

Komplexität und Einfachheit in der Evolution von Dokumentgrammatiken

Henning Lobin
Justus-Liebig Universität Gießen

1 Einleitung

Der Begriff der Sprache spielt nicht nur in der Linguistik eine zentrale Rolle, auch die Informatik hat diesen Begriff in ihr terminologisches Inventar aufgenommen, um damit ganz allgemein Mengen von Zeichenketten zu benennen. Derartige formale Sprachen lassen sich nach den Eigenschaften der Grammatiken, mit denen sie erzeugt werden können, klassifizieren und hinsichtlich der Möglichkeiten ihrer automatischen Verarbeitung bewerten.

Im Gebiet der Computerlinguistik fließen beide Sichtweisen auf Sprache zusammen: Die Computerlinguistik befasst sich mit natürlichen Sprachen, versucht diese aber als formale Sprachen zu modellieren. Formale Sprachen dienen dabei also als ein Instrument, die Eigenschaften der natürlichen Sprache besser zu verstehen – vor allem in Hinsicht auf ihre prozeduralen Aspekte. Neuerdings werden formale Sprachen allerdings auch eingesetzt, um Informationen nach dem Vorbild der natürlichen Sprache zu strukturieren. Stand bislang allein das Bild der tabellarisch organisierten, in Datenbanken abgelegten Informationen im Zentrum des Interesses, werden nun auch Informationen in einer flexibleren, am Vorbild der sprachlich-textuellen Gestalt orientierten Form in Datenverarbeitungsprozesse integriert.

Mit der *Extensible Markup Language* (XML) steht seit Ende der neunziger Jahre ein Standard für die abstrakte textuelle Informationsstrukturierung zur Verfügung. XML beinhaltet zweierlei: eine Notation zur Spezifikation von formalen Grammatiken und eine Notation zur Auszeichnung von Daten. Beides greift ineinander, da für die Daten eine Grammatik festgelegt werden kann, der sie strukturell zu entsprechen haben. Damit kann man die Daten, die durch eine solche Grammatik definiert werden, auch als eine formale Sprache verstehen.

Damit hat sich die Computerlinguistik ein weiteres Anwendungsfeld erschlossen: Die Übertragung linguistischer Strukturierungsmethoden in außersprachliche Bereiche. Interessant ist daran aus linguistischer Sicht insbesondere, ob sich die Entwicklung der grammatischen Modellierung der Information in ähnlicher Weise vollzieht, wie es sich in der modernen Linguistik gezeigt hat. Zugleich richtet die Übertragung von Erkenntnissen der Grammatikforschung auf das Gebiet der Informationsstrukturierung den Blick auf inhärente Strukturen, die in tabellarischen Datenmodellen nicht zutage treten. Die Frage, wo die Grenzen der grammatischen Komplexität für Informationsstrukturen liegen, ist somit essenziell: Wie elaboriert müssen die grammatischen Strukturen auf der einen Seite sein, um relevante Zusammenhänge in den Daten darstellen zu können, und wie einfach dürfen sie auf der anderen Seite

sein, soll noch ein struktureller Gewinn gegenüber eine tabellarischen Informationsstrukturierung erzielt werden?

2 Grundelemente von XML

2.1 Textauszeichnung

Themen wie die im vorangegangenen Abschnitt skizzierten werden in letzter Zeit häufig dem Gebiet der Texttechnologie zugeordnet (vgl. z.B. Lobin/Lemnitzer 2003, Rehm 2003). Ein zentrales Mittel der Texttechnologie bildet die Textauszeichnung. Unter Textauszeichnung wird die Einfügung von Markierungen in einen Text verstanden, durch die einzelne Textteile in spezieller Weise verarbeitet werden können (vgl. Sperberg-McQueen 2001). Derartige Markierungen, auch *Annotation* oder *Markup* genannt, setzen sich aus einem *Anfangs*- und einem *Endtag* zusammen. Der Sinn der Textauszeichnung kann darin bestehen,

- einen Textabschnitt eindeutig identifizierbar zu machen (z.B. eine Definition),
- eine Zuordnung zu einer Gruppe gleich zu behandelnder Textabschnitte vorzunehmen (z.B. Überschriften) oder
- weitere Informationen, sog. Metadaten, zur Verfügung zu stellen (z.B. Bearbeiter und Datum der letzten Änderung eines Textabschnitts).

Dementsprechend können sehr unterschiedliche Verarbeitungsverfahren von der Annotation Gebrauch machen:

- In einem Hypertext-System kann durch Links auf eindeutige Textstellen verwiesen werden – die Textauszeichnung wird also für Browsing-Zwecke verwendet;
- eine Gruppe von gleich ausgezeichnet Textabschnitten kann grafisch gleich behandelt werden – die Textauszeichnung bildet dabei die Grundlage für die visuelle Textgestaltung;
- Metadaten können genutzt werden, um in einem Textkorpus Textabschnitte mit bestimmten Eigenschaften aufzufinden – die Textauszeichnung steht hier im Dienste des *Information Retrieval*.

