Gregor Büchel, Bernhard Schröder

Verfahren und Techniken in der computergestützten Lexikographie

Jedes Vorhaben, bei dem die Erfassung und Verarbeitung von Textdaten eine wichtige Rolle spielt, wird vor den Problemen stehen,

- in welcher Weise die Textdaten erfasst werden sollen,
- in welcher Form die Textdaten gespeichert werden sollen und
- mit welchen Hilfsmitteln das Textmaterial für die weitere Arbeit oder für eine Veröffentlichung des Materials erschlossen werden soll.

Dabei steht nicht mehr die Bewältigung großer Textmengen im Vordergrund der Überlegungen – textuelle Information, die dem Umfang ganzer Bücherregale entspricht, lässt sich heute mit jedem handelsüblichen PC und preisgünstig zu erwerbender oder gar kostenfreier Software verwalten –, sondern der Umgang mit *strukturierter* textueller Information. Was ist damit gemeint?

Für die allermeisten Formen der digitalen Repräsentation von Texten bildet die lineare Abfolge von Basiseinheiten des Textes – i.d.R. Zeichen – die Grundstruktur der Repräsentation. Einfache Textdateien bestehen schlicht aus einer Abfolge von Zeichenrepräsentationen (Zeichencodes). Neben der linearen Strukturierung, die sich in dieser Art der Kodierung wiederspiegelt, weisen Texte immer auch vielfältige andere explizite und implizite Strukturen auf, die hier pauschal als *nichtlineare* Strukturen bezeichnet werden sollen. Zu den nichtlinearen Strukturen in diesem weiten Sinne gehören:

- Formatierungen und Layout von Texten und Textteilen, Hervorhebung von Textteilen durch den Autor
- Metainformation zum Text, z.B. bibliographische Angaben oder andere Angaben zur Textherkunft, Angaben zur Digitalisierung, Angaben zu Korrekturen usw.,

- Textgliederung,
- interpretierende Angaben zum Text oder zu Textteilen (linguistischer, philologischer, historischer, soziologischer usw. Art),
- Verknüpfungen zwischen Texten oder Textteilen (darunter können explizit vom Autor vorgenommene Verknüpfungen wie Fußnoten oder explizite Querverweise sein oder von einem Interpreten vorgenomme Verknüpfungen) und
- Verknüpfungen mit nicht-textuellen Medien, wie statischen oder bewegten Bildern, Klangdaten, interaktiven Anwendungen, Internet-Ressourcen.

Die Verknüpfungsstrukturen werden auch als die in einem engeren Sinne nichtlinearen Strukturen bezeichnet.

Die explizite Kodierung der genannten nichtlinearen Strukturen wird i.d.R. von modernen Textverarbeitungsprogrammen – zumindest in Ansätzen – unterstützt. Die nichtlineare Kodierungsfähigkeit eines Systems wird allerdings oft bei retrospektiven Projekten zur Digitalisierung historischer Dokumente vor besondere Herausforderungen gestellt: Bei der retrospektiven Digitalisierung hat man es nicht selten mit unterschiedlichen Textzeugen und -versionen, Graden an Sicherheit bei der Entschlüsselung, Textlücken, Textfragmenten mit unklarer Anordnung usw. zu tun. Einerseits erfordert die Abbildung dieser Strukturen auf die von Textverarbeitungsprogrammen gebotenen Möglichkeiten viel Phantasie, und entsprechend lässt der Bedienungskomfort der Textverarbeitungsprogramme bei derartigen Anwendungen zu wünschen übrig. Andererseits wirft der Versuch, die erfassten Daten in unterschiedlichen Medien (Druck, WWW, CD-ROM) oder nur selektiv zu publizieren, gerade bei den nichtlinearen Strukturen nur schwer lösbare Probleme auf. Und schließlich scheitert man zumeist an der Aufgabe, nichtlineare Strukturen in andere Anwendungen oder Kodierungsformate zu exportieren.

Man wird aber auch schon bei linearen Strukturen auf Probleme stoßen, wenn man vor der Aufgabe steht, Zeichen zu kodieren, die in den üblichen Zeichensätzen des Betriebssystems nicht verfügbar sind. Die Darstellbarkeit solcher Zeichen auf dem Bildschirm ist nicht immer auch eine Garantie für die Druckbarkeit von Zeichen oder die Darstellbarkeit der Zeichen auf WWW-Seiten. Und den Export in andere Anwendungen überleben solche aus anderen Codes entliehenen Zeichen selten.

Lexikographische Projekte, wie sie in diesem Band geschildert werden, sind mit den aufgeführten Textkodierungs- und -strukturierungsproblemen in besonderer Weise konfrontiert: Zum einen stellen Lexikoneinträge stark nichtlinear strukturierte Texte dar; im gedruckten Wörterbuch wird durch typographische Mittel eine oft sehr filigrane Mikrostruktur erschließbar gemacht. Die implizite Kodierung der Mikrostruktur durch die Typographie reicht aber normalerweise nicht aus, wenn aus dieser Textgrundlage Wörterbuchversionen für das elektronische Medium abgeleitet werden sollen. Der Hauptgrund dafür ist die Polyfunktionalität typographischer Merkmale. Kursivierung kann in einem Wörterbuch unterschiedliche Funktionen haben; im für die digitale Verarbeitung günstigeren Fall, lässt sich die Funktion rein formal durch den Kontext ermitteln, im ungünstigeren Fall kann die Ermittlung der Funktion nur durch eine Interpretationsleistung der Wörterbuchbenutzerin oder des Wörterbuchbenutzers geschehen. Zur Implementierung unterschiedlicher Sichten auf Wörterbuchartikel in einem elektronischen Wörterbuch, die die Ausblendung von Teilen der Mikrostruktur gestatten, ist eine eindeutige explizite Markierung der Mikrostruktur vonnöten.

Zum zweiten weisen Wörterbücher in starkem Maße Verknüpfungsstrukturen auf. Es bestehen eine Vielzahl von Verweisen innerhalb der Artikel, zwischen Artikeln und von

Wörterbuchartikeln auf Belege. Zunehmend werden auch Verweise auf nicht-textuelle Medien integriert.

Und schließlich arbeiten alle Wörterbuchprojekte mit Korpora retrospektiv digitalisierter Texte. Neben den bereits erwähnten Kodierungsproblemen bei diesen Korpora, stellt sich spätestens bei der elektronischen Publikation eines Wörterbuchs die Frage, ob die Korpora oder ausgewählte Belegstellen zusammen mit dem Wörterbuch publiziert werden sollen und welche Verweisstrukturen zwischen dem Wörterbuch und den Korpora explizit gemacht werden sollen.

Um die angesprochenen Strukturierungs- und Kodierungsfragen zu beantworten und die damit verbundenen Erfassungs- und Publikationsprobleme zu lösen, müssen eine Vielzahl inhaltlicher und technischer Probleme gelöst werden; ein weites Spektrum an Fragen und an prinzipiellen und exemplarischen Lösungen wird in diesem Band angesprochen. In diesem Beitrag geht es um die Grundlagen der Kodierung strukturierter textueller Information mit SGML und XML und der Verwaltung dieser Strukturen in Datenbanksystemen.

1 Kodierung strukturierter Texte mit SGML und XML

In diesem Abschnitt wird diskutiert, welche Anforderungen an ein adäquates Kodierungssystem für strukturierte textuelle Information zu stellen sind. SGML und XML werden als Kodierungssysteme vor dem Hintergrund dieser Forderungen bewertet.

1.1 Anforderungen an ein Kodierungssystem

Je nach Projekt wird man mit unterschiedlicher Gewichtung fordern, dass das gewählte Kodierungssystem hinreichend mächtig ist, die zu kodierenden Strukturen eindeutig repräsentiert, gut maschinell zu interpretieren ist, dass die Kodierung nachhaltigen Bestand hat und auf andere Computersysteme portierbar ist, ferner dass benutzerfreundliche Werkzeuge zur Arbeit mit dem Kodierungssystem bereitstehen. Die folgenden Abschnitte sollen die Anforderungen verdeutlichen.

1.1.1 Mächtigkeit

Ein Kodierungssystem muss zunächst für die explizit zu kodierende Information auch Kodierungsmechanismen bereitstellen. Ein Kodierungssystem, das zur Kodierung rein linearer Strukturen dient, ist beispielsweise ungeeignet zur Kodierung von Verweisstrukturen. Das schließt nicht aus, dass man mit zusätzlichen Konventionen, z.B. durch Reservierung bestimmter Zeichensequenzen für die Kodierung von Verweisstrukturen, die Mächtigkeit eines Kodierungssystems erweitern kann. Die Etablierung zusätzlicher Konventionen bedeutet aber nichts anderes als die Implementierung eines neuen mächtigeren Kodiersystems auf der Basis eines weniger mächtigen. Geschieht diese Erweiterung *ad hoc*, so läuft man ein nicht zu unterschätzendes Risiko, dass die übrigen Anforderungen bezüglich des erweiterten Kodierungssystems nicht mehr erfüllt sind.

