Die Chronik der „Chronik“ – Erfahrungen über die Konvertierung und Weiterverarbeitung proprietär annotierter Daten*

Georg Rehm und Markus Reinsch

Zusammenfassung

Die in einem datenbank-orientierten Format vorliegenden Substanzen der Reihe *Chronik des 20. Jahrhunderts* des Bertelsmann Lexikon Verlags werden automatisch in eine SGML/XML-basierte Annotation und daraufhin in die Hypertext Markup Language (HTML) transformiert. Für diese Aufgabe wurden verschiedene Werkzeuge implementiert, die die Rapid Prototyping-Möglichkeiten des Betriebssystems UNIX sowie dessen Funktionalität zur Kaskadierung ausnutzen. Des Weiteren wurde eine Text Mining-Komponente implementiert, die eine thematische Zuordnung der Artikel vornimmt und die auf bereits vorhandenen SGML-annotierten Daten basiert.

20.1. Einleitung

In der anwendungsbezogenen Literatur zu den Themen XML/SGML, Texttechnologie und Electronic Publishing steht meist die Frage nach dem Mehrwert von SGML (ISO8879, 1986, Goldfarb, 1990) als standardisiertem Format zur Datenhaltung und -verarbeitung im Vordergrund (etwa in Möhr und Schmidt, 1999). Dabei wird häufig die Frage außer Acht gelassen, auf welche Weise vorhandene Daten, die in einem proprietären Format vorliegen, in einen SGML-basierten Produktionsprozess integriert werden können. Dieser Beitrag erläutert – basierend auf dem Vorhaben, etwa 100 MB textuelle Daten vollautomatisch von einem datenbankorientierten Format in eine SGML-basierte Annotierung zu überführen – eine derartige Migration und geht vor allem auf die Frage ein, welche Methoden und Werkzeuge zur Konvertierung und Weiterverarbeitung der Daten ausgewählt bzw. entwickelt wurden und welche Probleme zu bewältigen waren.

Die Datengrundlage bilden die Einzelband-Substanzen der Jahre 1900 bis 1998 der Reihe *Chronik des 20. Jahrhunderts* der Firma Bertelsmann Lexikon Verlag, die – annotiert in dem Format BEES (*Bertelsmann Encyclopaedic Editorial System*) – strukturerhaltend und -erweiternd in ein SGML-basiertes Format überführt werden sollten. Die SGML-annotierten Daten sollten daraufhin weiteren Verarbeitungsschritten unterzogen werden, u. a. einer Klassifizierung hinsichtlich des Themengebietes sowie einer XSL-basierten Aufbereitung für das Wissensportal `wissen.de`.

* Erschienen in: *Proceedings der GLDV-Frühjahrstagung 2001*, Henning Lobin (Hrsg.), Universität Gießen, 28.–30. März 2001, Seite 211–221. `http://www.uni-giessen.de/fb09/ascl/gldv2001/`

```
1  #SW(STKA1) 19710901
2  #D8 71-148
3  #PR(PRKA1) [KALT1]1971[$][KALM1]September[$]
4  #AT(ATKA1) [TAG0]1.[$][MON0]September[$][JAHR0]1971[$][TNAM0]Mittwoch[$]
5  #TE(TEKA1) Das Bundesausbildungsförderungsgesetz (BAföG) tritt in Kraft.
6  Es regelt die staatliche Förderung der Schul- und Hochschulbesucher.
7  [SAV]S. 156[$]
```

Abbildung 20.1.: Ausschnitt aus der BEES-Quelle (*Chronik des Jahres 1971*, verschiedene Zeilenumbrüche und korrekte Umlaute wurden aus Gründen der Lesbarkeit manuell eingefügt)

