Institut für Informatik
der Universität Zürich
Abt. Computerlinguistik

Sommersemester 2000

Syntaxannotation

unter besonderer Betrachtung der

Projekte TIGER, MATE und NEGRA

sowie der

Sprache XML

Seminararbeit im Rahmen des Seminars
„Syntaxtheorien und computerlinguistische Praxis“
Prof. Dr. M. Hess, lic.phil. G. Schneider und lic.phil. S. Clematide

Andreas Schär
Rosenweg 5
5502 Hunzenschwil
Tel. 062 / 897 19 85
aoschaer@hotmail.com

22. Juni 2000

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit, welche im Rahmen eines Seminars über Syntaxtheorien und computerlinguistische Praxis verfasst worden ist, soll Probleme und Lösungsansätze bei der Darstellung von linguistischen Annotationen untersuchen. Ein Schwergewicht wird dabei auf die Projekte NEGRA, TIGER (mit ihren zugehörigen Korpora) und MATE sowie die Markup-Sprache XML gelegt. Unter Syntaxannotation oder Linguistic Annotation wird im Allgemeinen das Aufzeichnen syntaktischer Einheiten verstanden; idealerweise in einer Art, welche die Beziehungen und Hierarchien zwischen den einzelnen linguistischen Elementen ersichtlich macht. Eine Definition des Linguistic Data Consortium lautet: „‚Linguistic annotation‘ covers any descriptive or analytic notations applied to raw language data. The basic data may be in the form of time functions—audio, video and/or physiological recordings—or it may be textual.“[1] Die Notation kann dabei verschiedenste Transkriptionen (phonetische Merkmale, Diskursstrukturen etc.), Kennzeichnungen für Bedeutung und Part-of-speech, Syntaxanalysen, Koreferenz-Hinweise und anderes beinhalten.

2. XML

2.1. Zur Geschichte von XML

Die Entwicklung von XML begann 1996; eine Tatsache, die dazu verleitet, XML als junge und daher noch unausgereifte Technologie zu betrachten. XML—bereits seit Februar 1998 ein Standard des Internet-Konsortiums W3C—ist aus SGML[2] hervorgegangen, an welcher schon seit den frühen Achzigerjahren gearbeitet wurde und welche seit 1986 ein ISO-Standard[3] ist. Eben-falls aus SGML hervorgegangen ist seit 1990 die heute im Internet dominierende Markup-Sprache HTML[4]. Für XML wurden die besten Komponenten aus SGML entnommen und mit den Erfahrungen aus HTML gepaart. So wurde eine neue Sprache entwickelt, welche nicht weniger mächtig als SGML, dafür regelmässiger und einfacher in der Verwendung ist (mit anderen Worten: „an extremely simple dialect of SGML“[5]).
SGML wird heute mehrheitlich benutzt für technische Dokumentationen und weniger für andere Arten von Daten; bei XML verhält es sich gerade umgekehrt.

[1] ‚Linguistic Annotation‘, Steven Bird & Mark Liberman, Linguistic Data Consortium (LDC), University of Pennsylvania. http://www.ldc.upenn.edu/annotation/
[2] Standard Generalized Markup Language
[3] International Standardization Organization
[4] Hyper Text Markup Language
[5] http://www.xml-zone.com/xmlfaq.asp

2.2. XML, eine Markup-Sprache

XML ist die Abkürzung für *Extensible Markup Language*, was sich nur in unschöner Weise ins Deutsche übersetzen lässt. Viel hilfreicher ist eine kurze Erläuterung des Begriffes.
Extensible steht für ausdehnbar oder erweiterbar. XML ist deswegen *extensible*, weil diese Sprache (*language*) sich im Gegensatz zu—beispielsweise—HTML nicht an einem endlichen Inventar von Tags (Wörter zwischen < und >) orientiert, sondern x-beliebige, selbst definierte Merkmale zulässt.
Bleibt noch *Markup* zu erklären: man versteht darunter die Zusatzangaben zu einem Datensatz, die uns Informationen über dessen Darstellung und Formatierung geben. Sie sind vergleichbar mit den typographischen Anweisungen zur Gestaltung eines Dokumentes, wie sie in Bild 1 dargestellt sind.

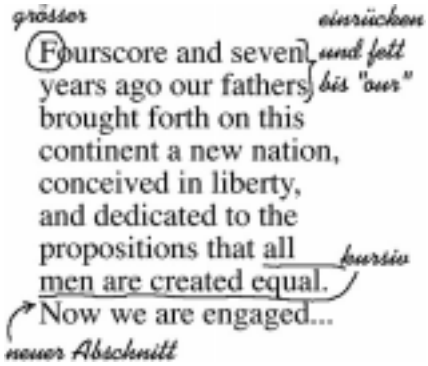

Bild 1: Markup im herkömmlichen Sinne

Eine genaue Definition von *Markup*[6] ist: *„Text that is added to the data of a document in order to convey information about it“*[7], wobei für *Document* und *Data* folgende Umschreibungen gelten sollen:
Data: „The characters of a document that represent the inherent information content; characters that are not recognized as markup“[8].
Document: „A collection of information that is processed as a unit. A document is classified as being of a particular document type“[9].

XML ist dafür ausgelegt, mittels Textdateien strukturierte Daten[10] zu verwalten, wiederzugeben und sie auf einfache Weise im Internet austauschbar zu machen. Seine Verwendung ist jedoch

[6] Es existieren vier verschiedene Arten von Markup: *descriptive Markup* (Beschreibung der Dokumentenstruktur in nicht-systemspezifischer Art, mit Tags), *References* (durch anderen Text ersetzte Markups), *Markup Declarations* (schreiben vor, wie andere Markups im Dokument ausgelegt werden müssen) und *Processing Instructions* (systemspezifische Anweisungen für die Verarbeitung eines Dokuments).
[7] Goldfarb 1990: 273, Definition 4.183
[8] Goldfarb 1990: 260, Definition 4.72
[9] Goldfarb 1990: 263, Definition 4.96

nicht auf das Internet beschränkt, sondern ermöglicht auch in Intranets grösserer Firmen Vereinfachungen. Da sich XML mit seinen Textdateien an einer Syntax orientiert und nicht an binären Zeichenketten, spielt das Betriebssystem bei diesem Datenaustausch keine Rolle; er ist auf allen Plattformen gewährleistet. Mit dem Textformat (im Gegensatz zu einem binären Format) ermöglicht XML das Erzeugen von Dateien, welche eindeutig und für den Computer einfach zu interpretieren und verarbeiten sind, sowie Fallen im Sinne von fehlender Kompabilität, Ungeeignetheit für internationalen Austausch etc. umgehen. Die Eigenschaft „Textdatei“ erleichtert nicht nur der Software, sondern auch Programmierern die Verwaltung von XML-Dateien: ist ein XML File fehlerhaft, so lässt sich mithilfe eines einfachen Texteditors der Fehler beheben. Und: obwohl je länger je bessere Software zum Verfassen von XML auf den Markt gebracht wird, genügt es zum Erzeugen eigener XML-Dateien im Grunde, über ein simples texterzeugendes Programm zu verfügen, denn schliesslich sind XML Files eben nichts anderes als ganz gewöhnliche Textdateien. Um XML-Dateien in einer benutzerfreundlicheren Art betrachten zu können, sind so genannte XML Parser nötig; diese lesen XML-Dateien und zeigen sie in einer übersichtlichen, leicht(er) verständlichen Weise an. Microsoft und Netscape sind daran, XML-parsing-Eigenschaften in ihre Browser einzubauen.
Da XML-Dateien Textdateien sind, brauchen sie mehr Speicherplatz als vergleichbare Dateien in einem binären Format. Da dieVorteile von Textdateien aber—wie oben erläutert—stark überwiegen, ist der erhöhte Speicheraufwand ein Punkt, den man gerne in Kauf nimmt, zumal die Preise für Speicherplatz tiefer denn je sind und zuverlässige Komprimierungssoftware für die gängigen Betriebssysteme (oft gratis) zur Verfügung steht.