1.1.2 Eindeutigkeit

Gerade bei *ad hoc* entworfenen Kodierungssystemen kann es leicht passieren, dass die gewählten Kodierungen nicht eindeutig sind. Im trivialen Fall, der bei Texten, die in den Anfangszeiten der linguistischen Datenverarbeitung erfasst wurden, nicht selten anzutreffen ist, wurden in den damaligen Codes nicht verfügbare Zeichen durch Zeichensequenzen ersetzt, die auch sonst im Text anzutreffen waren, so dass eine automatische Erkennung dieser Zeichen im Code nicht immer möglich ist. Ob *zuende* in einem Kodierungssystem, bei dem alle Umlaute als nicht-umgelauteter Vokal + *e* kodiert werden, als zwei- oder dreisilbig zu interpretieren ist, ist nur unter Zuhilfenahme sprachlichen Wissens entscheidbar. Weniger triviale Fälle sind aber auch in moderneren Kodierungssystemen anzutreffen: Bei Kodierungen, die von reservierten Codes eingeleitet werden, ist nicht immer klar, wo sie enden. Kodierungen, die nichtlineare strukturelle Information zu Textteilen beinhalten, sind oft nicht eindeutig, wenn es um die Frage geht, wo diese Textteile beginnen und wo sie enden; beendet eine neue Kodierung desselben Typs immer den Gültigkeitsbereich der vorangehenden? Diese Frage wird nicht von jedem Kodierungssystem eindeutig beantwortet.

1.1.3 Interpretierbarkeit

Auch die formale Eindeutigkeit von Kodierungen muss nicht immer eine günstige maschinelle Interpretierbarkeit bedeuten. Konventionen können von den Bearbeiterinnen und Bearbeitern aufgrund des sprachlichen und außersprachlichen Wissens, über das sie verfügen, sehr leicht zu erlernen und zu interpretieren sein, ohne dass diese Konventionen deshalb auch leicht programmtechnisch umzusetzen sind. Dies ist immer dann der Fall, wenn es sehr komplexe Abhängigkeiten verschiedener Kodierungen voneinander gibt, eine Kodierung A beispielsweise im Kontext einer Kodierung B auf eine Weise, im Kontext einer Kodierung C aber auf eine ganz andere Weise verwendet wird.

1.1.4 Nachhaltigkeit

In diesem Band wird von Langzeitprojekten berichtet, die sich über viele Jahrzehnte erstrecken. Es ist selbstverständlich, dass die in diesen Projekten akkumulierten elektronischen Textressourcen für die Projektdauer und die elektronisch publizierten Ergebnisse möglichst lange darüber hinaus Bestand haben sollen. Es ist nicht allzu verwegen, zu prognostizieren, dass die Anwendungsprogramme, die heute die Benutzerschnittstellen zu Wörterbüchern auf CD-ROM oder im World-Wide Web (WWW) bereitstellen, nicht über Jahrzehnte hinweg benutzbar sein werden, sofern sie auf avancierteren Techniken beruhen als bloßen auf einem Server bereitliegenden HTML-Seiten. Die Analogie zur bisherigen Entwicklung berechtigt zu dieser Annahme. Nicht wenige nur ein Jahrzehnt alte Anwendungsprogramme sind auf heutigen PCs nur dann noch nutzbar, wenn neben aktuellen Betriebssystemsversionen auch alte DOS-Versionen installiert werden. Und manches alte Programm verweigert selbst dann die Zusammenarbeit mit neuer Hardware.

Hard- und Softwarehersteller haben nur ein geringes wirtschaftliches Interesse an der Gewährleistung einer nachhaltigen Abwärtskompatibilität neuer Systeme, also der Möglichkeit, ältere Software und Daten auf neueren Systemen zu nutzen. Die zusätzliche Forde-

rung weitreichender Abwärtskompatibilität verteuert nämlich die Hard- und Softwareentwicklung und verringert den Anreiz, die neuesten Softwareprodukte zu erwerben.

Noch bedrohlicher als die mangelnde Abwärtskompatibilität der Weiterentwicklungen eines bestimmten Systems kann für Anwendungssoftware das vollständige Verschwinden bestimmter Hardware- und Betriebssystemsplattformen vom Markt sein. Es ist schwer, für manche Anwendungssoftware, die auf in den 80er Jahren noch weit verbreiteten Großrechnern lief, heute noch geeignete Rechner zu finden.

Und schließlich entspricht veraltete Anwendungssoftware auch nicht mehr den heutigen Benutzungsgewohnheiten, ihre Bedienung ist oft nur umständlich zu erlernen, sie wird von den potentiellen Benutzerinnen und Benutzern nicht selten abgelehnt.

Kompatibilitätsprobleme stellen sich aber nicht nur bei Anwendungsprogrammen, sondern auch bei den zugehörigen Dateiformaten. Aktuelle Programmversionen von Textverarbeitungsprogrammen haben nicht selten Schwierigkeiten bei der Wiederherstellung aller Formatierungsinformationen aus Dateien, die mit Vorgängerversionen vor etwa fünf Jahren hergestellt wurden, an weiteren nichtlinearen Strukturen in diesen Texten scheitern sie oft genug. Eine Unterstützung von Formaten nicht mehr auf dem Markt befindlicher Textverarbeitungssysteme ist selten anzutreffen.

Vor diesem Hintergrund muss ein Kodierungsschema auch daraufhin untersucht werden, ob man berechtigterweise prognostizieren kann, dass es auch in einigen Jahrzehnten noch ohne großen Eigenaufwand zur Herstellung von Kompatibilität und ohne Informationsverlust zu verwenden ist.

1.1.5 Portierbarkeit

Will man sich in einem Projekt nicht ein für alle Mal auf bestimmte Hard- und Softwareprodukte festlegen, so stellt sich die Frage der Portierbarkeit der Daten. Können die Daten ohne großen Aufwand und ohne Informationsverlust auch auf anderen Systemen verwendet werden? Zwar ist Portierbarkeit meist auch in den Überlegungen zur Nachhaltigkeit eingeschlossen, aber es kommt auch vor, dass sich Nachhaltigkeitsüberlegungen auf die Annahme stützen, dass sich ein bestimmtes technisches System durchsetzen werde und nachhaltig verfügbar bleibe, z.B. aufgrund seiner technischen Überlegenheit oder der Marktmacht des Herstellers. So kommt es bei Projekten mit starkem Bezug zum WWW vor, dass bei der Entscheidung für ein bestimmtes Kodierungssystem den Ausschlag gibt, ob das Kodierungssystem eine Zukunft im WWW hat. Gleichzeitig kann es aber der Fall sein, dass dieses Kodierungssystemen von den konkurrierenden Browsern in sehr unterschiedlicher Weise unterstützt wird, von einzelnen Browsern also möglicherweise derzeit gar nicht unterstützt wird.

1.1.6 Softwareunterstützung

Die Arbeit mit komplexen Kodierungssystemen kann umständlich, ermüdend und fehleranfällig sein. Es ist für eine ergonomisch sinnvolle Erfassung der Daten wichtig, dass geeignete Softwarewerkzeuge, Spezialeditoren, zur Verfügung stehen. Die effiziente Auswertung der Daten kann wesentlich von der Verfügbarkeit von Recherchewerkzeugen abhängen, die sowohl die linearen als auch die nichtlinearen Textstrukturen nutzen. Und schließlich kann geeignete Publikationssoftware die Druck- oder CD-ROM-Versionen der zu veröffentlichenden Daten sehr erleichtern.

In den folgenden beiden Unterabschnitten werden SGML und XML als Formalismen, mit denen sich komplexe Kodierungssysteme definieren lassen, in Grundzügen vorgestellt, um dann zu diskutieren, inwiefern SGML- oder XML-basierte Formalismen den Anforderungen genügen.

1.2 Die Grundkonzepte von SGML und XML

SGML (Standard Generalized Markup Language) und XML (Extensible Markup Language) sind Formalismen,[1] mit denen sich Kodierungssysteme formal definieren lassen. Man kann SGML und XML als Metasprachen auffassen, mit denen spezielle Kodierungssysteme beschrieben werden. Diese speziellen Kodierungssysteme werden gerne als *Markup Languages* (Markierungs-/Annotierungssprachen) bezeichnet. Insofern als diese Bezeichnung suggeriert, dass die Aufgabe SGML- oder XML-basierter Kodierungssysteme grundsätzlich sei, Markierungs- oder Annotierungsmöglichkeiten für vorgegebene Texte zu definieren, ist sie ungenau. SGML- und XML-basierte Kodierungssysteme werden inzwischen auch zu anderen Zwecken als der Repräsentation textueller Strukturen verwendet, mit SGML und XML lässt sich die Syntax von Programmiersprachen ebenso definieren wie Datenaustauschformate für Datenbanken mit beliebigem Inhalt. Der Schwerpunkt von SGML- und XML-Anwendungen liegt jedoch nach wie vor auf der Repräsentation textueller Strukturen. Und nur darauf soll im Folgenden Bezug genommen werden.