20.2. Format der Ausgangsdaten: BEES

Das datenbankgetriebene *Bertelsmann Encyclopaedic Editorial System* (BEES) definiert ca. 400 verschiedene Auszeichnungselemente, die sowohl strukturelle als auch typographische Funktionen aufweisen. Etwa ein Drittel dieser Elemente wird, in Kombination mit etwa 140 verschiedenen Sonderzeichen (etwa `|o_ti|` für „õ“, `|A_um|` für „Ä“), in den Substanzen der *Chronik*-Reihe eingesetzt, wobei jeder Artikel der Chronik – Abb. 20.1 zeigt als Beispiel einen Eintrag des Kalendariums aus der Substanz *Chronik des Jahres 1971* – einige identische BEES-Elemente besitzt: Das Element `#SW` kennzeichnet beispielsweise das Schlüsselwort eines Standardartikels, im Falle der Chronik ist dies immer eine Datumsangabe. Das Element `#SW(STKA1)` markiert das Schlüsselwort (auch hier immer eine Datumsangabe) eines Kalendariumeintrags. Eine zusätzliche Kopfzeile bzgl. eines Geburtstages gibt das getypte Element `#PR(PRGE1)` an, ein Todestag wird mit `#PR(PRST1)` vermerkt. Innerhalb des Kalendariums wird ein sog. Präfix gesetzt, um Einträge zu gruppieren; die Abbildung zeigt `#PR(PRKA1)`: *Präfix Absatz Kalendarium Typ 1*.[1] Das Element `[KALT1]` markiert eine Jahresangabe in der Schlagzeile eines Kalendariumeintrags, `[KALM1]` markiert eine Monatsangabe. Das generische Element `[$]` beendet derartige „innere“ Elemente (die durch eckige Klammern markiert werden), was die Identifizierung von Elementgrenzen – vor allem im Fließtext bei typographischen Elementen und Querverweisen – sehr erschwert hat. Das Element `#AT(ATKA1)` markiert den *Abschnittstitel Kalendarium Typ 1*, die Suffixe in den darauf folgenden Elementnamen bedeuten *wird in der Printfassung gedruckt* (Suffix `1`) bzw. *wird nicht gedruckt* (Suffix `0`).

Der eigentliche Fließtext eines Artikels wird durch das Element `#TE` eingeleitet. Das Beispiel zeigt die Variante `#TE(TEKA1)`, *Absatzelementlayout Text Kalendarium Typ 1*. Ein `#TE`-Element sowie eine der vielen Ausprägungen des Elements `#SW` sind obligatorisch. Die Reihenfolge der einzelnen BEES-Elemente eines Eintrags sowie deren Verankerung am Zeilenanfang sind signifikant; Abschnittsgrenzen im Text werden durch einen Zeilenumbruch markiert. Innerhalb des Elements `#SF`, das den Titel eines Artikels enthält und das nicht bei Kalendariumseinträgen eingesetzt wird, ist des Weiteren die Angabe typographischer Informationen (Type, Schriftgrad und -schnitt etc.) vorgesehen. Abschließend befindet sich in dem Beispielartikel ein *Siehe auch Verweis*, der durch das Element `[SAV]` markiert wird.

[1] Diese hier in kursiver Schrift wiedergegebenen Erläuterungen stammen aus den Lexikon Verlag-internen Dokumenten *Typing Guidelines for the "Chronicle of the 20th Century"* und aus der *Elementlayoutliste*.

	Pentium 2 333 MHz 192 MB RAM	Pentium 3 850 MHz 1024 MB RAM
Konvertierung BEES → SGML pro Dokument (ø-Größe: 990 394 Bytes)	19:48 Min 12,0 Sek	7:55 Min 4,8 Sek
Validierung der SGML-Dokumente pro Dokument (ø-Größe: 2 103 466 Bytes)	6:07 Min 3,7 Sek	2:08 Min 1,3 Sek
Konvertierung SGML → HTML pro Dokument (ø-Größe: 1 580 609 Bytes)	41:33 Min 25,2 Sek	19:15 Min 11,7 Sek

Tabelle 20.1.: Dauer der verschiedenen Konvertierungsstufen bei insgesamt 99 Dokumenten (*Chronik des Jahres 1900* bis *1998*) auf zwei unterschiedlichen Systemen