2.3. Unterschiede XML – HTML

Während in HTML die Bedeutung eines jeden Tags spezifiziert ist, werden in XML dieselben nur dazu verwendet, Dateneinheiten abzugrenzen. Die Interpretation der Daten bleibt aber der benutzten Anwendung überlassen. Im weiteren beschreibt HTML lediglich, wie ein Dokument in Erscheinung treten soll, sagt aber nichts aus über die Art der enthaltenen Informationen und deren Organisation. XML füllt diese Lücke, indem sie dem Autoren eines Dokuments die Möglichkeit gibt, Information in standardisierter Weise zu organisieren.
Zu beachten ist, dass XML-Regeln viel strenger sind als solche für HTML. Während HTML das Vergessen von Anführungszeichen für ein Attribut vielleicht noch toleriert, führt ein solcher Mangel in XML dazu, dass die Datei unbrauchbar wird. Eine geöffnete Anwendung wird beim entsprechenden Fehler anhalten und die zugehörige Fehlermeldung ausgeben.

[10] Daten in strukturierter Form springen tagtäglich in unser Auge, sei es im Vorlesungsverzeichnis, im Adressbuch, im Börsenteil der Tageszeitung oder auf einem simplen Einladungsschreiben.

2.4. Unterschiede XML – SGML[11]

SGML ist die Muttersprache von XML, letzteres eine gekürzte Form von ersterem. XML lässt die komplizierten und selten verwendeten Teile von SGML weg, ist daher leichter zu verstehen, geeigneter für Datentransfers im Internet und einfacher, um dafür Anwendungen zu programmieren. Eng gesehen ist XML immer noch SGML und könnte (Programmierer bevorzugen vielleicht diese Schreibweise) als SGML^{--} oder HTML^{++} bezeichnet werden[12].

3. Die Projekte

3.1. Das NEGRA Projekt[13]

NEGRA steht für NEbenläufige GRAmmatische Verarbeitung und ist ein Gemeinschaftsprojekt von Wissenschaftlern aus den Bereichen der Computerlinguistik und Informatik. Hauptziel des Projekts ist die Entwicklung neuer Technologien zur grammatischen Verarbeitung, was einerseits gefördert wird durch neue Berechnungsverfahren sowie die Kombination und Weiterentwicklung von Wissenschaftsbereichen, welche bisher getrennte Ansätze verfolgt haben.
Das vielleicht verwirrend wirkende „nebenläufig“ kommt von den nebenläufigen Berechnungsverfahren, welche im NEGRA Projekt einen Schwerpunkt bilden. Sie sind—wie in Bild 2[14] ersichtlich—Bestandteil des Performanzmodells NEGRA, das sich zusätzlich auf die Bereiche Linguistik und Empirie stützt.

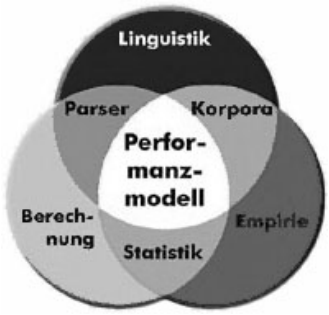

Bild 2: Performanzmodell NEGRA

[11] unter http://www.w3.org/TR/NOTE-sgml-xml gibt es eine sehr detaillierte Liste aller Unterschiede
[12] http://www.ucc.ie/xml/
[13] http://www.coli.uni-sb.de/info/projects/negra.html und http://www.coli.uni-sb.de/sfb378/negra-corpus/negra-corpus.html
[14] Illustration (leicht modifiziert) entnommen aus negra.html (siehe Fussnote 13).

Das NEGRA Korpus gilt als wichtigstes Teilergebnis des NEGRA Projekts. Heute bestehend aus rund 20‘000 annotierten Sätzen aus der Frankfurter Rundschau, wächst das Korpus kontinuierlich. Auch hier werden (wie im TIGER Projekt) Sätze von zwei Personen, die voneinander unabhängig arbeiten, annotiert. Die Annotationen werden sodann verglichen und einer eventuellen Korrektur unterzogen. Zur Steigerung der Effizienz und des Korrektheitsgrades werden die menschlichen Annotatoren durch drei Programme unterstützt: 1. durch das Annotationstool „@nnotate“, welches eine graphische Oberfläche anbietet und mit externen Parsern verbunden ist; 2. durch den statistischen Part-of-Speech Tagger „TnT“, der neben Part-of-Speech-Zuweisungen auch Zuteilungen der grammatischen Funktionen und der Phrasenkategorien ausführt; sowie 3. durch den NP-Chunker „Chunkie“, der Nominal- und Präpositionalphrasen erkennt. Intern liegt das NEGRA Korpus als eine SQL-Datenbank vor, extern ist es im zeilenbasierten Exportformat erhältlich.
Ein kurzes Beispiel für einen annotierten Satz im Exportformat:

```
%% word          tag             morph               edge      parent

#BOS 1
Die              ART             Def.Fem.Nom.Sg      NK        500
Tagung           NN              Fem.Nom.Sg.*        NK        500
hat              VVFIN           3.Sg.Pres.Ind       HD        504
mehr             PIAT            --                  HD        502
Teilnehmer       NN              Masc.Akk.Pl.*       NK        503
als              KOKOM           --                  CM        501
je               ADV             --                  MO        501
zuvor            ADV             --                  HD        501
#500             NP              --                  SB        504
#501             AVP             --                  CC        502
#502             AP              --                  NK        503
#503             NP              --                  OA        504
#504             S               --                  --        0
#EOS 1
```

Die im NEGRA-Korpus kodiert enthaltenen Informationen sind: [A] Part-of-Speech-Tags (nach STTS Stuttgart-Tübingen-Tagset) und morphologische Analyse (terminale Knoten), [B] grammatische Funktionen in den direkt dominierenden Phrasen (Kanten) sowie [C] Kategorien von Phrasen (nichtterminalen Knoten). Die Listen für die entsprechenden Abkürzungen (ausgenommen morphologische Analyse) befinden sich im Anhang unter den oben vergebenen Buchstaben [A] – [C].

3.2. Das TIGER Projekt[15]

Das TIGER Projekt, welches von der Abteilung Computerlinguistik in Saarbrücken, dem Institut für Maschinelle Sprachverarbeitung (IMS) in Stuttgart und dem Institut für Germanistik in Potsdam geführt wird, wurde im Frühjahr 1999 initiiert. In seinem Zentrum steht—so verrät es auch der offizielle Untertitel des Projekts—der Aufbau eines linguistisch interpretierten Korpus des Deutschen. Es sollen dabei neue und verbesserte Methoden für die empirische Sprachuntersuchung entwickelt werden.
Um diesem Ziele gerecht werden zu können, sind grosse Mengen linguistisch interpretierter Daten unabdingbar. Diese waren jedoch zu Beginn des Projekts noch nicht verfügbar, so dass mithilfe des NEGRA Korpus eine erste Überbrückung der grossen Leere ermöglicht wurde. Obwohl das NEGRA Korpus aus rund 350'000 Tokens oder anders gesagt ca. 20‘000 Sätzen deutschsprachigen Zeitungstextes aus der Frankfurter Rundschau besteht, ist dieses Korpus für die Arbeit in den Bereichen korpusbasiertes Parsing oder theoretische linguistische Untersuchungen noch nicht genügend umfangreich. Aus diesem Grunde hat sich die Gruppe um das TIGER Projekt zur Aufgabe gesetzt, in Bezug auf NEGRA ein noch grösseres und detaillierteres Korpus aus deutschem Zeitungstext aufzubauen.
Das TIGER Projekt umfasst im Detail folgende vier Ziele:

1. Entwicklung eines Schemas für die syntaktische Annotation deutschsprachiger Zeitungstexte. Das Schema soll alle Phänomene, welche in derartigen Texten auftauchen, abdecken können und dabei so theorie-unabhängig wie möglich sein. In dieser Weise soll ein grosses Mass an Akzeptanz und Möglichkeit des Wiedergebrauchs (Re-usability) gesichert werden.
2. Entwicklung neuer Techniken für die Automation von Korpus-Annotationen, wobei das Bestreben möglichst schnelle, aber trotzdem verlässliche und genaue Annotationen sind.
3. Syntaktische Annotation von Zeitungstext mithilfe des Annotationsschemas und Werkzeugen für die Automation. Jeder Satz im Korpus wird halb-automatisch annotiert von zwei unabhängig voneinander arbeitenden (menschlichen) Annotatoren. Die zwei Annotationen werden in der Folge miteinander verglichen und—falls nötig—korrigiert.
4. Phänomen-basierte Wiedergewinnung von Sätzen aus dem annotierten Korpus. Es wird eine Abfragesprache für syntaktische Strukturen entwickelt und implementiert, was zu einem mächtigen Werkzeug für linguistische Untersuchungen führen wird.

[15] http://www.ims.uni-stuttgart.de/projekte/TIGER/

3.3. Das MATE Projekt[16]

Die Abkürzung MATE bedeutet „Multilevel Annotation, Tools Engineering“. Ziel des MATE Projekts (ansässig in Dänemark) ist das Vereinfachen des Wiedergebrauchs (re-use) von sprachlichem (Quellen)material, vor allem in Hinblick auf die Entwicklung von Sprachsystemen. Im Vordergrund stehen dabei Fragestellungen bezüglich Erstellung, Gewinnung und Unterhaltung von Textkorpora. Anhand zweier Punkte soll das Ziel hauptsächlich in Angriff genommen werden:

1. Entwicklung eines Standards für die Annotation sprachlichen Materials und
2. Bereitstellung von Werkzeugen, welche einen effizienteren Verlauf von Wissensgewinnung und –extraktion ermöglichen.

Im besonderen will das MATE Projektteam Korpora gesprochenen Dialogs auf mehreren Ebenen untersuchen mit spezieller Berücksichtigung von Betonung und Rhythmus, (Morpho-)Syntax, Koreferenz, Sprechakt und Kommunikationsschwierigkeiten. Die Ergebnisse des Projekts werden vor allem für Entwickler von Dialogsystemen mit gesprochener Sprache wertvoll sein, aber auch Nutzen bringen in der Weiterentwicklung anderer Bereiche des Language Engineering.
Die Technologie im Bereich von SLDSs (Spoken Language Dialogue Systems) ist heute soweit vorangeschritten, dass bereits grosses Interesse seitens der Industrie bemerkbar ist. Für die erfolgreiche Kommerzialisierung von SLDSs fehlen aber immer noch Methoden und Werkzeuge, welche schnellere und effizientere Entwicklungs- und Evaluationsprozesse zulassen. Es gibt zwar eine grosse Zahl verschiedener Annotationsschemata und einige Tools für die Behandlung von annotierten Dialogen, nicht jedoch einen Standard oder allgemeingültige methodologische Richtlinien für die Arbeit mit annotierten Textkorpora. Ein zentrales Problem sind nun also die fehlenden Mittel, auf schnellem Wege zu guten SLDS-Prototypen zu kommen, und zwar mit der Möglichkeit, auf bereits vorhandenes sprachliches (und eben standardisiertes) Grundmaterial zurückgreifen zu können.
MATE beabsichtigt, einen ersten Standard sowie eine Arbeitsplattform für die Annotation von Korpora gesprochenen Dialogs einzuführen. Der angesprochene Standard soll folgenden Bedingungen Rechnung tragen:

1. erlauben, dass es mehrere Annotationsstufen gibt, wobei die verschiedenen Stufen zueinander in Verbindung gebracht werden können,
2. die Koexistenz einer Vielzahl von Codierschemata und Standards zulassen,
3. Mehrsprachigkeit ermöglichen,

[16] http://mate.nis.sdu.dk

4. Einbeziehen von Standardisierungsbemühungen zwischen den USA, Europa und Japan, sowie
5. offen sein in Bezug auf Informationsstufen und die Kategorien innerhalb jeder Stufe.

Die Verwendung des einzuführenden Standards wird unterstützt von der Entwicklung einer frühen Version der Arbeitsplattform. Folgendes sind ihre Schlüsselpunkte:

- Gewinnung und manuelle / halb-automatische Annotation / Modifikation von Datenmaterial unter Verwendung des einzuführenden Standards,
- Visualisierung von umschriebenem Text und Annotationen verschiedener Stufen gemäss benutzerdefinierter Teilansichten (irrelevante Stufen beispielsweise werden ausgeblendet), Vergleichen und Kontrastieren von Annotationen mit verschiedener Codierungen,
- Extraktion / Wiedergewinnung aus annotierten Korpora gemäss beliebigen Auszugskombinationen von Dialogtext einerseits und etwelchen Annotationen anderseits,
- Integration von Tools für das Aufzeichnen (mapping), die Extraktion, die Visualisierung und die Evaluation annotierten Datenmaterials; Import / Export von (teilweise) annotiertem Material und einfaches Einfügen von Resultaten aus externenWerkzeugen.

Das MATE Projekt spielt für die Entwicklung von SLDSs (vor allem) in Europa eine wichtige ökonomische Rolle: der Markt (heute noch ein Nischenmarkt) für derartige Sprachsysteme wird sich einer stark wachsenden Nachfrage erfreuen können—z.B. dank Call Centers und ähnlichen Abnehmern. Die mit der Entwicklung und Produktion involvierten Firmen sind aber vornehmlich kleine und mittlere Unternehmen (KMU) und können es sich nicht leisten, selber Lösungen, wie sie im MATE Projekt vorgestellt wurden, zu entwickeln.

Das MATE Projekt wird besonders relevant sein für:

- das Erstellen von Lexika für SLDSs,
- korpusbasierte Lernprozesse für die Gewinnung von Sprachmodellen, Part-of-Speech-Tagging, Einführen von Grammatiken, Extraktion von Strukturen bei der Verwendung in Dialogkontrollen von SLDSs,
- Entwicklung von Lexika und Grammatiken basierend auf expliziten Beschreibungen der Beziehungen zwischen Phänomenen auf verschiedenen deskriptiven Stufen (z.B. lexikalische und grammatikalische Anhaltspunkte für die Semantik, die Diskurssegmentierung etc.).

4. An XML-based Encoding Format for Syntactically Annotated Corpora

Hinter dem englischen Titel verbirgt sich eine Zusammenarbeit der soeben vorgestellten Projekte TIGER und MATE.
In diesem und folgenden Abschnitten wird etwas eingehender die Beschreibung und Kodierung von Textkorpora behandelt, welche mit hierarchisch strukturierter syntaktischer Information annotiert sind. Wegen der Mannigfaltigkeit von Formaten für die Beschreibung und Speicherung linguistischer Korpora existiert auch eine Vielzahl verschiedener Formate für deren Repräsentation.
Die relativ grosse Zahl verschiedener Annotationsformate, die untereinander nicht kompatibel sind, haben den Wunsch nach einem allgemeinen Grundformat bestärkt. Mit der Hilfe von XML als Markup-Sprache versucht man, die im Wege stehenden theoretischen und technischen Schwierigkeiten aus der Bahn zu räumen oder sie zumindest zu ebnen.