SGML ist die ältere der beiden Metasprachen, sie wurde bereits 1986 definiert und als ISO-Norm 8879 standardisiert. Im folgenden soll zunächst überwiegend von SGML die Rede sein, aber bis auf weiter unten ausdrücklich vermerkte Unterschiede ist die Darstellung auch auf XML übertragbar.

Bei den Grundkonzepten von SGML sind die, die sich ausschließlich auf die bereitgestellten formalen Strukturen beziehen, von denen zu unterscheiden, die sich auf den intendierten Umgang mit den formalen Konzepten beziehen.

1.2.1 Die formalen Grundkonzepte

Der Aufbau eines Textes aus Teilen wird in SGML durch *Elemente* modelliert. Terminologisch soll hier zwischen Elementen als Typen von Textteilen und den konkreten Textteilen selbst als Instanzen dieser Elemente, kurz: *Elementinstanzen*, unterschieden werden. Das Element *abschnitt* kann also nach dieser Sprachweise mehrfach in einem Text vorkommen, jedes Vorkommen ist aber eine neue Instanz dieses Elements. Ein Text als ganzer bildet eine Elementinstanz. Jede Elementinstanz kann Textstücke und weitere Elementinstanzen enthalten. Die in einer Elementinstanz unmittelbar enthaltenen Elementinstanzen dürfen einander nicht überlappen.

Die Elementstruktur eines Textes entspricht also einer Baumstruktur, bei der das umfassendste Element die Wurzel, die enthaltenen Elemente die Zweige und die enthaltenen

---

[1] Wenn man sich im WWW über SGML und XML informieren möchte, bietet das Electronic Text Center an der University of Virginia einen guten Ausgangspunkt gerade im Bereich akademischer Anwendungen, s. http://etext.lib.virginia.edu/standard.html. Sehr viel weiterführende Information findet man auf den XML Cover Pages unter http://www.oasis-open.org/cover/sgml-xml.html.

Textstücke die Blätter bilden. Der Beginn und das Ende eines Elementes wird i.d.R. durch ein *Tag*, eine Marke, angezeigt, das durch die Markup-Begrenzer < und > vom umgebenden Text abgegrenzt ist. Im Anfangs-Tag steht direkt hinter dem linken Markup-Begrenzer der Name des Elementes, im End-Tag steht i.d.R. zwischen den Markup-Begrenzern ein Querstrich und der Name des zu beendenden Elementes. Hier ein Teil dieses Abschnitts mit Abschnitt, Sätzen und Nominalphrasen als Elementen:

```
<abschnitt>
<satz><np>Der Aufbau <np>eines Textes</np> aus <np>Teilen</np></np>
wird in <np>SGML</np> durch <np>Elemente</np> modelliert.</satz>
<satz><np>Ein Text als ganzer</np> bildet <np>ein
Element</np>.</satz>
...
</abschnitt>
```

Elemente können in SGML nun Träger von Eigenschaften sein. Zur Kodierung von Eigenschaften werden in den Anfangs-Tags von Elementen hinter dem Elementnamen Attribute mit zugehörigen Werten notiert. Wollen wir beispielsweise im obigen Abschnitt bei den Nominalphrasen auch deren Kasus und Numerus kodieren, so kann das wie folgt aussehen:

```
<abschnitt>
<satz><np kasus="nom" numerus="sg">Der Aufbau
<np kasus="gen" numerus="sg">eines Textes</np> aus
<np kasus="dat" numerus="pl">Teilen</np></np> wird in
<np kasus="dat" numerus="sg">SGML</np> durch
<np kasus="akk" numerus="pl">Elemente</np> modelliert.</satz>
<satz><np kasus="nom" numerus="sg">Ein Text als ganzer</np> bildet
<np kasus="akk" numerus="sg">ein Element</np>.</satz>
...
</abschnitt>
```

Vor der Kodierung der Gliederungsstruktur muss natürlich festgelegt werden, was kodiert werden soll. In einer Dokumenttypdefinition (DTD) geschieht der formale Teil der Festlegung: Es wird im Wesentlichen festgelegt, welche Elemente es gibt – im Beispiel *abschnitt*, *satz* und *np* –, welche Elemente in welchen anderen enthalten sein dürfen oder müssen, ob bestimmte Elemente unmittelbar Text enthalten können oder nur wieder weitere Elemente. Diese Spezifikation bezeichnet man auch als das Inhaltsmodell eines bestimmten Dokumenttyps. In der DTD wird ferner festgelegt, welche Attribute diese Elemente tragen können. Zu den Attributen wird angegeben, welchen Typs die Werte sind.

Je nach Kodierungszweck und Kodierungsstil kann es zu einem Dokument unzählige mögliche DTDs geben. Durch eine DTD ist nur bestimmt, welche Kodierungen zulässig sind, nicht aber, wie sie verwendet werden sollen. Dies sollte in Kommentaren zur DTD oder in einem separaten Kodierungshandbuch möglichst detailliert und praxisnah niedergelegt werden.

Aus dem hierarchischen SGML-Strukturierungsmodell für Dokumente ergibt sich eine nicht unwesentliche Einschränkung: Die gleichzeitige Strukturierung eines Dokumentes in einander überlappende Einheiten ist ausgeschlossen. Ein Dokument, das durch SGML-Elemente, die Abschnitte umfassen, gegliedert wird, kann nicht gleichzeitig durch SGML-Elemente gegliedert werden, die Seiten umfassen, sofern mehrere Abschnitte auf einer Seite vorkommen können, gleichzeitig aber auch ein Seitenumbruch innerhalb eines Abschnitts vorkommen kann. Denn in solchen Fällen kann weder die betreffende Seitenelementinstanz

als Teilelement der Elementinstanz des Abschnitts noch umgekehrt aufgefasst werden. Zwar sieht der SGML-Standard für solche Fälle die Möglichkeit vor, mit konkurrierenden DTDs zu arbeiten, also eine DTD bereitzuhalten, in der die Seitengliederung, nicht aber die Abschnittsgliederung berücksichtigt wird, und eine andere DTD für den umgekehrten Fall. In das Dokument werden speziell markierte Tags für beide DTDs aufgenommen. Leider gibt es jedoch kaum Werkzeuge, die die gleichzeitige Arbeit mit konkurrierenden DTDs unterstützen würden.

Als Ausweg aus dem Dilemma bietet sich an, bei einer der beiden konkurrierenden Gliederungsarten darauf zu verzichten, mit SGML-Elementen die Einheiten dieser Gliederungsebene zu *umschließen,* sondern stattdessen mithilfe leerer, d.h. weder Text noch weitere Elemente umfassender, Elemente nur den Anfang oder nur das Ende oder beides zu markieren. Damit verzichtet man allerdings auf Möglichkeiten, die zulässigen Strukturen dieser Gliederungsart genauer durch eine DTD zu beschreiben und die Gliederungseinheiten in bequemer Weise zu adressieren.[2]

Die bisher behandelten Mittel zur Kodierung von Textstrukturen erlauben, über den Text ein baumartige Gliederungsstruktur zu legen. Zwischen den Elementinstanzen sowie den Elementinstanzen und dem Text bestehen zwei Basisrelationen: Eine Elementinstanz oder ein Textstück kann in einer (anderen) Elementinstanz unmittelbar enthalten sein. In einer Elementinstanz unmittelbar enthaltene Elementinstanzen oder Textstücke stehen in einer linearen Abfolgerelation zueinander: Die unmittelbar enthaltenen Elementinstanzen oder Textstücke stehen eben immer rechts oder links von anderen Elementinstanzen oder Textstücken.

Zu der baumartigen Gliederungsstruktur tritt noch die Möglichkeit, Elementinstanzen mit Attributen und zugehörigen Werten zu versehen. Der Mechanismus der Attribuierung von Elementinstanzen wird in SGML dazu genutzt, weitere beliebige Relationen zwischen den Elementinstanzen eines Dokumentes oder auch zu Elementinstanzen anderer Dokumente herzustellen.