20.3. Konvertierung: Der UNIX-Werkzeugkasten

Die Konvertierung der Substanzen wurde vollständig unter Linux (RedHat 6.0) mit *Open Source* Software (vgl. DiBona et al., 1999, Rehm und Lobin, 2000) durchgeführt; vgl. Tab. 20.1 für einen Überblick bzgl. der Größe des Datenbestandes und der Dauer der einzelnen Konvertierungsschritte. Als Mitglied der UNIX-Betriebssystemfamilie verfügt Linux über eine sehr umfangreiche und flexible Sammlung von Werkzeugen zur Textmanipulation, die man in ihrer Gesamtheit – unter Einbeziehung der umfangreichen Fähigkeiten der Shell zur Dateimanipulation – auch als *UNIX-Werkzeugkasten* bezeichnet (vgl. Kernighan und Pike, 1986). Durch das Hintereinanderschalten spezialisierter Werkzeuge kann man in effizienter Weise völlig neue Funktionalitäten schaffen (Gancarz, 1995, McKelvie et al., 1998). Auf diese Weise konnten Teile der Konvertierungssteuerung (z. B. die Aktivierung der Systemkomponenten und anschließende Validierung des Ergebnisses mittels eines SGML-Parsers) innerhalb kürzester Zeit mithilfe einiger Shell-Skripte sowie des Tools *make* automatisiert werden. Das eigentliche Transformationssystem wurde mithilfe einer Reihe kaskadierter und modularer Werkzeuge in *C* und dem Scanner-Generator *flex* implementiert. Diese Werkzeuge wurden als Filter entworfen, d. h. sie halten sich an die schlichte, aber effektive Konvention praktisch aller UNIX-Werkzeuge, eine Eingabe zu erwarten und diese in modifizierter Form wieder auszugeben. Somit waren die einzelnen Quelldateien der Komponenten überschaubar und Fehler des Systems durch spezielle Diagnoseausgaben der Module einfacher lokalisierbar. Insgesamt enthält die Kaskade 12 hintereinander geschaltete Werkzeuge, von denen 10 speziell für dieses Projekt entwickelt wurden. Als Eingabe erwartet die Kaskade BEES-Daten, als Ausgabe produziert sie SGML-annotierte Daten (Abb. 20.2, S. 214, zeigt das Beispiel aus Abb. 20.1 nach der Konvertierung). Abb. 20.3 auf S. 215 zeigt eine Architekturskizze des Systems, das drei Verarbeitungsstufen umfasst, die im Folgenden erläutert werden.

```
1   <article id="YB71ART958" base="YB71_007.html" aka="stka1" type="calendar">
2           <date>
3                   <year>
4                           <data>1971</data>
5                   </year>
6                   <month>
7                           <data>09</data>
8                   </month>
9                   <day>
10                          <data>01</data>
11                  </day>
12          </date>
13          <d8>
14                  <year>
15                          <data>71</data>
16                  </year>
17                  <page page="148">
18                          <data>148</data>
19                  </page>
20          </d8>
21          <pr type="cal" print="yes">
22                  <caltitle print="yes">
23                          <data>1971</data>
24                  </caltitle>
25                  <calsubtitle print="yes">
26                          <data>September</data>
27                  </calsubtitle>
28          </pr>
29          <at type="cal1" print="yes">
30                  <attitle>
31                          <date>
32                                  <day print="no">
33                                          <data>1.</data>
34                                  </day>
35                                  <month print="no">
36                                          <data>September</data>
37                                  </month>
38                                  <year print="no">
39                                          <data>1971</data>
40                                  </year>
41                                  <dayname print="no">
42                                          <data>Mittwoch</data>
43                                  </dayname>
44                          </date>
45                  </attitle>
46          </at>
47          <te type="cal1"><data>Das Bundesausbildungsf&ouml;rderungsgesetz
48          (BAf&ouml;G) tritt in Kraft. Es regelt die staatliche F&ouml;rderung
49          der Schul- und Hochschulbesucher.</data><ref type="seealso"><page
50          page="156"><data>156</data></page></ref></te>
51  </article>
```

Abbildung 20.2.: Ausschnitt aus der automatisch erzeugten SGML-Version der *Chronik des Jahres 1971* (der Text innerhalb des Elements `<te>` wurde aufgrund der vorliegenden Platzbeschränkungen manuell formatiert)