Wie unter anderem im Abschnitt über das MATE Projekt bereits erläutert, sind die uneinheitlichen und miteinander nicht-kompatiblen Formate ein grosser Nachteil in der Arbeit mit linguistischen Korpora. Besonders störend sind die folgenden zwei Aspekte:

1. Jedes Korpus beansprucht ein eigenes Format und eine spezielle Grammatik für die Repräsentation der linguistischen Daten. Dies hat zur Folge, dass auch spezifische Tools für ihre Unterhaltung und den Zugang zu ihnen erforderlich sind.
2. Ebenso einem bestimmten Korpus eigen ist die darunterliegende Theorie mit der Art und Weise, wie die Beschreibung der verschiedenen Entitäten repräsentiert wird.

Das hier vorgestellte XML Format wird von einer Vielzahl von Tools unterstützt und ist offen für verschiedene theoretische Ansätze, während seine Anwendung auf syntaktische Information beschränkt ist.

Bei der Wahl des Kodierformats für syntaktische Annotationen müssen zwei Entscheidungen getroffen werden: erstens muss die zugrundeliegende Theorie (Annotationsschema) festgelegt werden. Da an dieser Stelle eine möglichst grosse Spannweite existierender Formate berücksichtigt werden will, muss so stark wie möglich generalisiert werden. Zweitens steht die geschickte Wahl des Repräsentationsformates an. In diesem Ansatz fiel die Wahl auf XML, primär aufgrund der eingangs der Arbeit erläuterten Vorteile der Portabilität.

Wir können von drei Ebenen sprechen:
1: die Ebene der Markup-Sprache mit dem Grundformat für die Kodierung der zu repräsentierenden Informationen

2: die Ebene der Gestaltungskriterien, welche die Organisation der Einträge innerhalb des Korpus beschreibt (gemäss der zu kodierenden Informationen)
3: die Ebene der Instanzen, welche ein festes Set von Tags und Namen für gegebene Korpora und ihren theoretischen Ansatz definiert.

Zur Ebene der Markup-Sprache: XML
XML wurde bereits eingehend beschrieben, deshalb sei an dieser Stelle nur nochmals auf folgende Punkte hingewiesen: XML ist ein Abkömmling von SGML, hat aber eine strengere Grammatik und ist deshalb einfacher zu parsen. Im Unterschied zu HTML ist XML nicht an eine Zahl vorgegebener Tags gebunden, sondern erlaubt dem Anwender, je nach Anwendung beliebige Tags und somit beliebige Strukturen zu erzeugen. Ferner macht XML keine Einschränkungen über die Entitäten und deren Bezüge, so dass XML als theorieunabhängig betrachtet werden kann.

Zur Ebene der Gestaltungskriterien: Kodierung von Syntax
Syntaktische Informationen in XML zu kodieren, sieht auf den ersten Blick trivial aus. Nehmen wir als Beispiel den Satz[17] „Der Knabe sah die Kuh“, welcher in XML in folgender Weise dargestellt werden könnte:

```
<s>
  <np>
    <w word=“Der“/>
    <w word=“Knabe“/>
  </np>
  <vp>
    <w word=“sah“/>
    <np>
      <w word=“die“/>
      <w word=“Kuh“/>
    </np>
  </vp>
</s>
```

Nun tauchen mit XML bereits zwei Einschränkungen auf, welche die einfache Einbettung in obigem Beispielsatz für die Darstellung der syntaktischen Information unpassend machen. Der Gebrauch von Einbettungen beschränkt nämlich die Vielfalt der Beschreibung zu gerade einmal einer Relation, der Teil-Gesamtheit-Relation. Auf diese Weise werden nicht syntaktische

[17] In Anlehnung an das englische Beispiel in *An XML representation for syntactically annotated corpora*, Presented at LREC 2000, Athens, Greece.

Beziehungen oder Bäume dargestellt, sondern bloss eine hierarchisch aufgebaute Sequenz von eingebetteten Segmenten.
Aus diesem Grunde müssen in der oben vorliegenden Struktur Elemente höherer Ordnung—also beispielsweise Sätze—eine fortlaufende Kette von Unterelementen (Phrasen oder Wörter) einschliessen. Diskontinuierliche Konstituenten können nicht repräsentiert werden. Desweitern besteht (da es nur eine Beziehung gibt) keine Bedeutung für die Kennzeichnung der Beziehung zwischen höheren Elementen und deren Konstituenten. Die Default-Beziehung ist—obgleich nicht explizit erkennbar—die Teil-Gesamtheit-Relation.
In diesem Gemeinschaftsprojekt wird die Verwendung spezieller Elemente für die Kanten in syntaktischen Bäumen oder DAGs vorgeschlagen. Dies erlaubt die explizite Repräsentation von Kanten und ihren zugehörigen Labels. DAGs werden im nächsten Abschnitt näher vorgestellt.

Zur Ebene der Instanzen: Kodierung mit DAGs (*Directed Acyclic Graphs*)
Viele Grammatiktheorien repräsentieren Satzstrukturen mit DAGs (etwa zu übersetzen mit „gerichteten azyklischen Graphen“). Es kommen bei der Beschreibung dieser Graphen grundlegend vier XML Elemente zum Tragen: Satzelemente, nicht-terminale Knoten, terminale Knoten sowie Kanten, welche zum Verbinden der Knoten gebraucht werden. Um ein Maximum an Portabilität zu erreichen, können die Kanten beschriftet werden. Dank dem Vorhandensein dieser baukastenähnlichen Verkettungen können auch kreuzende Kanten (siehe dargestellte Graphik weiter unten) mühelos bewältigt werden. In einem solchen Fall ist die Präzedenz des terminalen Knotens ein wichtiger Aspekt für das Definieren genereller Präzedenz.

Im untenstehenden Beispiel finden wir einen Satz, der nach dem NEGRA Schema annotiert und in XML konvertiert wurde. Die Kanten sind—wie im NEGRA Format üblich—mit Kästchen etikettiert, ferner kommt eine kreuzende Kante vor (*mehr Teilnehmer als je zuvor*).

