

DSSSL zur Verarbeitung linguistischer Korpora¹

Andreas WITT

Einleitung

Die Document Style Semantics and Specification Language [ISO 10179:1996] – oder kurz DSSSL – ist seit 1996 ein Standard der Internationalen Standardisierungsorganisation (ISO). In ihm werden insgesamt vier verschiedene – jedoch miteinander verwobene oder kombinierbare – Sprachen definiert. Mit Hilfe dieser Sprache(n) ist es möglich, Texte, die nach den Vorgaben der Standard Generalized Markup Language (SGML, [ISO 8879:1986]) annotiert wurden, zu transformieren und/oder zu formatieren. DSSSL stellt im Gegensatz zu den meisten anderen texttechnologischen Standards² eine Programmiersprache dar. In den meisten Anwendungen werden die Programmiermöglichkeiten insbesondere für die Druckaufbereitung von Texten genutzt (z.B. zur Index- oder Inhaltsverzeichnis-erzeugung). Die Möglichkeiten von DSSSL gehen jedoch über dieses primäre Anwendungsgebiet hinaus. Computerlinguistische Verfahren, z.B. Parsingalgorithmen, lassen sich gut mit den in DSSSL verwendeten Programmiermöglichkeiten verbinden. Es soll nachfolgend skizziert werden, wie eine solche Integration bewerkstelligt werden kann.

DSSSL & Co.

Die Document Style Semantics and Specification Language stellt nicht die einzige Möglichkeit dar, SGML-Dokumenten eine Semantik zuzuweisen. Neben proprietären Lösungen³ gibt es für diesen Zweck die

¹ Auf der WWW-Seite <http://coli.lili.uni-bielefeld.de/~andreas/dsssl> finden sich zusätzliche Informationen zu diesem Artikel.

² Als texttechnologische Standards werden hier sowohl die ISO Standards (z.B. [HYTIME] oder [SGML] selbst) als auch die Industriestandards (z.B. [XML], [XSL], [CSS]) und akademische Standards (z.B. [TEI]) zusammengefasst, die sich mit der Textanotierung bzw. der Dokumentenverarbeitung befassen.

³ Der SGML-Editor, der aus WordPerfect heraus zugreifbar ist, stellt ein Beispiel für

Standards des *World Wide Web Consortium* CSS und XSL. DSSSL unterscheidet sich von diesen Lösungen insbesondere durch die in sie integrierte Programmiersprache. Insgesamt definiert der Standard vier Sprachen:

transformation language: Die Transformationssprache ermöglicht es, ein strukturiertes SGML-Dokument automatisch in ein anders strukturiertes SGML-Dokument zu überführen. Das Ergebnisdokument kann derselben oder einer anderen DTD als das Ausgangsdokument genügen.

style language: Mittels der Formatierungssprache können Anweisungen definiert werden, die ein SGML-Dokument in ein Ausgabe-medium überführen.

expression language: Programmiersprache (s.u.)

query language: Die Abfragesprache stellt Funktionen zur Verfügung, die es erlauben, in einem SGML-Dokument strukturiert zu suchen.

Der *expression language* kommt ein besonderer Status zu, da sich die anderen drei Sprachen ihrer bedienen. Sie stellt eine komplette Programmiersprache zur Verfügung, die somit sowohl zur Formatierung als auch zur Transformation von SGML-Dokumenten verwendet werden kann. Die *expression language* ist in starker Weise an der Programmiersprache Scheme orientiert und stellt im Kern alle funktionalen Komponenten dieses LISP-Dialektes zur Verfügung. Dies bedeutet, aus der entgegengesetzten Perspektive betrachtet, dass DSSSL insbesondere die Funktionen von Scheme nicht übernommen hat, die dem funktionalen Charakter der LISP-Programmierung widersprechen, da sie nur zum Zweck ihrer *side-effects* Verwendung finden. Zwei Beispiele hierfür sind die Möglichkeit von Scheme, mehrere Funktionen innerhalb eines Blockes (mittels *begin*) aufzurufen, und zweitens *display*, deren *side-effect* dazu führt, dass ein