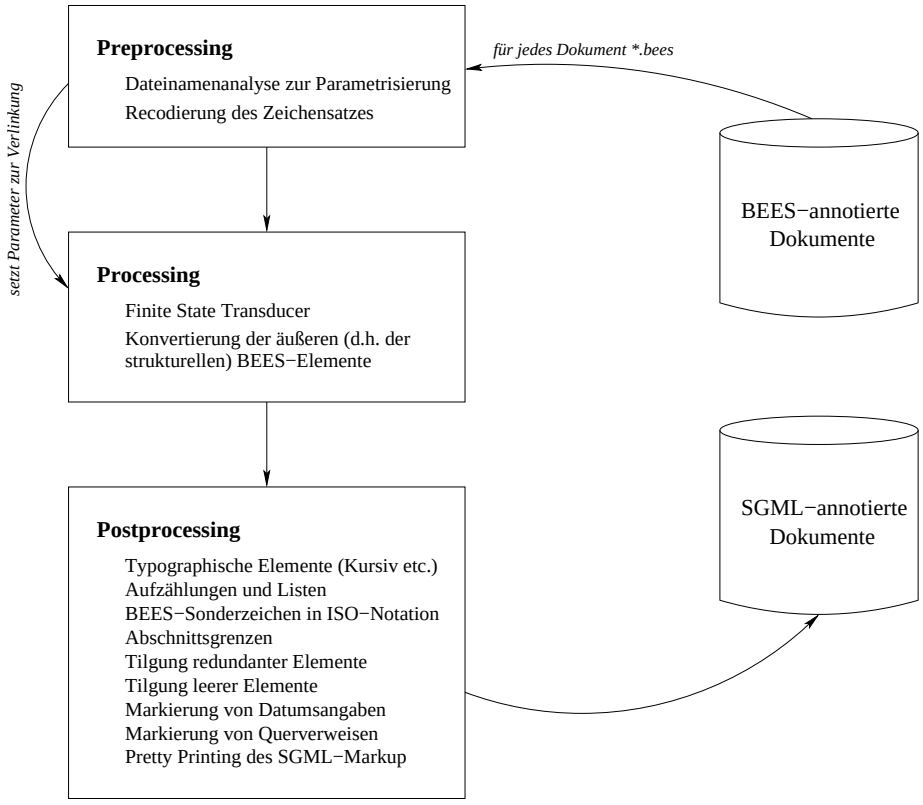

Abbildung 20.3.: Architektur des Konvertierungssystems

Die sich unmittelbar anschließende Validierung der automatisch erzeugten SGML-Dokumente hat sich bei der Entwicklung der einzelnen Module erneut (Rehm, 1999) als sehr hilfreich erwiesen. Die anfänglich noch rudimentäre Dokumenttyp-Definition (DTD), die parallel zur Entwicklung des Systems aus den BEES-annotierten Substanzen abgeleitet wurde, konnte fehlerhafte Konvertierungsregeln sofort aufdecken, da sie unmittelbar zu Fehlern bei der Validierung durch den SGML-Parser *nsgmls*[2] geführt haben. Des Weiteren wurden im Vergleich zu XML keine speziellen SGML-Konstrukte benutzt, sodass die durch die Konvertierung entstehenden SGML-Dokumente zugleich valide XML-Dokumente sind.

20.3.1. Vorverarbeitung

Die Vorverarbeitung der BEES-annotierten Daten umfasst einerseits die Analyse und Aufbereitung des Dateinamens der jeweiligen Substanz, andererseits die Recodierung des verwendeten Zeichensatzes. Der Dateiname, etwa `YB71_007.TXT` (*Yearbook 1971*), bestimmt hierbei zum einen den Dateinamen der SGML-Version, wobei lediglich das Suffix geändert wird, zum anderen wird er für die Parametrisierung der Hauptkomponente in Shell-Variablen abgelegt, um eine korrekte Verknüpfung in der HTML-Version einer Substanz zu gewährleisten (vgl. Zeile 1 in Abb. 20.2). Die Recodierung des Zeichensatzes erfolgt mithilfe des Werkzeugs `recode`, das Teil der GNU-Programmsammlung (vgl. etwa Rehm und Lobin, 2000) ist. Im vorliegenden Fall wird von einer *Codepage 850*-Kodierung in das *Latin 1*-Format konvertiert, das bei praktisch allen UNIX-Varianten den Standard darstellt.

20.3.2. Verarbeitung

Die sich anschließende Verarbeitungsstufe stellt die eigentliche Verarbeitung der BEES-annotierten Daten in ein SGML-basiertes Format dar. Zu diesem Zweck wurde, wie bereits erwähnt, ein Finite State Transducer in C mithilfe des Scanner-Generators *flex*[3] (vgl. Levine et al., 1992) implementiert, der alle `#`-Elemente (z. B. `#SW`, `#AT`, `#PR` etc.) in ihre SGML-Entsprechungen konvertiert.[4] Sobald der Präfix eines solchen strukturellen BEES-Elements erkannt wird (etwa `#SW` für verschiedene Ausprägungen wie `#SW(STKA1)` usw.), springt der Automat zu einem speziellen Regelwerk zur Verarbeitung dieser Elementfamilie, woraufhin die jeweilige Ausprägung erkannt und in einen spezielleren Zustand gesprungen wird. Dieses Prinzip wurde einerseits angewendet, um für die etwa 70 verschiedenen Zustände exklusive Regeln zu ermöglichen und andererseits, um eine Modularisierung vorzunehmen. Diese ermöglicht die Implementierung genau eines Zustands zur Verarbeitung von Datumsangaben, die in den Substanzen in zahlreichen Varianten enthalten sind.