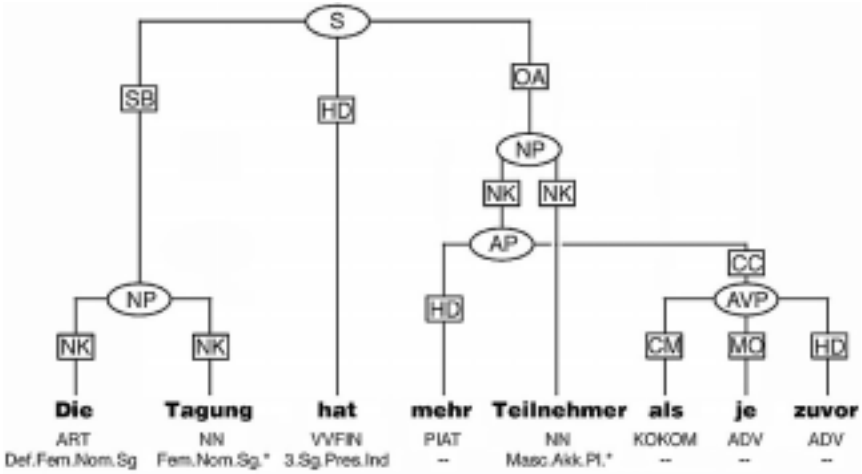

Bild 3: DAG-Darstellung eines Beispielsatzes

```
<s id="s1" xlink:href="s1_n504" crossedge="true"/>
<w id="s1_w1" word="Die" pos="ART" morph="Def.Fem.Nom.Sg"/>
<w id="s1_w2" word="Tagung" pos="NN" morph="Fem.Nom.Sg.*"/>
<w id="s1_w3" word="hat" pos="VVFIN" morph="3.Sg.Pres.Ind"/>
<w id="s1_w4" word="mehr" pos="PIAT" morph="--"/>
<w id="s1_w5" word="Teilnehmer" pos="NN" morph="Masc.Akk.Pl.*"/>
<w id="s1_w6" word="als" pos="KOKOM" morph="--"/>
<w id="s1_w7" word="je" pos="ADV" morph="--"/>
<w id="s1_w8" word="zuvor" pos="ADV" morph="--"/>
<n id="s1_n500" cat="NP">
  <edge id="s1_edge1" label="NK" xlink:href="s1_w1"/>
  <edge id="s1_edge2" label="NK" xlink:href="s1_w2"/>
</n>
<n id="s1_n501" cat="AVP">
  <edge id="s1_edge3" label="CM" xlink:href="s1_w6"/>
  <edge id="s1_edge4" label="MO" xlink:href="s1_w7"/>
  <edge id="s1_edge5" label="HD" xlink:href="s1_w8"/>
</n>
<n id="s1_n502" cat="AP">
  <edge id="s1_edge6" label="HD" xlink:href="s1_w4"/>
  <edge id="s1_edge7" label="CC" xlink:href="s1_n501"/>
</n>
<n id="s1_n503" cat="NP">
  <edge id="s1_edge8" label="NK" xlink:href="s1_n502"/>
  <edge id="s1_edge9" label="NK" xlink:href="s1_w5"/>
</n>
<n id="s1_n504" cat="S">
  <edge id="s1_edge10" label="SB" xlink:href="s1_n500"/>
  <edge id="s1_edge11" label="HD" xlink:href="s1_w3"/>
  <edge id="s1_edge12" label="OA" xlink:href="s1_n503"/>
</n>
```

4.1. Vorteile des XML-basierten Formats

Die Kodierung in diesem Format ist so generell wie möglich ausgestaltet, d.h. existierende Annotationsformate können mühelos repräsentiert und andere Darstellungen weiter ausgebaut werden, so dass zusätzliche Strukturtypen und Merkmale berücksichtigt werden können. Da dieses Format unabhängig von linguistischen Theorien ist, kann es als generelles Repräsentationsformat angewandt werden.

Ein weiterer wichtiger Vorzug ist die Verwendung von XML als Kodierformalismus oder –grammatik. Wie schon bekannt ist, ermöglicht XML ein leicht erweiterbares Markup, so dass völlig verschiedene Annotationsstufen (z.B. Semantik und Syntax) leicht miteinander kombiniert werden können. Es besteht auch die Möglichkeit (im Gegensatz zu SGML), unterschiedliche Stufen der Beschreibungen in mehrere Dateien aufzuteilen.
Die Zukunft von XML floriert: es sind schon viele wertvolle XML Tools erhältlich und laufend werden neue implementiert. Das vereinfacht den Zugang zu XML-gestützten Formalismen und deren Handhabung. Beispielsweise erlauben validierende Parser die Kontrolle über den In- und Output von XML Konvertierungsroutinen, Visualisierungstools ermöglichen Übersichten über vorhandene Satzstrukturen, XML-unterstützte Browser vereinfachen in Verbindung mit so genannten Style Sheets das Durchsehen von XML-annotierten Korpora, und XML Such-maschinen schliesslich, welche die Suche in hierarchisch strukturierten Dokumenten ermöglichen werden, stehen in der Entwicklungsphase.

4.2. Ein XML-formatierter Beispielsatz mit detaillierten Erläuterungen[18]

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

<!-- XML representation for syntactically annotated corpora
     defined by Wolfgang Lezius & Andreas Mengel.

     More information at: http://www.ims.uni-stuttgart.de/projekte/TIGER/xml
-->

<!DOCTYPE doc [

<!ELEMENT doc (head,body)>
<!ATTLIST doc
          id             ID         #REQUIRED>

<!ELEMENT head (name?,format?,author?,date?,description?,history?,definition?)>
<!ATTLIST head
          id             ID         #REQUIRED>

<!ELEMENT name (#PCDATA)*>
<!ATTLIST name
```

[18] Beispiel stammt von http://www.ims.uni-stuttgart.de/projekte/TIGER/xml/corpus/ims-democorpus.txt

```
          id             ID         #REQUIRED>

<!ELEMENT format (#PCDATA)*>
<!ATTLIST format
          id             ID         #REQUIRED>

<!ELEMENT author (#PCDATA)*>
<!ATTLIST author
          id             ID         #REQUIRED>

<!ELEMENT date (#PCDATA)*>
<!ATTLIST date
          id             ID         #REQUIRED>

<!ELEMENT description (#PCDATA)*>
<!ATTLIST description
          id             ID         #REQUIRED>

<!ELEMENT history (#PCDATA)*>
<!ATTLIST history
          id             ID         #REQUIRED>

<!ELEMENT definition (#PCDATA)*>
<!ATTLIST definition
          id             ID         #REQUIRED
          xmlns:xlink    CDATA      #FIXED "http://www.w3.org/1999/xlink"
          xlink:type     (simple)   #FIXED "simple"
          xlink:show     (embed)    #FIXED "embed"
          xlink:actuate  (onLoad)   #FIXED "onLoad"
          xlink:href     CDATA      #REQUIRED>

<!ELEMENT body (s|n|w)*>
<!ATTLIST body
          id             ID         #REQUIRED
          xmlns:xml      CDATA      #FIXED "http://www.w3.org/XML/1998/namespace"
          xml:base       CDATA      #REQUIRED>

<!ELEMENT s EMPTY>
<!ATTLIST s
          id             ID         #REQUIRED
```

```
          xmlns:xlink    CDATA      #FIXED "http://www.w3.org/1999/xlink"
          xlink:type     (simple)   #FIXED "simple"
          xlink:show     (embed)    #FIXED "embed"
          xlink:actuate  (onLoad)   #FIXED "onLoad"
          xlink:href     CDATA      #REQUIRED crossedge (true|false) "false">

<!ELEMENT n (edge)*>
<!ATTLIST n
          id             ID         #REQUIRED
          cat            CDATA      #IMPLIED>

<!ELEMENT edge EMPTY>
<!ATTLIST edge
          id             ID         #REQUIRED
          label          CDATA      #IMPLIED
          type           CDATA      #IMPLIED
          xmlns:xlink    CDATA      #FIXED "http://www.w3.org/1999/xlink"
          xlink:type     (simple)   #FIXED "simple"
          xlink:show     (embed)    #FIXED "embed"
          xlink:actuate  (onLoad)   #FIXED "onLoad"
          xlink:href     CDATA      #REQUIRED>

<!ELEMENT w (edge)*>
<!ATTLIST w
          id             ID         #REQUIRED
          word           CDATA      #IMPLIED
          pos            CDATA      #IMPLIED
          morph          CDATA      #IMPLIED>