Um beispielsweise einen Verweis von einem Wörterbuchartikel auf einen anderen zu kodieren, kann man ersteren eindeutig benennen, indem man der zu diesem Artikel gehörenden Elementinstanz ein Attribut mit einem Identifikator, einer zur Bezeichnung keiner anderen Elementinstanz verwendeten Zeichenkette zuordnet.

```
<artikel id=hund><lemma>Hund</lemma>
...
</artikel>
...
<artikel id=katze><lemma>Katze</lemma>
siehe auch <verweis idref=hund>Artikel Hund</verweis>
...
</artikel>
```

Die Elementinstanz von *verweis* im Artikel Katze verweist durch den Wert des Attributs *idref* also auf die Elementinstanz von *artikel* mit dem Identifikator *hund.* Führt man zusätzlich noch Konventionen zur eindeutigen Identifikation anderer Dokumente ein, wie es im WWW durch die Uniform Resource Identifiers (URIs) realisiert ist, so können Verweise auf beliebige Elementinstanzen in beliebigen SGML-Dokumenten realisiert werden. Mit-

---

[2] Vgl. hierzu auch den Beitrag von Thomas Burch und Johannes Fournier in diesem Band. Die TEI befasst sich im Kapitel 31 der Guidelines mit diesem Thema.

hilfe der für XML definierten XML Pointer Language (XPointer) kann auch auf Elementinstanzen ohne Identifikatoren referiert werden, indem man einen Pfad durch den Strukturbaum des Dokumentes beschreibt, der zu der gemeinten Elementinstanz führt. Das Verweiskonzept kann auch derart erweitert werden, dass Verweise mit mehreren Zielen zugelassen werden. Auf diese Weise kann von einer Elementinstanz auf mehrere andere verwiesen werden. Die XML Linking Language (XLink) definiert für XML erweiterte Verweiskonzepte, mit deren Hilfe beliebige Relationen zwischen Elementinstanzen von Dokumenten kodierbar sind und Verweis- und Gliederungsstruktur relativ unabhängig voneinander verwaltet werden können.

Nur erwähnt sei hier ein weiteres Sprachelement von SGML: *Entitäten* sind Code-Sequenzen, die einzelne Zeichen oder Zeichenfolgen vertreten. Damit lassen sich insbesondere Bezeichner für Zeichen bilden, die im aktuell benutzten Code nicht vorhanden sind oder die für die Portabilität der Daten hinderlich wären. Dadurch wird die Integration beliebiger Zeichensysteme in SGML möglich.

1.2.2 Das pragmatische Grundkonzept

SGML ist intendiert als ein Kodierungssystem für ein *deskriptives* Markup. SGML-Tags sollen nicht als Befehle für bestimmte Formatierungsoperationen verstanden werden, sondern mithilfe von SGML-Elementen sollen die inhaltliche Gliederung eines Dokumentes markiert werden.

Wie das inhaltlich durch SGML-Tags gegliederte Dokument für unterschiedliche Ausgabemedien zu formatieren ist, wird unabhängig vom SGML-Markup bestimmt. Ob eine Überschrift fett, größer, zentriert erscheint, soll nicht durch SGML-Tags im Dokument kodiert werden, sondern durch eine gesonderte Spezifikation, die Elementen in Abhängigkeit von ihrer Umgebung und Attributen, Formatierungseigenschaften zuweist. Diese Zuweisung geschieht außerhalb des eigentlichen Dokuments. Die so erreichte Modularisierung ermöglicht es, ohne Veränderung des Dokuments, an unterschiedliche Ausgabemedien optimal angepasste Darstellungsformen des Dokuments herzustellen. DSSSL (Document Style Semantics and Specification Language) ist der zu SGML gehörige Standard zur Spezifikation von Darstellungsformen eines Dokuments, XSL (Extensible Style Language) der entsprechende – z.T. noch in Entwicklung befindliche – Standard für XML, der viele Prinzipien von DSSSL aufgreift.

Dass das pragmatische Grundkonzept des deskriptiven Markup bei der Entwicklung von SGML leitend war, schließt jedoch nicht aus, dass SGML auch für ein darstellungsbezogenes Markup verwendet werden kann. HTML (Hypertext Markup Language), die Sprache des WWW, ist eine SGML-basierte Markup-Sprache, die ursprünglich einige wenige Gliederungsmöglichkeiten für Dokumente vorsah. Die Art der Darstellung der Dokumente blieb weitgehend den Betrachterprogrammen überlassen. Der zunehmende Bedarf an graphischen Gestaltungsmöglichkeiten und an kontrolliertem Layout hat zur Aufnahme von Elementen und Attributen zur Steuerung des Layout geführt. HTML wurde mehr und mehr zur Layout-Programmiersprache für das WWW. Erst die Einführung von Cascading Style Sheets (CSSs) zur Bestimmung von Layout-Eigenschaften von WWW-Seiten macht es möglich, den Anteil von nicht-deskriptivem Markup in den HTML-Dokumenten selbst zu verringern, ohne auf ansprechendes Layout zu verzichten.

Bei der retrospektiven Digitalisierung allerdings wird man des öfteren davon Gebrauch machen müssen, dass sich mit SGML prinzipiell auch graphische Eigenschaften von Texten

beschreiben lassen; überall da, wo unklar ist, wie bestimmte graphische Merkmale (z.B. Wechsel des Schrifttyps, der Schriftgröße oder -farbe) zu interpretieren sind, wird man sich bei der Kodierung auf die (typo-)graphischen Eigenschaften zurückziehen müssen.

1.3 SGML vs. XML

Die Grundkonzepte von SGML sind lückenlos auch auf XML übertragbar. Neben einigen rein notationellen Unterschieden schließt XML insbesondere Merkmale von SGML aus, die die automatische Analyse von SGML-Dokumenten erschweren, dazu gehören beispielsweise Möglichkeiten, Start- oder End-Tags, die aus dem Zusammenhang zu erschließen sind, nach entsprechender Deklaration in der DTD wegzulassen. Die Markup-Minimierungsmöglichkeiten von SGML haben zur Folge, dass viele SGML-Dokumente ohne eine DTD gar nicht eindeutig analysiert werden können, dass also nicht immer eindeutig zu ermitteln ist, wie weit sich eine Elementinstanz erstreckt. Fallen die Minimierungsmöglichkeiten, wie in XML, weg, müssen also alle Elementinstanzen explizit durch ein Start-Tag geöffnet und durch ein End-Tag geschlossen werden; so kann man auch ohne eine DTD überprüfen, ob zumindest die Verschachtelung der Elementinstanzen korrekt ist und ob die verwendeten Entitäten bekannt sind. Fällt die Überprüfung positiv aus, so hat man es mit einem *wohlgeformten* Dokument zu tun. Entspricht die Struktur des Dokuments darüber hinaus noch der zugehörigen DTD, so verfügt man über ein *gültiges* Dokument. Die zwei Stufen formaler Richtigkeit erweisen sich in der praktischen Arbeit als oft sehr nützlich. Nicht immer vollzieht sich die DTD-Entwicklung und die Entdeckung neuer Erfordernisse bei der Kodierungsarbeit synchron. In solchen Fällen kann eine Arbeit mit bezüglich der aktuellen DTD ungültigen Dokumenten vonnöten sein, nichtsdestotrotz bleibt deren Wohlgeformtheit überprüfbar.

1.4 SGML und XML als Antwort auf die Anforderungen?

Die in Abschnitt 1.1 genannten Anforderungen lassen sich mit SGML bzw. XML weitgehend erfüllen.

1.4.1 Mächtigkeit

Durch die große Flexibilität bei der Definition unterschiedlichster Dokumenttypen können SGML und XML als geeignete Werkzeuge für die meisten Kodierungsaufgaben gesehen werden. Mithilfe der erweiterten Verweistechniken lassen sich beliebige Relationen zwischen Dokumententeilen kodieren. Einander überlappende Dokumentteile zwingen jedoch häufig zu intuitiv weniger eingängigen Kodierungsmechanismen. SGML und XML sind insgesamt aber mächtig genug, prinzipiell jede Datenstruktur kodieren zu können.

1.4.2 Eindeutigkeit

Mithilfe von SGML oder XML definierte Kodierungssysteme sind formal eindeutig. Eine Verwechslung von Textstücken oder Markup oder Unklarheiten, auf welche Textteile sich ein bestimmtes Markup erstreckt, sind bei geeigneter DTD-Konstruktion und gültigen

SGML-Dokumenten oder wohlgeformten XML-Dokumenten ausgeschlossen. Dass Kodierungen auch von allen Benutzerinnen und Benutzern eindeutig und hinreichend klar verstanden werden, kann natürlich auf formalem Weg nicht sichergestellt werden. Von der Text Encoding Initiative (TEI) werden jedoch in den TEI Guidelines für zahlreiche Kodierungsprobleme detaillierte Empfehlungen für die Verwendung von SGML-Markup ausgesprochen. Diese Empfehlungen können als eine Grundlage für ein detailliertes Kodierungshandbuch dienen, das auch die pragmatisch-inhaltlichen Fragen der Markup-Verwendung klärt.

1.4.3 Interpretierbarkeit

Durch SGML- bzw. XML-DTDs werden kontextfreie Grammatiken spezifiziert. SGML- oder XML-Dokumente stellen an automatische Analyseprogramme damit ähnliche Anforderungen wie der Code von Programmiersprachen. XML bedeutet gegenüber SGML eine deutliche Reduktion des Analyseaufwands.

1.4.4 Nachhaltigkeit

Jede Prognose über zukünftige technische Entwicklungen ist mit Unsicherheit behaftet. Die wachsende Bedeutung von XML im Internet-Bereich[3] und als Grundlage von Industriestandards in Wachstumsbereichen[4] berechtigt zu der Annahme, dass XML auf Jahre hin ein zentraler Standard zur Textkodierung sein wird. Verlage und Redaktionen setzen zunehmend mit langfristigen Investitionen auf den Einsatz von SGML oder XML bei der medienneutralen Texterfassung. Aber Nachhaltigkeit wird nicht zuletzt auch von den Eigenschaften gesichert, die ein hohes Maß an Portierbarkeit garantieren.

