Werkzeugen zusammen, die es ermöglichen

- die im System verwendeten Lexika zu überarbeiten
  (Suche und Korrektur inkonsistenter Einträge, Einfügen neuer Einträge, Datenexport, …)
- in den Korpusdateien gezielt nach Wortformen, Tags oder Tagsequenzen zu suchen und die gefundenen Ergebnisse direkt zu bearbeiten
- auf der Grundlage von Korpusdateien Frequenzwörterbücher zu generieren, um so einen Vergleich der Lexik verschiedener Korpora zu erleichtern
- einfache statistische Analysen (*deskriptive* Statistik) der Daten durchzuführen (Häufigkeitsverteilung von Token, Lemmata und Tags, durchschnittliche Satzlänge, etc.).

Die Generierung der vom System zurzeit unterstützten statistischen Analysen erfolgt durch XSLT-Skripte. So erzeugt z.B. eines von ihnen eine Tabelle, die für jeden Text einer Korpusdatei die Anzahl der Sätze und Token, sowie die mittlere Satzlänge enthält. Anschließend werden für jeden Text die nach aufsteigender Länge sortierten Sätze ausgegeben (vgl. Abb. 3):



Artikel	Sätze	Token	Ratio
1	95	1500	15.789473684210525
2	96	1672	17.416666666666668
3	82	1359	16.573170731707318
4	293	6105	20.83617747440273
5	63	1575	25
6	106	1676	15.81132075471698
7	109	1849	16.96330275229358
8	84	1834	21.833333333333332
9	85	1970	23.176470588235293
10	85	1569	18.458823529411763
11	126	2519	19.99206349206349
12	97	1765	18.195876288659793
13	111	2036	18.34234234234234
14	94	1432	15.23404255319149
15	115	1684	14.643478260869566

Abb. 3: Korpusauswertung durch XSLT-Skript.

Literatur

Ide, Nancy/Romary, Laurent (2000): Encoding Syntactic annotation. In: Abeillé, Anne (Hrg.), Building and using syntactic annotated corpora. Dordrecht: Kluwer.

– (2001): A Common Framework for Syntactic Annotation. In: *Proceeding of AC*L 2001, 298 – 305. Toulouse.

Lezius, Wolfgang (2002): Ein Suchwerkzeug für syntaktisch annotierte Textkorpora. Stuttgart: Dissertation.

Mikheev, Andrei (2002): Periods, Capitalized Words, etc. In: *Computational Linguistics*, ACL 23(3), 289-318.

Schmid, Helmut (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. In: Proc. of the International conference on New Methods in Language Processing, Manchester, U.K.

Schmid, Helmut (1995). Improvements in Part-of-Speech Tagging with an Application to German. In: Proc. of the EACL SIGDAT workshop, Dublin.