]>

<doc id="doc">

<head id="head">
  <format id="format">Negra 3</format>
</head>

<body id="body" xml:base="#">

<s id="s1" xlink:href="s1_n504" crossedge="true"/>
```

```
<w id="s1_w1" word="Die" pos="ART" morph="Def.Fem.Nom.Sg"/>
<w id="s1_w2" word="Tagung" pos="NN" morph="Fem.Nom.Sg.*"/>
<w id="s1_w3" word="hat" pos="VVFIN" morph="3.Sg.Pres.Ind"/>
<w id="s1_w4" word="mehr" pos="PIAT" morph="--"/>
<w id="s1_w5" word="Teilnehmer" pos="NN" morph="Masc.Akk.Pl.*"/>
<w id="s1_w6" word="als" pos="KOKOM" morph="--"/>
<w id="s1_w7" word="je" pos="ADV" morph="--"/>
<w id="s1_w8" word="zuvor" pos="ADV" morph="--"/>
<n id="s1_n500" cat="NP">
  <edge id="s1_edge1" label="NK" xlink:href="s1_w1"/>
  <edge id="s1_edge2" label="NK" xlink:href="s1_w2"/>
</n>
<n id="s1_n501" cat="AVP">
  <edge id="s1_edge3" label="CM" xlink:href="s1_w6"/>
  <edge id="s1_edge4" label="MO" xlink:href="s1_w7"/>
  <edge id="s1_edge5" label="HD" xlink:href="s1_w8"/>
</n>
<n id="s1_n502" cat="AP">
  <edge id="s1_edge6" label="HD" xlink:href="s1_w4"/>
  <edge id="s1_edge7" label="CC" xlink:href="s1_n501"/>
</n>
<n id="s1_n503" cat="NP">
  <edge id="s1_edge8" label="NK" xlink:href="s1_n502"/>
  <edge id="s1_edge9" label="NK" xlink:href="s1_w5"/>
</n>
<n id="s1_n504" cat="S">
  <edge id="s1_edge10" label="SB" xlink:href="s1_n500"/>
  <edge id="s1_edge11" label="HD" xlink:href="s1_w3"/>
  <edge id="s1_edge12" label="OA" xlink:href="s1_n503"/>
</n>