[2] Entwickelt von James Clark, siehe `http://www.jclark.com`.

[3] Regeln in Quelldateien für den Scanner-Generator *flex* haben die folgende schematische Form:

```
[Zustand]regulärer_Ausdruck { C-Programmtext; }
```

Derartige Regeln profitieren also einerseits von der Mächtigkeit erweiterter regulärer Ausdrücke (vgl. Friedl, 1997), andererseits von den umfangreichen Möglichkeiten der Programmiersprache C.

[4] Ein zentraler Grundsatz bei der Umsetzung war die Aufnahme *aller* in den Substanzen enthaltenen Informationen, d. h. auch leere oder redundante BEES-Elemente werden in die SGML-Version übernommen. Im Zuge des Postprocessing der Daten kann das entsprechende Modul zur Tilgung dieser redundanten Informationen nicht in die Verarbeitungskaskade aufgenommen werden, um die redundanten Informationen in den SGML-Dokumenten beizubehalten.

20.3.3. Nachverarbeitung

Zum Zwecke der Nachverarbeitung der noch nicht vollständig erzeugten SGML-Dokumente wurden acht Module implementiert, die spezielle Aufgaben übernehmen: Ein Modul konvertiert alle explizit angegebenen typographischen Elemente (z. B. `[K]` für Kursivschrift, `[H]` für Hoch-, `[T]` für Tiefstellung etc.). Weitere Module annotieren im Fließtext von Artikeln enthaltene Aufzählungen (Element `[AUFZ1]`) und Listen, konvertieren BEES-Sonderzeichen in (sofern existent) die korrespondierenden ISO 8859 Umschriften, markieren Abschnittsgrenzen und nehmen ein *Pretty Printing* der SGML-Dokumente vor, d. h. das SGML-Markup wird entsprechend der hierarchischen Strukturierung eingerückt (vgl. Abb. 20.2).