1.4.5 Portierbarkeit

SGML- und XML-Dateien sind reine Textdateien, d.h. die enthaltenen Zeichen kann man mit jedem einfachen Texteditor betrachten. Im Allgemeinen beschränkt man sich auf diejenigen Zeichen des ASCII-Zeichensatzes, die auf verschiedenen Systemen gleich interpretiert werden. Damit stellen SGML- und XML-Dateien an Editier- und Betrachterprogramme minimale Anforderungen, sofern keine SGML- oder XML-spezifischen Editier- oder Formatierfunktionen erwartet werden.

1.4.6 Softwareunterstützung

Der Softwaremarkt stellt Spezialwerkzeuge zur Arbeit mit SGML und zunehmend auch mit XML für die unterschiedlichsten Aufgaben zur Verfügung.[5] Die XML-Tauglichkeit neuer

---

[3] Über aktuelle Standardisierungsbemühungen im WWW-Bereich kann man sich auf den WWW-Seiten des World Wide Web Consortium (W3C) informieren: http://www.w3c.org .

[4] WML, die Beschreibungssprache für Internetseiten, die auf Mobiltelefonen angezeigt werden können, ist beispielsweise ein XML-basierter Standard, vgl. http://www.wapforum.org .

[5] Einen sehr gut aufbereiteten Überblick über Software zur Arbeit mit SGML und XML gibt Steve Pepper in seinem Whirlwind Guide unter http://www.infotek.no/sgmltool/guide.htm .

WWW-Browser dürfte stark zu Popularisierung von XML-Werkzeugen beitragen. Nicht alle Spezialwerkzeuge unterstützen den SGML- oder XML-Standard in allen Einzelheiten. Und nicht jede Funktionalität, die man sich für die Arbeit mit diesen Kodierungssystemen wünscht, ist in einer vorkonfektionierten Softwarelösung erhältlich. Spezielle Anwendungen werden deshalb auf die Entwicklung spezialisierter Lösungen setzen. Im Zentrum dieser Lösungen werden meist Datenbanksysteme stehen, in denen die SGML- oder XML-kodierten Daten verwaltet werden. Darauf gehen die folgenden Abschnitte ein.

2 Datenbanksysteme zur Verwaltung strukturierter Textdaten

2.1 Datenbanksysteme: Allgemeiner Aufbau

Unter dem Begriff *Datenbank (DB)* versteht man eine strukturierte Sammlung von Daten, welche für eine Reihe von unter Umständen unterschiedlichen Anwendungssystemen ein Modell eines in der Regel kleinen Teiles der von Menschen wahrgenommenen Welt repräsentiert (Heuer/Saake [1997], Vossen [1994]).

Das *Datenbank-Management-System (DBMS)* bezeichnet die notwendige Sammlung von Software, mit der unabhängig von Anwendungssystemen die lesenden und schreibenden Zugriffe (Einfügen, Ändern, Löschen) auf eine Datenbank durchgeführt und verwaltet werden. Wenn im weiteren Text von „Datenbanksoftware" oder von „Software des Datenbanksystems" gesprochen wird, sollen damit Synonyme zum Wort Datenbank-Management-System gemeint sein.

Der Begriff *Datenbanksystem (DBS)* beschreibt den Verbund einer nicht leeren Menge von Datenbanken mit einem Datenbank-Management-System.

Datenbanksystem	=	Datenbank	+	Datenbank-Management-System

Bei einem Datenbanksystem handelt es sich um eine für Anwendungssysteme weiterentwickelte Spezialisierung der Grundfunktionalität des Dateisystems eines Betriebssystems. Beim Dateisystem eines Betriebssystems (sequentielles Dateisystem) können mehrere Anwender nur unter erhöhtem Programmieraufwand für ein Anwendungssystem (z.B. durch Programmierung des wechselseitigen Ausschlusses von Schreiboperationen auf einen Datenbestand) in einem gleichen Zeitintervall lesend und schreibend auf den gleichen Datenbestand zugreifen. Sie benötigen weiterhin Werkzeuge für elementare Operationen auf Dateien: Editieren (nach Möglichkeit mit strukturierten Editierhilfen), Suchen, Sortieren. Diese Werkzeuge sind in der Regel mit dem Dateisystem alleine nicht gegeben. Die Folge ist ein in der Regel erhöhter Programmier-(bzw. Installations-)aufwand und ein eher ineffizient zeitlich ausgenutzter Datenbestand im Mehrbenutzerbetrieb.

Ein Datenbanksystem ermöglicht es mehreren Benutzern, in einem gemeinsam genutzten Laufzeitintervall mit dem gleichen physischen Datenbestand zu arbeiten. Die Benutzer greifen nicht mehr direkt, sondern über die Datenbanksoftware auf den Datenbestand zu. Die Datenbanksoftware übernimmt die Kommunikation zwischen den Datenbankbenutzern und den Schreib-/Leseoperationen auf die Datenbestände. Sie bietet folgende Grundfunktionalität an:

1. **Persistenz (= dauerhafte Verwaltung von Datenbeständen) und Sekundärspeicherverwaltung:** Die Daten sollen während der Laufzeit *strukturtreu* vom Haupt- in den Hintergrundspeicher geschrieben werden können. Die Datenbanksoftware muss Funktionen bereitstellen, um adressengesteuert Direktzugriffe auf Datensätze im Hintergrundspeicher ausführen zu können. Hinzu kommt Software, die anwendungsunabhängig das Einfügen, Ändern und Löschen von Daten auf Hintergrundspeichern ausführt.
2. **Verwaltung eines Schemakataloges (Data Dictionary):** Das Schema ist die konzeptuelle Beschreibung der Datenbanken eines Datenbanksystems. Es enthält die Datendefinitionen für sämtliche Daten, die durch das Datenbanksystem verwaltet werden. Dieser Schemakatalog, der auch ‚Data-Dictionary' genannt wird, ermöglicht während der Laufzeit des Datenbanksystems Online-Zugriffe auf die Datendefinitionen des Systems. Die in diesem und im vorhergehenden Absatz genannte Software garantiert die Forderung der *Datenunabhängigkeit* an eine Datenbank. Diese Forderung hat das Ziel, eine in der Regel langlebige Datenbank von ständig auftretenden Änderungen der auf sie zugreifenden Anwendungssysteme abzukoppeln.
3. **Interpretation einer Anfragesprache:** Ein Datenbanksystem bietet dem Benutzer eine Anfragesprache an, mit der er, ohne die interne Speicherung der Daten in der Datenbank zu kennen, auf die Datenmengen des Datenbanksystems zugreifen kann. Der Zugriff geschieht in der Regel interaktiv.

Weitere sechs notwendige Merkmale eines DBMS sind: Sicherung der Integrität, Verwaltung von Benutzersichten, Datenschutz, Transaktionsverwaltung, Synchronisation, Recovery/Datensicherung.

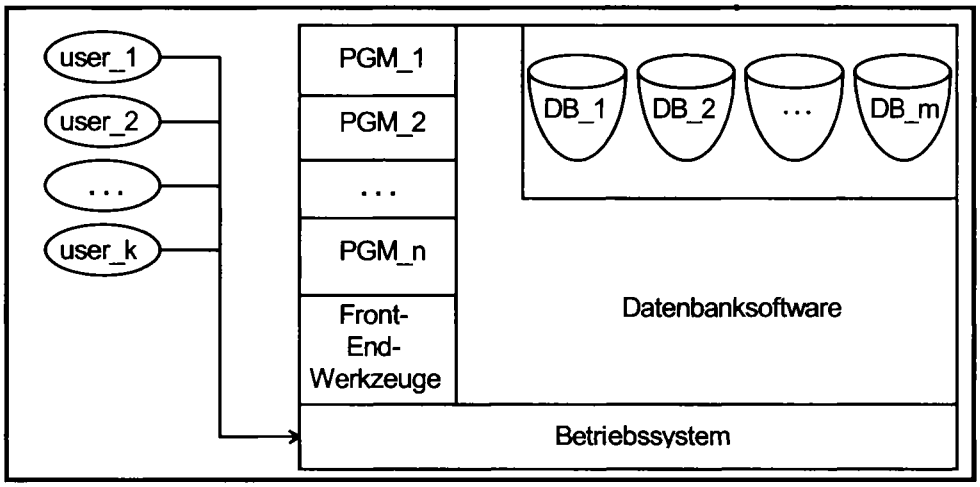


Abb. 1: Allgemeiner Aufbau eines Datenbanksystems

Legende zu Abbildung 1:

user_i ($1 \leq i \leq k$): Anwender des Datenbanksystems, die ihre Schreib- bzw. Leseanforderungen an das Datenbanksystem über einen vernetzten Rechner an den Rechner, auf dem das Datenbanksystem installiert ist, weitergeben.