</body>
</doc>
```

Der Beispielsatz aus dem TIGER Korpus enthält weitere Sätze[19], welche jedoch aus platzlichen Gründen wie auch wegen fehlendem weitergehenden und didaktischen Vorteil weggelassen wurden.

[19] auf http://www.ims.uni-stuttgart.de/projekte/TIGER/xml/corpus/ims-democorpus.txt

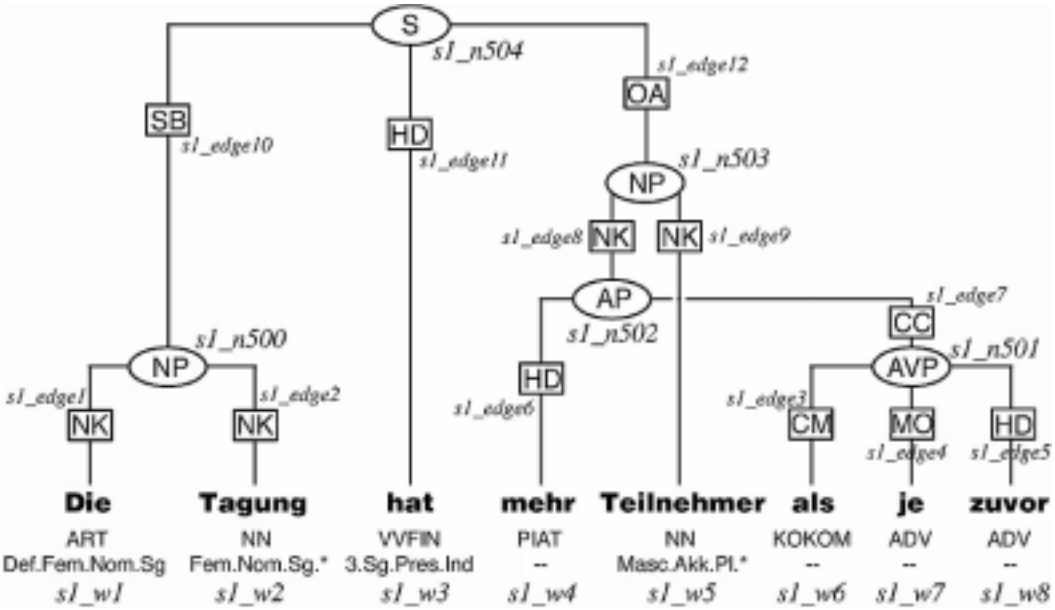

Bild 4: Baum des Beispielsatzes mit markierten Knoten und Kanten

1. Prolog

Ein XML Dokument beginnt im Idealfall mit dem *Prolog*, welcher Auskunkft gibt über die XML Version (*XML Declaration*), die Dokumententypen (*Document Type Declaration*) und andere Charakteristika des Dokumentes. Die *XML Declaration* wie auch *die Document Type Declaration* sind optional, werden aber mit Vorteil angegeben, um ein Höchstmass an Zuverlässigkeit zu erreichen. Unter diese zwei Declarations können beliebige Kommentare (zwischen `<!--` und `-->`) sowie Leerraum eingesetzt werden. Das Ende des Prologs liegt beim ersten Start-Tag.

In unserem Beispiel erkennen wir, dass es sich um ein Dokument der XML Version 1.0 handelt und erklärender Kommentar von den Verfassern mitgeliefert wird.

`<!` ist der Beginn einer Markup-Deklaration, während `<?` für den Anfang einer Ablaufinstruktion steht. Steht `?` am Schluss eines Ausdruckes, so impliziert er Optionalität, d.h. 0- oder 1-maliges Vorkommen desselben.

2. Die Document Type Declaration

Die *Document Type Declaration* beginnt mit `<!DOCTYPE...` und hört kurz vor dem Datenkörper (*body*) mit `]>` auf. Sie sagt aus, um was für ein Dokument es sich handelt, etwa im Sinne von: „Dieses Dokument ist ein Konzertposter.“[20] Die in dieser Declaration *enthaltene Document Type Definition* (*DTD*[21]) sagt mehr darüber aus, welche Anforderungen an ein Konzertposter gestellt

[20] Vergleich entnommen aus Goldfarb 1998:450

[21] Nota bene: DTD ist für Document Type Definitions reserviert, da viel öfter von diesen gesprochen wird, als von einer Document Type Declaration.

werden und was auf ihm wie beschrieben wird. Sie definiert Elententypen, Attribute, Entitäten und Notationen und hält fest, welche von ihnen innerhalb des Dokumentes erlaubt sind und an welchen Stellen sie erlaubt sind.

3. `DOCTYPE`
Mit diesem Kürzel wird ein kurzer Hinweis auf die Art des XML Dokumentes gegeben. Im vorliegenden Beispiel liegt ein fantasieloses `doc` vor, es könnte gerade so gut auch ein `label`, ein `flyer` oder ein `pricelist` sein—die Wahl ist ganz dem Autoren überlassen.
Während in unserem Beispiel die DTD in der selben Datei liegt wie die (linguistischen) Daten, ist auch ein Auslagern der DTD in ein separates File denkbar. In dem Falle würde in diesem Bereich der Verweis auf die zugehörige, externe DTD plaziert sein.

4. `ELEMENT`
Mit dem Ausdruck `ELEMENT` wird die Definition der im Dokument vorkommenden Elemente markiert. Diese bilden die Stütze des XML Markup, denn jedes Element in einem gültigen XML Dokument muss konform sein mit einem in der DTD deklarierten Elententyp. Eine so genannte *Element Type Declaration* muss mit `<!ELEMENT` beginnen, den Namen des Elementes ersichtlich machen (man nennt den Elementenname *generic identifier*) und schliesslich eine Inhaltsspezifikation enthalten. In unserem Beispiel heisst das erste Element `doc` und enthält einen `head` und einen `body`. Das nächste Element nennt sich `head` und hat die Inhalte `name`, `format`, `author` etc., welche aber alle optional sind (d.h. 0 oder 1 Mal vorkommen dürfen), da hinter diese ein `?` gesetzt wurde. Im folgenden Element name ist der Inhalt `PCDATA` (*parsed character data*). Dies sind erwartete Zeichen, welche beim Parsen des XML Dokuments nicht als Markup erkannt werden, sondern als Datenmaterial. Verwandt mit `PCDATA` ist `CDATA` (z.B. im Attribut `xlink:href` des Elements `definition`), dem einfachsten Wertetyp für Attribute: hier genügt als einzusetzender Wert eine beliebige Zeichenkette. `CDATA` teilt dem Prozessor mit, dass es hier nichts zu parsen gibt.
Der Stern hinter `PCDATA` (`*`) bedeutet ebenfalls Optionalität, jedoch im Sinne von 0 oder mehreren Vorkommen.

5. `ATTLIST`
XML bedient sich Attributen, um zu gegebenen Elementen zusätzliche kurze und unstrukturierte Information hinzuzufügen. Dies geschieht in der so genannten *Attribute-list declaration*, die so heisst, weil zu einem Element beliebig viele Attribute definiert werden dürfen, was schliesslich zu einer Liste führen würde (siehe z.B. Element `definition` dieses Dokuments). Der Aufbau einer *Attribut-list declaration* sieht wie folgt aus: hinter dem Initial `<!ATTLIST` steht *der generic identifier* (siehe unter 4.) des Elementes, auf welchem das Attribut Gültigkeit hat, dann folgt der Name des Attributs, der Typ des Attributwertes und schliesslich der Defaultwert. Das vierte Element in unserem Beispiel (Element `format`) hat eine *Attribute-list declaration* mit einem einzigen Attribut: es heisst `id`, ist gültig auf dem Element `format` und muss den Wertetyp `ID`

haben. Einen Defaultwert hat es nicht, weil `#REQUIRED` die Erforderlichkeit des Wertes aus ID ausdrückt (`#` markiert einen reservierten Namen als Gegensatz zu einem benutzerdefinierten Namen). ID steht für *unique identifier*, einen Namen, welcher in eindeutiger Weise ein Element identifiziert. Ein Element darf höchstens *einen* solchen Verweis besitzen, deshalb gibt es auch in unserem Beispieldokument kein einziges Element, unter dessen ATTLIST mehr als ein id-Attribut zu finden wäre.
Neben `REQUIRED` kommen in den diversen *Attribute-list declarations* `FIXED` und `IMPLIED` vor. `FIXED` bedeutet fixes Attribut, dessen spezifischer Wert (falls vorhanden) identisch sein muss mit dem Defaultwert, d.h. kein anderer Wert ist zugelassen. `IMPLIED` bedeutet, dass ein Quellen- oder Ergebniselement durch die Anwendung impliziert wird.

6. `ELEMENT body (s|n|w)*`
Der Inhalt dieses Elements kann `s`, `n` oder `w` sein, also Satz (*sentence*), Knoten (*node*) oder Wort (*word*). Der senkrechte Strich (`|`) steht für die Auswahl im Sinne von: wähle eines aus, aber nur eines. Auch hier steht zudem ein `*`, was beliebiges Vorkommen (0 oder mehrere Male) des ausgewählten Dokuments angibt.

7. Leeres Element (`EMPTY`)
Den Elementen s und edge wird kein Inhalt im herkömmlichen Sinn zugesprochen, da anstelle einer Inhaltsformulierung die Angabe `EMPTY` notiert ist. Beide Elemente verfügen aber über Attribute, welche dieselben näher bestimmen. Leere Elemente stehen meist als Platzhalter für Inhalt, welcher generiert wird. Dies wird z.B. bei der Erstellung von Inhaltsverzeichnissen verwendet, die durch anderen Text bestimmt werden, oder als Markierer für eine Abbildung, welche vom System bei der Zusammenstellung oder von Hand eingesetzt wird.

8. Xlink-Verweise
Ein wiederkehrendes Element in den *Attribute-list declarations* sind die `xlink` Verweise. *XLink*, die XML Linking Sprache, ist ein spezielles Konstrukt in der Welt von XML und stellt Link-Techniken zur Verfügung, die weit über die Möglichkeiten derjeniger von HTML hinausgehen. Links können hier verschiedene Endpunkte haben, in verschiedenen Richtungen begangen oder in Gruppen resp. Datensätzen—unabhängig von ihrem Mutterdokument—abgelegt werden.
Wir stossen in den letzten *Attribute-list declarations* wiederholt auf die Attributnamen `xlink:type`, `show`, `actuate` und `href`. Vor allem letzteres mag HTML Programmierern ein Begriff sein; es handelt sich nämlich um die Zielangabe eines Links. XML führt den Benutzer mittels `href`—im Beispieldokument unter Verwendung von `CDATA`—an die entsprechende Stelle im linguistischen Datensatz.

Die Bedeutung des Attributs `type` konnte nicht mit Sicherheit ausgemacht werden. Wahrscheinlich handelt es sich hierbei um die Art der Beschriftung des betroffenen Elements (`simple`).

Das Attribut `show` gibt an, wie das durch den Link beschaffte Datenmaterial angezeigt werden soll. Im vorliegenden Fall geschieht das durch Einbettung (`embed`), d.h. nach Ablaufen des Links wird die Information von der Link-Zielseite an den Ort, von welchem der Link ausgegangen ist, zur Ansicht oder Verarbeitung eingefügt.