die Verwendung proprietärer Formatierungsbeschreibungen dar. Die SGML-Browser Panorama von SoftQuad und Multidoc von Citec verwenden ebenfalls eigene *style-sheets*. Ein Vorläufer all dieser Formatierungssprachen stellt die Verwendung von FOSI (*Formatted Output Specification Instances*) dar. Diese finden nach wie vor Verwendung, z.B. im Adept-Editor der Firma Arbor-Text, dem umfangreichsten SGML/XML-Editor überhaupt.

Text auf den Bildschirm ausgegeben wird und von der viele Programmierinnen und Programmierer vermutlich nicht einmal wissen, wie ihr genauer Rückgabewert lautet. Konsequenterweise können in DSSSL die “Funktionen” *begin* und *display* nicht verwendet werden.

Automatische Annotation

Obwohl bereits im Text des Standards darauf verwiesen wird, dass es DSSSL ermöglicht, unstrukturierte, d.h. nicht-annotierte Texte zu verarbeiten⁴, wird diese Möglichkeit der Nutzung von DSSSL bisher kaum zur Kenntnis genommen.

Um einen nicht-annotierten Text mit DSSSL verarbeiten zu können, ist es notwendig, diesen Text in ein SGML-Dokument zu überführen. Hierfür muss nicht einmal die Textdatei verändert werden – es genügt, diese Textdatei in ein SGML-Rahmendokument zu integrieren. Dazu wird die den Text enthaltende Datei an eine allgemeine Entität gebunden und diese zwischen den öffnenden und schließenden Elementen aufgerufen.⁵

```
<!DOCTYPE unstrukturierter_text [
<!ELEMENT unstrukturierter_text (#PCDATA)>
<!ENTITY Textdatei SYSTEM "datei.txt">]>
<unstrukturierter_text>
&Textdatei;
</unstrukturierter_text>
```

Im Bereich der Computerlinguistik, insbesondere aber in der Korpuslinguistik, werden große, meist (noch) nicht annotierte Texte als Eingabe

⁴ “DSSSL includes: a) Constructs that provide access to, and control of, all possible marked-up information in an SGML document, as well as *mechanisms for string processing to allow for the manipulation of non-marked up data.*” (ISO10179,2, Hvhbg. AW)

⁵ Zu beachten ist allerdings bei diesem Vorgehen, dass die Textdatei bestimmte SGML-Zeichen (z.B. ‘<’) nicht beinhalten darf, da das Inhaltsmodell als *#PCDATA* definiert wurde. Diese Definition erfolgte aus Rücksicht auf XML - in vollem SGML kann dieses Problem mit der Festlegung des Inhaltsmodells auf *CDATA* oder *RCDATA* umgangen werden.

für eine maschinelle Verarbeitung genutzt. Auch diese "rohen" Daten können untersucht werden.⁶ Für viele linguistische Anwendungen stellen jedoch Korpora, die nach bestimmten Kriterien annotiert wurden, eine bessere Datengrundlage dar. Aus diesem Grund werden die Rohdaten von Hand oder mittels *tagger* automatisch annotiert.⁷ Zur maschinellen Annotation steht im Prinzip die gesamte Palette der existierenden Programmiersprachen zur Verfügung, wobei Scripting-Sprachen aus dem UNIX-Umfeld (zu denen auch komplette Programmiersprachen wie zum Beispiel Perl zu zählen sind) sehr gut geeignet sind, eine Datenbasis schnell zu annotieren (vgl. LAWLER 1998). Die Ursache für den Einsatz dieser Sprachen liegt insbesondere darin, dass mit ihnen direkt auf den Textdaten gearbeitet werden kann. Andere Sprachen müssen die Textdaten vor der Annotation erst in einen anderen Datentyp konvertieren, um optimal eingesetzt werden zu können. Ist eine solche Konvertierung einmal erfolgt, dann ist es jedoch möglich, die Stärken dieser meist mächtigeren Programmiersprachen auszunutzen.