20.3.4. Konvertierung von SGML nach HTML

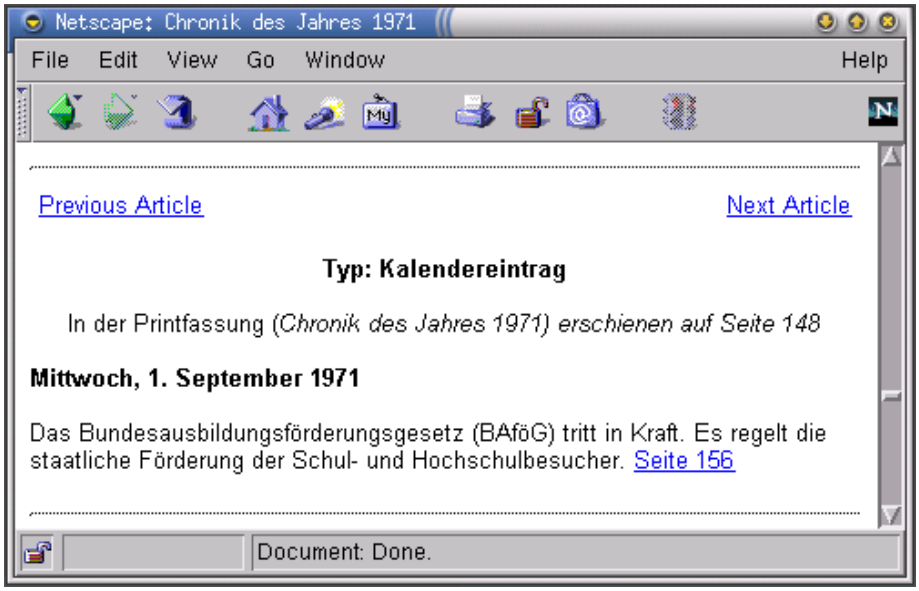

Abbildung 20.4.: Ausschnitt aus der HTML-Version der *Chronik des Jahres 1971*

Zur vereinfachten Verifikation der fehlerfrei annotierten SGML-Dokumente wurde weiterhin ein DSSSL-Stylesheet (ISO10179, 1996) implementiert, das die SGML-Quellen – ebenfalls zentral gesteuert über den eingangs erwähnten Shell-Skript- und Makefile-getriebenen Mechanismus – automatisch in die Hypertext Markup Language (HTML) konvertiert. Auf diese Weise konnten die Dokumente mit einer ansprechenden Oberfläche, d. h. einem beliebigen Webbrowser (vgl. Abb. 20.4), überflogen werden, um eventuelle Fehler in der Konvertierung zu visualisieren (vgl. Rehm, 1999), die sich nicht auf das SGML-Format, sondern auf die Interpretation der (in gelegentlichen Fällen undokumentierten) BEES-Elemente beziehen.

20.4. Weiterverarbeitung

Das Ziel der Weiterverarbeitung ist die automatische Einordnung der Chronik-Artikel in einzelne Kategorien wie *Wirtschaft*, *Politik*, *Sport* oder *Technik*. Eine Kategorisierung erlaubt dem Anwen-

der insbesondere eine gezieltere Suche. Ein Benutzer, der z. B. nach politischen Ereignissen in der Stadt Berlin sucht, wird die Suche auf die Kategorie *Politik* einschränken, wohingegen eine allgemeine Suche auch sportliche oder gesellschaftliche Ereignisse fände.

Bestehende Systeme zur Textkategorisierung arbeiten in erster Linie mit statistischen Methoden (siehe Aas und Eikvil, 1999). Anhand der Vorkommen der Wörter im Text lässt sich ein Muster berechnen, das als Eingabe für eine Mustererkennungs-Funktion dient (beispielsweise ein neuronales Netz). Diese Algorithmen müssen trainiert werden, indem der Anwender bereits kategorisierte Trainingsdokumente vorgibt. In den Beschreibungen der kommerziellen Systeme (z. B. *Intelligent Miner for Text*[5]) lässt sich leider nicht erkennen, welchen genauen Ansatz die Hersteller verfolgen. Da diese Systeme jedoch in der Regel ebenfalls trainiert werden müssen, ist davon auszugehen, dass hier ähnliche Methoden verwendet werden.

20.4.1. Erstellung einer Datengrundlage

Ein zentrales Prinzip des hier beschriebenen Ansatzes ist, den in SGML-annotierter Form und in einer Oracle-Datenbank vorliegenden, bereits kategorisierten Datenbestand des Bertelsmann Lexikon Verlags zur Weiterverarbeitung der Chronik-Substanzen einzusetzen. Aus diesem Datenbestand werden die Wörter oder Wortfolgen entnommen, die sich auf eine oder mehrere Kategorien abbilden lassen (*Gerhart Hauptmann* wird beispielsweise dem thematischen Konzept *Literatur* zugeordnet). Ausgewählt wurden zunächst Einträge von Personen (ca. 25 000) und Organisationen (ca. 1 700). Geplant ist, auch geographische Begriffe (ca. 16 000) und andere Eigennamen (z. B. Veranstaltungen) aufzunehmen. Zur Ergänzung dient eine manuell zusammengetragene Liste von Schlüsselwörtern, die häufig in den Artikeln zu finden sind (z. B. *Minister*, *Fußball*, *Theater*).

Wortfolgen, die optionale Elemente enthalten, müssen in der Datenbasis speziell bearbeitet werden (z. B. *Otto Fürst von Bismarck*, *Karstadt AG*). Die Kennzeichnung der optionalen Elemente kann zwar nur halbautomatisch vorgenommen werden, sie betrifft jedoch nur einen kleinen Anteil der Einträge (z. B. ca. 2 000 adelige Personen).

20.4.2. Verarbeitung der Artikel

Die Verarbeitung der Artikel wird von einem vollständig in Java implementierten Programm vorgenommen, das über JDBC auf die Oracle Datenbank zugreift. Das Parsen der XML-Struktur wird von einem SAX-Parser durchgeführt.

Bei der Verarbeitung werden die Wörter des Artikels nacheinander abgearbeitet. Im ersten Schritt werden die für die Kategorisierung uninteressanten Wörter anhand einer Stoppwortliste gefiltert. Im zweiten Schritt wird ein Pfad in einer Baumstruktur (s. folgender Abschnitt) gesucht. Bei einem Erfolg werden die gefundenen Informationen annotiert.