PGM_i ($1 \leq i \leq n$): DB-Anwendungsprogramm, das durch Schreib- bzw. Leseanforderungen eines Anwenders gestartet wird und mittels der Datenbanksoftware auf Datenbestände des Datenbanksystems zugreift. Jedes DB-Anwendungsprogramm ist in der Regel durch folgenden Schichtenaufbau gekennzeichnet:

Benutzerschnittstelle	Melden und Empfangen von User-Daten
Algorithmische Schicht	Verfahren zur Lösung des Anwendungsproblems
DB-Zugriff	Aufruf von Funktionen der DB-Software

Abb. 2: Schichten eines DB-Anwendungsprogramms

Front-End-Werkzeuge:

- *a) Benutzerschnittstelle* zur interaktiven Verarbeitung von Kommandos der Anfragesprache: Der Benutzer kann hier unmittelbar Kommandos eingeben, die an den Kommandointerpreter weitergeleitet, analysiert und ausgeführt werden.
- *b) Dialogmaskengenerator*: Der Dialogmaskengenerator ist ein Softwaresystem zum Erzeugen von Dialogmaskenprogrammen. Dialogmasken (engl. *forms*) gestalten eine formatierte Eingabe für den lesenden und schreibenden Zugriff auf Datenbanken. Wegen der formatierten Eingabe und der damit verbundenen Möglichkeit der Integritätskontrolle sind Dialogmaskenprogramme für die Bearbeitung von Daten in Lexikonartikeln gut geeignet. Beim lesenden Zugriff ist insbesondere die Unterstützung von *Suchfunktionen* (trunkierte Suche, maskierte Suche) durch Dialogmaskenprogramme hervorzuheben, die inzwischen bei gängigen Datenbanksystemen auch in Form von WWW-Programmen implementiert sind. Die Ergebnisse einer Suchanfrage werden als sortierte Listen ausgegeben. Hierfür stellt das DBMS komfortable, für den Benutzer unaufwendige, in Hinsicht auf das Datenmodell einer Datenbank flexible *Sortierroutinen* zur Verfügung.
- *c) Berichtsgenerator*: Berichtsprogramme sind Programme, die in der Betriebsart Stapelverarbeitung des gegebenen Betriebssystems ablaufen. (Ein Desiderat der computergestützten Lexikographie sind z.B. SGML-Berichtsgeneratoren).

2.2 Datenbankentwurf und Datenbankmodelle

Das Design einer Datenbank entspricht dem Einteilen der Daten eines geplanten Anwendungssystems in verschiedene Entitätenmengen (spätere Tabellen der Datenbank) und dem Bestimmen der zwischen den Entitätenmengen (Entities) bestehenden Beziehungstypen (Relationships). Zum Design von Datenbanken hat sich das Entity-Relationship-Modell als Entwicklungsmethode bewährt. Entity-Relationship-Modelle unterstützen die Abbildung formaler Beschreibungen von Lexikonartikeln, die z.B. in SGML vorliegen, auf Tabellenstrukturen eines einzurichtenden DBS zur Verwaltung lexikographischer Daten. Graphisch werden Entity-Relationship-Modelle durch E/R-Diagramme dargestellt.

Syntaktisch bestehen E/R-Diagramme nur aus zwei Symbolen. Jeweils eines für Entities (Entitytypen) und eines für Relationships (bzw. Beziehungstypen oder Relationstypen). Die Kanten eines E/R-Diagramms können mit Quantitäten (sog. erweitertes E/R-Diagramm) und auch mit einem Durchlaufsinn eingefärbt sein. Es hat sich folgende Darstellungsweise durchgesetzt (Abbildung 3):

Bezeichnung	Symbol	Bemerkung
Entitytypen	Entity-Name	
Relationen	r_1 — Relationstyp — r_2	Die Quantitäten r_1, r_2 bezeichnen Anzahlen sich entsprechender Elemente in den Tupeln des Relationstyps R, der zwischen zwei Entitätenmengen A und B besteht. Es gibt folgende Standardquantitätsangaben: $r_1, r_2 \in \{1, n, m, c, c{\cdot}n, c{\cdot}m\}$ $c \in \{0,1\}; n, m \in \mathbf{N}$.

Abb. 3: Syntax von E/R-Diagrammen

Nachfolgend ist als Beispiel für die Informationselemente der Artikel des „Hegel-Lexikons" von Hermann Glockner (Glockner [1935]) ein Entity-Relationship-Modell angegeben (Abb. 4).

Nach Aufstellung des Entity-Relationship-Modells kann in Hinsicht auf die Merkmale des vorliegenden lexikographischen Datenmodells die Auswahl des Typs des zu benutzenden DBMS (z.B.: relational oder objektorientiert) getroffen werden. Fragen für die Entscheidungsfindung können z.B. sein:

- Sind die Daten vorwiegend als Tupel strukturiert oder sind sie hauptsächlich in Hierarchien bzw. Netzwerken angeordnet?
- Will man die Daten durch eine komplexe oder durch eine einfache Anfragesprache verwalten?
- Hat man zwischen den verschiedenen Entitätenmengen viele oder wenige Eigenschaften, die sich „vererben" lassen?
- Möchte man ein weitverbreitetes DBMS oder kann man auch mit einem weniger verbreiteten DBMS arbeiten?

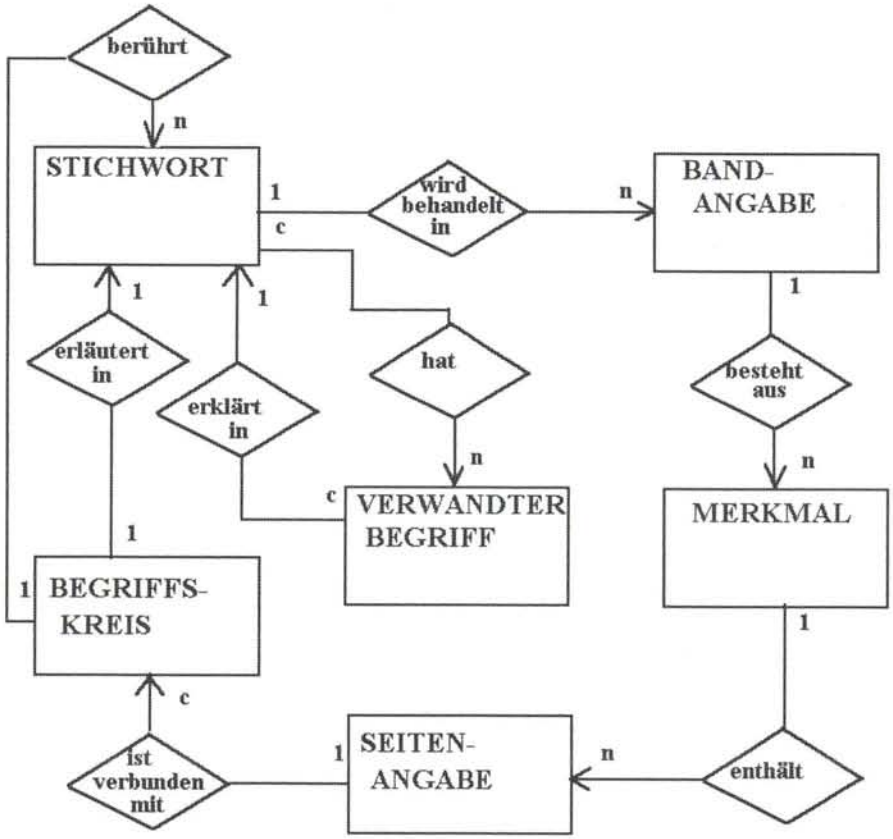


Abb. 4: E / R-Diagramm zu H. Glockner: „Hegel-Lexikon"

2.2.1 Relationale Datenbanksysteme (RDBS)

Das relationale Datenbankmodell wurde 1970 von E.F. Codd entwickelt. Die unmittelbar abbildbaren Datenstrukturen sind Tupel elementarer Datentypen, die in Tabellen zusammengefasst werden. Das relationale Datenbankmodell basiert auf den mathematischen Operationen der relationalen Algebra (Selektion, Projektion, Verbund, Differenz und Vereinigung von Relationen) (Sauer [1994], Schwinn [1992]). Die Relationen (Beziehungen) werden durch zweidimensionale Tabellen dargestellt. Dabei wird die Anzahl der Spalten fest vorgegeben. Die Zeilen der Tabelle enthalten dabei die Datenobjekte (Datensätze), die durch ihren Schlüssel unterschieden werden können. Es kommt also keine Zeile einer Tabelle zweimal vor. Die Spalten (Datenfelder) enthalten die Attributwerte, die in einer Spalte immer vom gleichen elementaren Datentyp sind. Elementare Datentypen sind z.B. INTEGER (ganze Zahl), CHAR(n) (Zeichenkette der Länge n), FLOAT (Gleitkommazahl), usw. Die Reihenfolge der Zeilen und Spalten ist im relationalen Datenmodell gleichgültig.