Integrierte Syntaxprüfung mit DSSSL

Programmiersprachen aus der LISP-Familie gehören zu den wichtigsten Programmiersprachen in der Computerlinguistik und der KI (vgl. u.a. GAZDAR & MELLISH 1991, STOYAN 1988, GÖRZ 1993). Da DSSSL eine definierte Untermenge von Scheme unterstützt, steht der Texttechnologie eine standardisierte Sprache zur Verfügung, die ihrer Bestimmung nach für SGML-annotierte Texte verwendet wird, aber, darüber hinaus, wie jeder andere LISP-Dialekt, sehr gut geeignet ist, computerlinguistische Aufgaben zu bearbeiten. Sämtliche in den LISP-Sprachen geschriebenen Programme können ohne große Probleme, möglicherweise sogar automatisch, in die streng funktionale Untermenge von Scheme überführt werden, die in DSSSL integriert ist. Dies bedeutet, dass quasi alle LISP-Programme in die Verarbeitung mit DSSSL eingebunden werden können. Für die

⁶ Das Kapitel 3 in MCENERY & WILSON (1996) beinhaltet eine Übersicht zu den statistischen Methoden für die Analyse "roher" Korpora.

⁷ Zu einer Fallstudie zum Für und Wider der beiden Methoden vgl. KÄLLGREN (1996).

Computerlinguistik heißt dies, dass die vielen existierenden LISP-Parser wiederverwendet werden können.⁸

Zusammenführung der Komponenten

Ein Annotationsprogramm, welches auf Funktionen einer Syntaxprüfung zugreifen kann, bietet eine Vielzahl potentieller Anwendungen. Eine einfache Beispielanwendung sei kurz vorgestellt:

```
Der Mann arbeitet. Er sieht den Mann mit  
dem Fernrohr. Der arbeitet Mann.
```

Dieser Beispieltext wird mittels des oben angegebenen Mechanismus in ein SGML-Dokument integriert. Ein Programm kann sich einer bestimmten Heuristik bedienen, um aus einem unstrukturierten Text die Sätze zu annotieren. Eine sehr einfache Herangehensweise ist, die Einheiten als Satz zu behandeln, die zwischen zwei Punkten (.) stehen. Dieses Textstück wird dann als `satz` annotiert.

```
<!DOCTYPE satzfolge SYSTEM "lang.dtd">  
<satzfolge>  
  <satz>Der Mann arbeitet.</satz>  
  <satz>Er sieht den Mann mit dem  
    Fernrohr.</satz>  
  <satz>Der arbeitet Mann.</satz>  
</satzfolge>
```

⁸ Zur Verdeutlichung findet sich auf der WWW-Seite zu diesem Artikel ein Miniprogramm zur Ermittlung der Anzahl der Lesarten eines Satzes, welches sowohl von einem *Scheme-Interpreter* als auch von einer *DSSSL-Engine* verarbeitet wird.

Der wichtige Teil des DSSSL-Programms, welches dafür sorgt, dass eine solche Annotation ermöglicht wird, ist nachfolgend aufgeführt.⁹

```
(make element
      gi: "satz"
      (literal (erster-satz text)))
```

Ein neues Element wird mit dem Konstrukt `make element` gebildet. Der Inhalt dieses Elementes ist der von der Funktion `erster-satz` gelieferte Textausschnitt, im vorliegenden Fall der Textausschnitt, der bis zum ersten möglichen Satzendezeichen (‘.’, ‘!’ oder ‘?’) reicht.