Im letzten Schritt werden die Wörter, die weder von Schritt eins noch von Schritt zwei behandelt wurden, auf Schlüsselwörter (beispielsweise *Minister*, *Fußball*, *Theater*) untersucht. Dabei wird getestet, ob die zu verarbeitenden Wörter Schlüsselwörter enthalten (z. B. *Außenministerium* enthält *Minister*).

[5] `http://www-4.ibm.com/software/data/iminer/fortext/index.html`

```
1   Dort fand z.B. der <headword topic="Literatur">Dichter</headword>
2   <person topic="Literatur">Gerhart Hauptmann</person> schon 1890
3   auf <headword topic="Sport">Skiern</headword> seinen ...
4   ... &uuml;ber den neuen
5   <headword topic="Sport">Wintersport</headword>.
6   Es tr&auml;gt den Titel ‘‘Alpine Lilienfelder
7   <headword topic="Sport">Skilauftechnik</headword>’’.
```

Abbildung 20.5.: Ausschnitt eines annotierten Chronik-Artikels: Die (hier vereinfacht dargestellten) Kategorien der einzelnen Wörter werden im Text mit XML/SGML-Markup annotiert.

20.4.3. Aufbau einer Datenstruktur

Ein Problem bei der Verarbeitung der Artikel sind Stichwörter, die aus mehreren zum Teil optionalen Teilen bestehen. Eine Datenbankabfrage mit Wildcards bietet hier keine ausreichende Hilfe.

Daher wird aus dem Datenbestand eine baumähnliche Datenstruktur aufgebaut, die hier als `RecursiveHashtable` bezeichnet wird (siehe Abb. 20.6). Beim Aufbau der Struktur werden für Wortfolgen mit optionalen Elementen mehrfache Einträge vorgenommen, sodass jeder mögliche Pfad wieder zu finden ist. Alle terminalen Knoten werden mit Kategorien versehen, wobei bei ambigen Einträgen mehrere Kategorien an einem Knoten vermerkt werden. Bei der Verarbeitung der Texte werden die längsten Pfade vom Wurzelknoten zu einem terminalen Knoten gesucht. Um eine möglichst optimale Performanz zu erreichen, werden die Einträge im `RecursiveHashtable` erst im Verarbeitungsprozess aus der Datenbank erstellt und zur späteren Wiederverwendung im Cache gehalten.

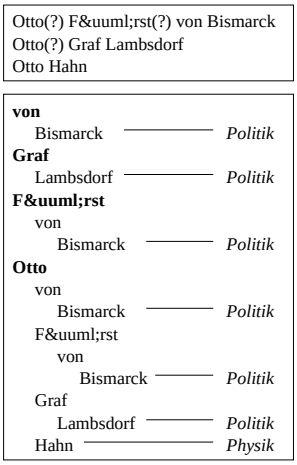

Abbildung 20.6.: Die Umsetzung der Wortfolgen in die Datenstruktur. Optionale Elemente sind mit (?) gekennzeichnet.

20.4.4. Auswertung der Annotierungen

In der abschließenden Auswertung werden alle zum Artikel zugeordneten Kategorien zu einem Vektor zusammengezogen. Der Vektor für den Beispielsatz aus Abb. 20.5 würde demnach die Kategorien *Literatur* und *Sport* erhalten. Anhand dieses Vektors wird abschließend die Kategorie des Artikels bestimmt. Im Beispiel wurden dem Text zwei nicht vereinbare Kategorien zugeordnet. Der Algorithmus muss sich daher auf eine Kategorie festlegen. Zum einen geschieht dies über die Häufigkeit der genannten Kategorien. In dem gesamten Artikel überwiegt klar die Anzahl der mit Sport kategorisierten Wörter. Zum anderen wird der Algorithmus so konfiguriert, dass die Kategorie von Personen im Zusammenhang mit Sport-Stichwörtern eine geringe Auswirkung hat. Vorstellbar wären auch Politiker, die eine Sportveranstaltung besuchen. Wenige Vorkommen oder widersprüchliche Kategorien steigern das Maß der Unsicherheit. Der Algorithmus kann jedoch leicht unsichere Kategorisierungen erkennen und darauf hinweisen.