	SP_1	SP_2	...	SP_k	...	SP_N
Z_1				a_{1k}		
...				....		
Z_M				a_{Mk}		

Abb. 5: Allgemeiner Tabellenaufbau

Legende zu Abbildung 5:

- SP_K: Name der k-ten Spalte (=: Name des Attributes A_k)
- Datensätze sind in den Zeilen $Z_1, Z_2, \ldots, Z_M$ enthalten.
- Tupelstruktur: $Z_i=(a_{i1}, a_{i2}, \ldots, a_{iN}) \in W_1 \times W_2 \times W_3 \times \ldots \times W_N$ (W_j ist Wertebereich zur Spalte Sp_i). Alle Attributwerte des Attributes A_k sind vom gleichen elementaren Datentyp $dtyp(A_k)$.

Mit relationalen Datenbanken ist in kanonischer Weise die Datenbanksprache SQL (SQL = *S*tructured *Q*uery *L*anguage) verbunden (Knebel/Postels [1991], Miesgeld [1991], Petkovic [1995]). SQL basiert auf der Relationenalgebra und dem Tupelkalkül, das dem Tabellenaufbau relationaler Datenbanken zugrunde liegt. SQL ist eine Anfrage-, Datendefinitions- und Datenmanipulationssprache. Mit ihr können Benutzersichten, Dateiorganisationsformen und Zugriffspfade definiert werden. SQL ist eine genormte Datenbanksprache (ANSI/ISO [SQL-92], eine neue Norm [SQL3] steht zur Verabschiedung an).

In einem einfachen lexikographischen Datenbanksystem werden Daten aus Hegel-Registern und Hegel-Lexika verwaltet. Hieraus ist das Beispiel in Abbildung 6 entnommen worden.

```
                                Terminal - averoest
File  Edit  Session  Options  Help
PERFORM:   Query  Next  Previous  View  Add  Update  Remove  Table  Screen  ...
Shows the next row in the Current List.          ** 1: glocktab table**
 +-------------------------------------------------------------------------+
 |                      Glockner : Hegel - Lexikon                         |
 |-------------------------------------------------------------------------|
 |                                                                         |
 |  Schlagwort   [Abstraktion                          ]                   |
 |  Grundform    [Abstraktion                          ]                   |
 |                                                                         |
 |  Schlagwortzeile                                                        |
 |  [(abstrakt,abstrakt-konkret,Allgemeines)                           ]   |
 |  [                                                                  ]   |
 |                                                                         |
 |  Bemerkung 1 : [B          ]                                            |
 |  Bemerkung 2 : [           ]                                            |
 |                                                                         |
 |  Begriffskreise :                                                       |
 |  [Begriff                                                           ]   |
 |  [Dasein                                                            ]   |
 |  [symbolische Architektur                                           ]   |
 |  [indische Religion                                                 ]   |
 +-------------------------------------------------------------------------+
```

Abb. 6: SQL-FORM zur Verarbeitung der GLOCKNER-Tabelle

2.2.2 Objektorientierte Datenbanksysteme (OODBS)

Das Konzept *objektorientierter Datenbanken* vereinigt **zwei** Mengen wesentlicher Eigenschaften (Atkinson [1992]):
(1.) Es beinhaltet ein *objektorientiertes Datenmodell.*
(2.) Es enthält die Speicherungstechniken eines *gewöhnlichen Datenbanksystems* (Persistenz, Sekundärspeicherverwaltung, Verwaltung von Transaktionen, Recovery-Techniken, ...; vgl. Heuer [1997], Hughes [1992], Saake/Türker/Schmitt [1997]).

Auszug aus der Liste der Forderungen zu (1.):

(1.1) Möglichkeit der *direkten Modellierung komplexer Objekte* (abstrakte Datentypen).
(1.2) *Objektidentität (OID).*
(1.3) Jedes Objekt kapselt Struktur und Verhalten *(Kapselung).* Die Struktur eines Objekts wird durch Instanz-Variablen beschrieben. Das Verhalten eines Objekts wird durch Methoden beschrieben.
(1.4) Objekte mit gemeinsamer Struktur und gemeinsamen Verhalten werden in *Klassen* gruppiert. Jedes Objekt ist Instanz einer Klasse.
(1.5) Klassen können als Spezialisierung anderer Klassen definiert werden *(Vererbung).*

OODBS eignen sich zur Speicherung von komplizierten Datenstrukturen, deren Abbildung auf Tabellen eines RDBMS mit Schwierigkeiten verbunden ist. (Eine Tabelle hat eine möglicherweise große, aber immer konstante, den Anwendungsdaten beim Betrieb des DBS vorgegebene Anzahl von Spalten, die während der Laufzeit nicht geändert werden kann!) Zu solchen Datenstrukturen zählen z.B.: Baumstrukturen (Hierarchien) mit vielen Hierarchiestufen, netzwerkartige Strukturen mit unterschiedlichen Kantentypen.

Eine Klasse definiert den Aufbau und beschreibt über die Methoden das Verhalten eines Objektes. Alle Instanzen einer Klasse besitzen somit das gleiche Verhalten und die gleiche Wertestruktur. Durch Definition von Klassen vergrößert sich die Menge der zulässigen Datentypen im Data-Dictionary. Damit bietet sich die Möglichkeit, abstrakte Datentypen zu definieren (List-Typen, Mengentypen). Als Anfragesprachen für OODBS werden höhere Programmiersprachen verwendet, die Klassen als Datentypen zulassen, wie C++, JAVA und SMALLTALK. Wie in der Programmiersprache C++ sind Objekte in einem OODBS verkapselt, d.h. Methoden und Daten eines Objektes bilden eine Einheit. Der Benutzer kann nicht direkt auf die Werte eines Objektes zugreifen, sondern nur über die Methoden, die der Klasse des Objektes bekannt sind. In einem OODBS können dynamische Listen verhältnismäßig einfach als Klassen modelliert werden. OODBS werden besonders zur Realisierung komplexer, datenintensiver Anwendungen im Konstruktions- und Entwurfsbereich (CAD) eingesetzt.

Ein OODBS muss weiterhin das Konzept der Vererbung (Spezialisierung) unterstützen. Beispiel: Die Entität Student ist eine Spezialisierung der Entität Person. Man unterscheidet zwischen Superklassen (hier Person) und sogenannten Subklassen (hier Student). Die Spezialisierungsbeziehung zwischen Klassen führt zu Klassenhierarchien.

Ein Beispiel für Vererbungsbeziehungen zwischen Klassen eines lexikographischen Datenmodells kann in einem Modell einer datenbankorientierten Beschreibung ausgewählter Informationselemente des Artikelaufbaus des „DUDEN – Das große Wörterbuch der deutschen Sprache" (DUDEN [1993]) gegeben werden.[6] Folgende Tabelle (Tab. 1), die in Anlehnung an Lenders (1990) eine „Vorform" einer Artikelbeschreibung darstellt, gibt eine

[6] An dieser Stelle möchten wir Frau Dr. A. Storrer (IDS Mannheim) danken, die uns den Hinweis auf dieses schöne Beispiel einer Vererbungsstruktur gab.

hierarchisch strukturierte Liste von Entitäten (mit Attributen) an, die auf unterschiedlichen Ebenen mit grammatischen Angaben versehen sind.

Tab. 1:

ENTITÄTENMENGEN	LISTENELEMENTE	BEISPIEL	ANM.
LEMMA (Stichwort)		Abstich	(1)
GRAMMATISCHE ANGABE zum LEMMA (=: GRAM_H)		der; -[e]s, -e	
LISTE von LESARTEN	pro Lesart:		(2)
	TITEL der Lesart	das Abstechen	
	ERLÄUTERUNG der Lesart	der A. von Torf, Rasen	
	GRAMMATISCHE ANGABE zur Lesart (optional) (=: GRAM_L)	<o.Pl.>	
LISTE von KOMPOSITA	∅	∅	(3)
Etc.	...	...	...

Anmerkungen zu Tabelle 1:

(1) DUDEN (1993), S. 98.

(2) Das Beispiel weist drei weitere Lesarten auf, wobei die Lesart Nr. 3 zwei Varianten hat:

NR	TITEL der Lesart	ERLÄUTERUNG der Lesart	GRAMM. ANGABE	SACHGEBIET
2)	Art des Kantenverlaufs beim Sakko ...	stark fliehender A.	∅	Schneiderei
3a)	das Abstechen	der A. des [Roh]eisens	<o.Pl.>	Hüttenwesen
3b)	Teil eines Hochofens, ...	die Gießpfanne unter den A. rücken	∅	
4)	das Abstechen, Kontrast	dort erschien sie licht, im A. ihrer nächtlichen Umgebung (Grillparzer, Medea I)	∅	

(3) Das Beispiel enthält keine Komposita.