Wird dieser Textteil darüber hinaus an einen Lesartenzähler übergeben, kann dem einzufügenden Element `satz` ein Attribut, nämlich `lesarten`, hinzugefügt werden, welches als Wert die (nach der im *style-sheet* enthaltenen Grammatik) ermittelten Lesarten erhält. Die Anzahl der Lesarten wird in dem Beispielprogramm durch die Funktion `lesartenberechnung` ermittelt. Konkret sieht die Erweiterung der oben aufgeführten Anweisung um diese Komponente folgendermaßen aus:

```
(make element
      gi: "satz"
      attributes:
        `(("lesarten"
           ,(number->string
              (lesartenberechnung
                (erster-satz text))))))
      (literal (erster-satz text)))
```

⁹ Der dargestellte Ausschnitt verwendet einen nicht standardisierten Weg der Transformation von SGML-Dokumenten mittels der *style language* und nicht den Weg über die *transformation language*. Der Grund für dieses Vorgehen besteht darin, dass die DSSSL-Engine *jade* die *transformation language* nicht implementiert hat.

Wird das Annotationsprogramm auf den Beispieltext angewendet, ergibt sich die folgende Ausgabe:

```
<!DOCTYPE satzfolge SYSTEM "lang.dtd">
<satzfolge>
  <satz lesarten="1">
    Der Mann arbeitet.</satz>
  <satz lesarten="2">
    Er sieht den Mann mit dem Fernrohr.</satz>
  <satz lesarten="0">
    Der arbeitet Mann.</satz>
</satzfolge>
```

Eine praktische Anwendung würde sich mit dieser Markierung der Lesarten nicht zufrieden geben, sondern das Ergebnis der Lesartenprüfung weiter verwenden. So könnte z.B. eine genauere Überprüfung ergeben, dass einer der vermeintlichen Satzendepte ein Abk.-Punkt ist. Es ist auch denkbar, dass ein Programm, welches die Verständlichkeit des Textes überprüft, die ambigen Sätze besonders markiert.

Literatur

- GAZDAR, Gerald and MELLISH, Chris (1989): Natural language processing in LISP. Wokingham (u.a.): Addison-Wesley.
- GÖRZ, Günther (Hrsg., 1992): Einführung in die künstliche Intelligenz. Bonn (u.a.): Addison-Wesley.
- KÄLLGREN, Gunnel (1996): Man vs. Machine: Which is the Most Reliable Annotator? In: Giorgio PERISSINOTTO, Research in Humanities Computing, 5. Oxford: Clarendon Press.
- MCENEY, Tony and WILSON, Andrew (1996): Corpus Linguistics. Edinburgh: Edinburgh University Press.
- LAWLER, John (1998): "The Unix language family". In: John LAWLER and Helen Arister DRY, Using Computers in Linguistics. London und New York: Routledge.
- STOYAN, Herbert (1988): Programmiermethoden der künstlichen Intelligenz. Berlin (u.a.): Springer.

Standards

- CSS: LIE, Håkon Wium and BOS, Bert, Cascading Style Sheets, level 1. W3C Recommendation. 1996.
- DSSSL: ISO/IEC 10179:1996. Document Style Semantics and Specification Language. Genf 1996.
- HYTIME: ISO/IEC 10744-1992 (E). Information Technology — Hypermedia/Time-based Structuring Language (HyTime). Genf 1992/1996
- TEI: SPERBERG-MCQUEEN, C.M. and BURNARD, Lou (eds.), Guidelines for Electronic Text Encoding and Interchange. Chicago / Oxford: Text Encoding Initiative, 1994.
- XML: BRAY, Tim, PAOLI, Jean and SPERBERG-MCQUEEN, C.M.: Extensible Markup Language (XML) 1.0. W3C Recommendation. 1998.
- XSL: Extensible Stylesheet Language (XSL) Specification. W3C Working Draft, 21. April 1999.
- XSLT: CLARK, James, XSL Transformations Version 1.0. W3C Working Draft, 13. August 1999.