20.5. Schlussfolgerungen

Die Wahl von UNIX als Basis des Konvertierungssystems hat sich bewährt: Die effiziente und Fehler unmittelbar aufdeckende Kaskadierung sowie die zahlreichen Hilfsprogramme ermöglichten die schnelle Entwicklung eines Prototyps. Etwa sechs Monate nach dessen Fertigstellung wurde ein weiterer Band, die *Chronik der Stadt Berlin*, mit minimalen Änderungen des Systems transformiert und erfolgreich validiert.[6]

Die Verwendung der bereits bestehenden Daten zur Klassifikation erspart ein aufwändiges Training und gewährleistet, dass besonders eindeutig kategorisierbare Wörter mit in die Entscheidung einfließen. Da sich die in den Chronik-Artikeln genannten Personen und Eigennamen in der Regel in den Lexikoneinträgen wieder finden, eignet sich die Datenbasis aus kategorisierten Lexikonartikeln sehr gut als Grundlage zur Verarbeitung der Chronik-Artikel. Problematisch bei der Kategorisierung sind kurze Artikel, da hier nur wenige Wörter zur Entscheidung beitragen, und einzelne Ausreißer das Ergebnis verfälschen können. Diese Problemfälle lassen sich jedoch leicht erkennen und sind somit kontrollierbar.

Literaturverzeichnis

Aas, Kjersti und Eikvil, Line (1999): "Text Categorisation – A Survey". Technischer Bericht 941, Norwegian Computing Center. Online verfügbar: `http://www.nr.no/research/samba/textmining.html`.

DiBona, Chris; Ockman, Sam und Stone, Mark (Herausgeber) (1999): *Open Sources: Voices from the Open Source Revolution*. Peking, Cambridge, Köln etc.: O'Reilly & Associates.

Friedl, Jeffrey E. F. (1997): *Mastering Regular Expressions*. Cambridge, Köln, Paris etc.: O'Reilly & Associates.

Gancarz, Mike (1995): *The UNIX Philosophy*. Boston, Oxford, Melbourne etc.: Digital Press.

Goldfarb, Charles F. (1990): *The SGML Handbook*. Oxford: Oxford University Press.

[6] Es mussten drei neue SGML-Elemente, ein neues Attribut und zwei neue Attribut-Werte eingeführt werden, um die strukturellen Eigenheiten dieser Substanz erfassen zu können.

ISO10179 (1996): "Information Processing – Processing Languages – Document Style Semantics and Specification Language (DSSSL)". Internationaler Standard, International Organization for Standardization, Genf. Online verfügbar: `http://www.ornl.gov/sgml/wg8/`.

ISO8879 (1986): "Information Processing – Text and Office Information Systems – Standard Generalized Markup Language". Internationaler Standard, International Organization for Standardization, Genf.

Kernighan, Brian W. und Pike, Rob (1986): *Der UNIX-Werkzeugkasten*. München, Wien: Hanser.

Levine, John R.; Mason, Tony und Brown, Doug (1992): *lex & yacc*. A Nutshell Handbook. Cambridge, Köln, Paris etc.: O'Reilly & Associates, 2. Auflage.

McKelvie, David; Brew, Chris und Thompson, Henry (1998): "Using SGML as a Basis for Data-Intensive NLP". Erweiterte Version des ANLP 97 Artikels. Online verfügbar: `http://www.ltg.hcrc.ed.ac.uk/~dmck/chum/thepaper.ps`.

Möhr, Wiebke und Schmidt, Ingrid (Herausgeber) (1999): *SGML und XML – Anwendungen und Perspektiven*. Berlin, Heidelberg, New York etc.: Springer.

Rehm, Georg (1999): "Automatische Textannotation – Ein SGML- und DSSSL-basierter Ansatz zur angewandten Textlinguistik". In: *Text im digitalen Medium – Linguistische Aspekte von Textdesign, Texttechnologie und Hypertext Engineering*, herausgegeben von Lobin, Henning, Wiesbaden: Westdeutscher Verlag, S. 179–195.

Rehm, Georg und Lobin, Henning (2000): "From Open Source to Open Information – Collaborative Methods in Creating XML-based Markup Languages". In: *Proceedings of Electronic Publishing 2000*. International Federation for Information Processing and International Council for Computer Communication, Kaliningrad, Svetlogorsk. Online verfügbar: `http://www.uni-giessen.de/~g91063/pdf/open-information.pdf`.