Das verbleibende Attribut `actuate` steuert den Ablauf des Links: er bestimmt, auf welchen Befehl ein Link vollzogen werden soll. In unserem Beispiel folgt `actuate` dem Wert `onLoad`, was bedeutet, dass der Benutzer einen Load-Vorgang starten muss (sei dies durch Klicken auf ein Load-Icon, Drücken von ctrl + l oder dergleichen).

Die abgestossen wirkenden Zeilen, welche die DTD vom eigentlichen Datenkörper trennen, bestimmen die IDs der Konstrukte `doc` und `head`: `doc` beansprucht als Eigennamen `doc`, bei `head` verhält es sich in entsprechender Weise. `format` wird identifiziert mit „Negra 3“.

9. Tags
An dieser Stelle tritt nun ein bisher unbeschriebenes Konstrukt auf: die sogenannten Tags. Im Datenteil des Beispiels zu Dutzenden sichtbar, markieren sie die Anfänge und Enden von Elementen. Start-Tags werden mit einem < , dem Namen des Elementes (*generic identifier*) sowie dem abschliessenden > markiert. Sie können—im Gegensatz zu End-Tags—Attribute enthalten, wie dies beispielsweise bei der Identifikation von `head` der Fall ist. End-Tags sind beschrieben in der Form `</x>`, wobei x für den *generic identifier* steht.

10. Eigentliche Datenblock
Der Datenblock des Dokuments (im vorliegenden Fall `body` benannt) enthält die eigentlichen Datenwerte aus dem linguistischen Korpus. Sämtliche in der DTD beschriebenen Elemente kommen hier nun zum Zuge als Datenträger. Verwendet werden (pro memoria) die Elemente Satz, Non-terminale (Knoten), Terminale (Wörter) und Kanten. Alle entsprechenden Abkürzungslisten befinden sich im Anhang.

Ein Satz trägt als Kennzeichnung den Buchstaben `s`. Gemäss Beschreibung soll er ein `id` zugesprochen erhalten, eine Art Knotenverweis `href` sowie die Angabe über kreuzende Kanten. Vergleichen wir dies mit dem Element `s` im Datensatz, so ist eine korrekte Unifikationsweise feststellbar.

Ebenso gut nachvollziehbar ist der Referenzvorgang für Knoten. In einer ersten Zeile finden wir jeweils die `id` und die Katergorie (`cat`), darunter die Angaben für von den Knoten ausgehende Kanten (`edge`), welche für sich ein `id`, ein `label` (grammatische Funktion), sowie ein `xlink:href` (unteres Ende der Kante: Terminal oder Nicht-terminal) beanspruchen (alle Daten sind aus dem `CDATA` Bereich).

Bleibt noch die Beschreibung von `w`, den Wörtern. Wie aus der *entsprechenden Attribute-list declaration* ersichtlich, sind für Wörter eine Identifikation `id`, das explizite Wort, die Part-of-Speech Angabe `pos` sowie Angaben für die Morphologie `morph` vorgesehen.

5. Schlusswort

Ich hoffe, mit dieser Arbeit mindestens einen guten Einblick in die Welt von XML, linguistischen Korpora und Syntaxannotationen gegeben haben zu können. Vielleicht entspringt mit ihrer Lektüre sogar die Lust, noch mehr über diese Themen zu erforschen. Im Speziellen denke ich hier an XML, welche zwar jung ist, aber in den nächsten Jahren noch intensiv im Gespräch sein wird. Sie bietet—wie in meiner Arbeit grob vorgestellt—enorme Vorteile im Bereich des Datenverkehrs auf Netzwerken und ich vermute, dass sie einen anhaltenden Impact auf die Welt des Internets haben wird. Nicht weniger interessant werden auch die Entwicklungen in Sachen Textkorpora, Annotationsschemata etc. ausfallen. Hier verweise ich besonders auf den Abschnitt über das MATE Projekt.
Weitergehende Fragen können mit Besuchen auf den Sites, welche ich in der Bibliographie (welche fast eher Linkographie genannt werden sollte, und da ist meine Arbeit überhaupt kein Einzelfall) angegeben habe, beantwortet werden. Ich empfehle jeder und jedem, sich einmal „einzuklicken“...

5.1. Dankesworte

Ich danke all jenen, die mir durch ihre Hinweise und Anregungen neue Impulse für meine Arbeit gegeben haben oder mich in irgend einer anderen Weise unterstützt haben.
Spezieller Dank gebührt Nicole Wicki für die grosszügige Unterstützung beim Zusammentragen relevanter Informationen aus dem Internet, Tobi Wicki für seine spontane helfende Hand in der Ausdruckstufe, Markus Stampfli für das Zurverfügungstellen seines Computers für diverse Anwendungen, und denen, die mich zwischendurch mit kulinarischen Kleinigkeiten aufgebaut haben.

6. Anhang

[A]

ADJA	attributives Adjektiv	PWS	substituierendes Interrogativpronomen
ADJD	adverbiales oder prädikatives Adjektiv	PWAT	attribuierendes Interrogativpronomen
ADV	Adverb	PWAV	adverbiales Interrogativ- oder Relativpronomen
APPR	Präposition; Zirkumposition links	PAV	Pronominaladverb
APPRART	Präposition mit Artikel	PTKZU	zu vor Infinitiv
APPO	Postposition	PTKNEG	Negationspartikel
APZR	Zirkumposition rechts	PTKVZ	abgetrennter Verbzusatz
ART	bestimmter oder unbestimmter Artikel	PTKANT	Antwortpartikel
CARD	Kardinalzahl	PTKA	Partikel bei Adjektiv oder Adverb
FM	Fremdsprachliches Material	SGML	SGML Markup
ITJ	Interjektion	SPELL	Buchstabierfolge
ORD	Ordinalzahl	TRUNC	Kompositions-Erstglied
KOUI	unterordnende Konj. mit „zu“ + Infinitiv	VVFIN	finites Verb, voll
KOUS	unterordnende Konjunktion mit Satz	VVIMP	Imperativ, voll
KON	nebenordnende Konjunktion	VVINF	Infinitiv, voll
KOKOM	vergleichskonjunktion	VVIZU	Infinitiv mit „zu“, voll
NN	normales Nomen	VVPP	Partizip Perfekt, voll
NE	Eigennamen	VAFIN	finites Verb, aux
PDS	substituierendes Demonstrativpronomen	VAIMP	Imperativ, aux
PDAT	attribuierendes Demonstrativpronomen	VAINF	Infinitiv, aux
PIS	substituierendes Indefinitpronomen	VAPP	Partizip Perfekt, aux
PIAT	attr. Indefinitpronomen ohne Determiner	VMFIN	finites Verb, modal
PIDAT	attr. Indefinitpronomen mit Determiner	VMINF	Infinitiv, modal
PPER	irreflexives Personalpronomen	VMPP	Partizip Perfekt, modal
PPOSS	substituierendes Possessivpronomen	XY	Nichtwort, Sonderzeichen enthaltend
PPOSAT	attribuierendes Possessivpronomen	\$,	Komma
PRELS	substituierendes Relativpronomen	\$.	satzbeendende Interpunktion
PRELAT	attribuierendes Relativpronomen	\$(	sonstige Satzzeichen; satzintern
PRF	reflexives Personalpronomen		

[B]

AC	adpositional case marker	MW	way (directional modifier)
ADC	adjective component	NG	negation
AMS	measure argument of adj	NK	noun kernel modifier
APP	apposition	NMC	numerical component
AVC	adverbial phrase component	OA	accusative object
CC	comparative compliment	OA2	second accusative object

CD	coordinating conjunction
CJ	conjunct
CM	comparative conjunction
CP	complementizer
DA	dative
DH	discourse-level head
DM	discourse marker
GL	prenominal genitive
GR	postnominal genitive
HD	head
JU	junctor
MC	comitative
MI	instrumental
ML	locative
MNR	postnominal modifier
MO	modifier
MR	rhetorical modifier
OC	clausal object
OG	genitiv object
PD	predicate
PG	pseudo-genitive
PH	placeholder
PM	morphological particle
PNC	proper noun component
RCr	elative clause
RE	repeated element
RS	reported speech
SB	subject
SBP	passivised subject (PP)
SP	subject or predicate
SVP	separable verb prefix
UC	(idiosyncratic) unit component
VO	vocative

[C]

AA	superlative phrase
AP	adjective phrase
AVP	adverbial phrase
CAC	coordinated adposition
CAP	coordinated adjective phrase
CAVP	coordinated adverbial phrase
CCP	coordinated complementizer
CH	chunk
CNP	coordinated noun phrase
CO	coordination
CPP	coordinated adpositional phrase
CS	coordinated sentence
CVP	coordinated verb phrase (non-finite)
CVZ	coordinated „zu“-marked infinitive
DL	discourse level constituent
ISU	idiosyncratis unit
MPN	multi-word proper noun
MTA	multi-token adjective
NM	multi-token number
NP	noun phrase
PP	adpositional phrase
QL	quasi-language
S	sentence
VP	verb phrase (non-finite)
VZ	„zu“-marked infinitive

7. Bibliographie

Bücher

- Goldfarb, Charles F., The XML Handbook, Prentice-Hall, Upper Saddle River, 1998
- Goldfarb, Charles F., The SGML Handbook, Oxford University Press, New York, 1990
- Gough, Kevin John, Syntax Analysis and Software Tools, Addison-Wesley Publishing, Cornwall, 1988

WWW-Seiten

XML
Es gibt schon sehr viel Material zu XML. Empfehlenswerte Seiten sind zum Beispiel:

- http://www.w3.org/XML/ (offizielle XML Seite)
- http://www.xml-zone.com/xmlfaq.asp
- http://www.irt.org/articles/js072/index.htm
- http://www.ucc.ie/xml/

Syntaxannotationen

- http://www.ldc.upenn.edu/annotation/

TIGER

- http://www.ims.uni-stuttgart.de/projekte/TIGER

MATE

- http://mate.nis.sdu.dk

NEGRA

- http://www.coli.uni-sb.de/info/projects/negra.html
- http://www.coli.uni-sb.de/sfb378/negra-corpus/negra-corpus.html