Im Folgenden wird für die in dieser Tabelle gegebenen Entitätenmengen ein Klassenbeziehungsgraph (Abb. 8) angegeben, der als Instrument des Entwurfes eines OODBS dienen kann. Das Aufstellen eines Klassenbeziehungsgraphen setzt in der Regel eine Entity-Relationship-Analyse voraus.

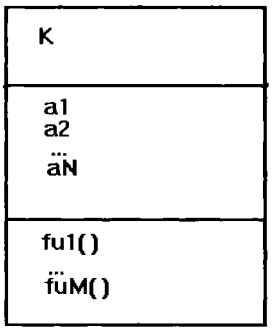


Abb. 7: Klassenbeziehungsgraph

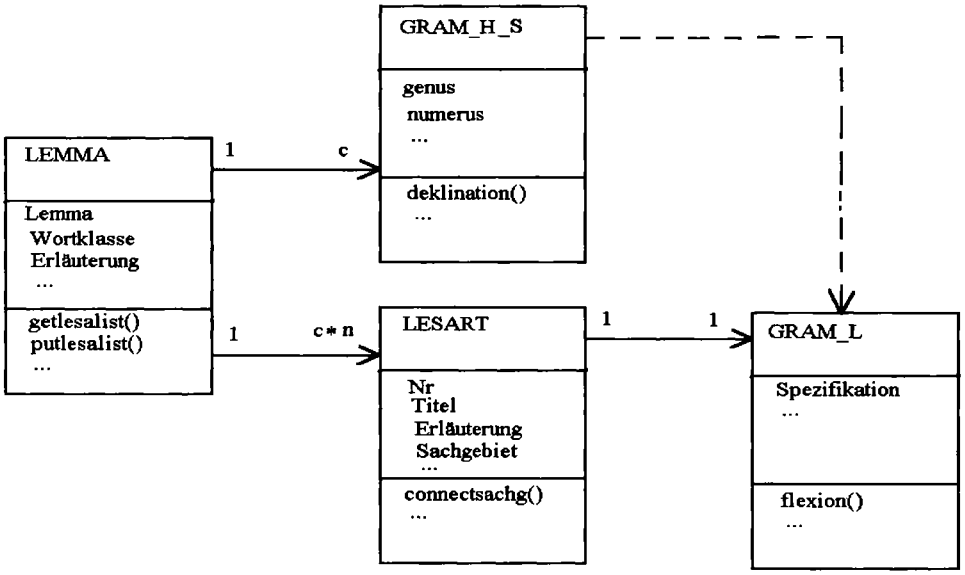
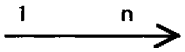
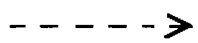


Abb. 8: Klassenbeziehungsgraph

Der (1:n)-Beziehungstyp wird als sog. gerichtete Assoziation mit dem Symbol



dargestellt (Fowler [1995], vgl. auch: Booch [1995], Rumbaugh [1993]). Die Vererbung zwischen zwei Klassen wird hier mit dem Symbol



bezeichnet.

Das Symbol einer Klasse im Klassenbeziehungsgraph (vgl. Abb.6) setzt sich zusammen aus einem Klassennamen K, einer Liste von Attributen a1,..., aN und einer Liste von Methoden fu1(),..., fuM(), die die Daten der Klasse K verarbeiten.

3 Resümee

Dieser Beitrag geht auf Entscheidungskriterien ein, die für die Wahl eines Kodierungssystems für komplexe Textstrukturen relevant sind. In den meisten Fällen dürfte die Wahl eines SGML- oder XML-basierten Kodierungssystems adäquat sein. Da SGML und XML zur Kodierung beliebiger Datenstrukturen verwendet werden können, wird die Verwendung von SGML oder XML immer prinzipiell möglich sein. Die Verwaltung SGML- oder XML-kodierter Daten kann mithilfe von Datenbanksystemen geschehen. Dazu sind die in SGML oder XML kodierten Textobjekte und Beziehungen auf die Ausdrucksmittel des gewählten Datenbanktyps abzubilden. Dies ist in jedem Fall möglich, da in SGML- und XML-kodierten Dokumenten vorhandene Strukturen auf wenige Relationen zwischen Elementinstanzen oder zwischen Text und Elementinstanzen oder zwischen Elementinstanzen, Attributen und Werten abbildbar sind. Welche Datenbank-Modellierung sich unter Gesichtspunkten des Datenzugriffs und der Datenpflege als günstig erweist, hängt stark von der Art der Dokumentstrukturierung und dem Nutzungszweck ab.

4 Literatur

Atkinson, M., et. al. (1992): The Object Oriented Database System Manifesto – In: Bancilhon, F., et al.: Building an Object-Oriented Database System – The Story of O2. – San Francisco, Ca.: Morgan-Kaufmann.

Booch, G. (1995): Objektorientierte Analyse und Design. – Bonn, Albany: Addison-Wesley.

DUDEN: DAS GROSSE WÖRTERBUCH DER DEUTSCHEN SPRACHE in acht Bänden. Hg. G. Drosdowski. Mannheim: Dudenverlag 1993.

Fowler, M. (1997): UML Distilled – Applying the Standard Object Modeling Language. – Reading (Mass.) et al.: Addison-Wesley.

Glockner, H. (1935): Hegel-Lexikon. – In Hegel, G.W.F.: Sämtliche Werke (Jubiläumsausgabe), Bd. 23–26, Stuttgart: Frommann.

Goldfarb, Charles F. (1990): The SGML Handbook. – Oxford: Clarendon Press.

– und Prescod, Paul (1998): The XML Handbook. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

Graham, Ian S., Quin, Liam (1999): XML Specification Guide. New York, NY: John Wiley & Sons.

Hald, A., Nevermann, W. (1995): Datenbank-Engineering für Wirtschaftsinformatiker. – Braunschweig, Wiesbaden: Vieweg.

Heuer, A. (1997): Objektorientierte Datenbanken, Bonn: Addison-Wesley.

– und Saake G. (1997): Datenbanken – Konzepte und Sprachen. – Bonn, Albany: Int. Thomson Publishing Comp.

Hughes, J.G. (1992): Objektorientierte Datenbanken. – München, Wien: C. Hanser (in coedition with Prentice Hall).

Knebel, B., Postels, G. (1991): Einführung in Informix-SQL. – Heidelberg: Hüthig.

Lenders, W. (1990): Semantische Relationen in Wörterbuch-Einträgen – Eine Computeranalyse des DUDEN-Universalwörterbuches. – In: Schaeder, B., Rieger, B. (Hgg.): Lexikon und Lexikographie. Hildesheim: Olms.

– (Hg.) (1993): Computereinsatz in der angewandten Linguistik – Konstruktion und Weiterverarbeitung sprachlicher Korpora. - Frankfurt a.M.: Lang.

Lobin, Henning (Hg.) (1999): Text im digitalen Medium. – Opladen, Wiesbaden: Westdeutscher Verlag.

Misgeld, W. (1991): SQL – Einstieg und Anwendung. – München, Wien: Hanser Verlag.

Möhr, Wiebke, Schmidt, Ingrid (Hgg.) (1999): SGML und XML – Anwendungen und Perspektiven. – Berlin etc.: Springer.

Musciano, Chuck, Kennedy, Bill (1999): HTML – Das umfassende Referenzwerk. 2. Aufl. – Köln: O'Reilly.
Petkovic, D. (1995): Informix 6.0/7.1. – Bonn: Addison-Wesley.
Rumbaugh, J., Blaha, M., et.al. (1993): Objektorientiertes Modellieren und Entwerfen, München, London: Hanser/Prentice-Hall.
Saake G., Türker C., Schmitt I. (1997): Objektdatenbanken. – Bonn, Albany: Int. Thomson Publishing Comp.
Sauer H. (1994): Relationale Datenbanken – Theorie und Praxis. – Bonn, et al.: Addison-Wesley.
Schröder, Bernhard (1998): Pro-SGML: Ein Prolog-basiertes System zum Textretrieval. – In: Gerhard Heyer, Christian Wolff (Hgg.): Linguistik und neue Medien, 205–216. Wiesbaden, DUV.
– und Ostermann-Heimig, Jens (1998): Kants Werke als Hypertext. – In: Angelika Storrer, Bettina Harriehausen (Hgg.): Hypermedia für Lexikon und Grammatik, 233–246. Tübingen, Narr.
Schwinn, H. (1992): Relationale Datenbanksysteme. – München, Wien: C. Hanser Verlag.
Sperberg-McQueen, C. M., Burnard, Lou (1994): TEI Guidelines for Electronic Text Encoding and Interchange (P3). 1.4.2000, http://etext.lib.virginia.edu/TEI.html.
Vossen, G. (1994): Datenbanken, Datenmodelle, Zugriffssprachen. – Bonn: Addison-Wesley.
Wilde, Erik (1999): World Wide Web – Technische Grundlagen. – Berlin, etc.: Springer.

Gregor Büchel, Köln
Bernhard Schröder, Bonn