Insbesondere für Layoutierungsprogramme werden schon seit den sechziger Jahren Textauszeichnungsverfahren angewendet. Aber erst mit der *Standard Generalized Markup Language* (SGML), einer standardisierten Metasprache zur Spezifikation von Textauszeichnungsvokabularen, stand ab 1986 ein Instrumentarium zur Verfügung, durch das die Textauszeichnung von bestimmten Anwendungssystemen unabhängig wurde und zugleich für unterschiedliche Anforderungen spezifisch zugeschnitten werden konnte. Die bekannteste Anwendung von SGML wurde in den neunziger Jahren HTML, die "Sprache des World Wide Web"; die vereinfachte Version von SGML, XML, hat heute SGML weitgehend ersetzt.

Eine der grundlegenden Neuerungen im Bereich des Dokument-Markup, die durch SGML eingeführt wurden und sich auch auf XML auswirken, besteht darin, die durch Tags annotier-

ten Textteile in einen hierarchischen Zusammenhang zu bringen, anstatt sie lediglich linear anzuordnen (vgl. Lobin 2000). Die Textteile befinden sich dadurch nicht nur in der natürlich immer noch vorhandenen sequenziellen Reihenfolge, sondern stehen zusätzlich auch noch in einer Teil-Ganzes-Beziehung zueinander. Ein Textelement „Kapitel“ kann danach z.B. aus den Textelementen „Überschrift“ und „Absatz“ bestehen. Das Kapitelelement seinerseits konstituiert mit anderen Elementen z.B. ein Textelement „Buch“:

```
(1) <buch>
      <autor>
        <vorname>Franz</vorname>
        <name>Kafka</name>
      </autor>
      <titel>Der Prozeß</titel>
      <kapitel>
        <überschrift>Verhaftung - Gespräch mit Frau Grubach - Dann
        Fräulein Bürstner</überschrift>
        <absatz>Jemand mußte Josef K. verleumdet haben, denn ohne daß
        er etwas Böses getan hätte, wurde er eines Morgens verhaftet.
        ...</absatz>
        <absatz>...</absatz>
        ...
      </kapitel>
      <kapitel>
        <überschrift>Erste Untersuchung</überschrift>
        ...
      </kapitel>
      ...
    </buch>
```

Informationselemente, die in dieser Weise verschachtelt sind, lassen sich grafisch und formal als Bäume darstellen. Die Interpretation von Textdokumenten als Bäume hat einige große Vorteile gegenüber einer linearen Textauszeichnung: Wenn in einem Baum einem Element eine Eigenschaft zugeordnet ist, so kann etwa festgelegt werden, dass diese Eigenschaft auch allen untergeordneten Elementen zukommt, sofern bei ihnen nicht explizit etwas anderes vermerkt ist. Dieses Prinzip der *Vererbung* von Eigenschaften ist also ein Mittel, Eigenschaften effizient Teilen eines Dokuments zuzuordnen. Weiterhin erlaubt die Baumstruktur eine recht präzise Navigation im Dokument. Ein Ausdruck wie „der zweite Absatz im nachfolgenden Kapitel“ kann leicht als ein Pfad durch den Dokumentbaum dargestellt werden.

Aus Sicht der maschinellen Verarbeitung von annotierten Dokumenten können wegen der Universalität von baumartig strukturierter Information in der Informationstechnologie viele Verfahren aus anderen Bereichen übernommen werden. Bei der Überprüfung beispielsweise, ob ein Dokument auch eine korrekt aufgebaute Baumstruktur aufweist, werden Verfahren eingesetzt, die für die Analyse von Programmen entwickelt worden sind.

2.2 Grammatische Strukturierung

Aus Sicht der Linguistik ist jedoch der wichtigste Vorteil der Verwendung von Baumstrukturen, dass man diese durch *Grammatiken* beschreiben kann, und Grammatiktheorie und

Grammatikformalismen in der Linguistik als gut untersuchte Gebiete gelten können. Sowohl XML als auch sein Vorläuferstandard SGML definieren nämlich nicht nur, wie die Baumstruktur eines Dokuments formal zu notieren ist, beide Standards beinhalten auch eine Sprache zur Definition von Baumgrammatiken.

Bei der Informationsstrukturierung können wir zunächst zwei verschiedene Ebenen erkennen, auf denen Informationseinheiten erscheinen können:

1. die Ebene der konkreten Daten,
2. die Ebene der abstrakten Einheiten, die Daten Funktionen zuordnen oder gruppieren.

Die abstrakten Einheiten werden in XML, wie wir bereits gesehen haben, als Elemente bezeichnet. Zwei verschiedene Typen von Elementen können unterschieden werden:

1. Daten-Elemente: Elemente dieses Typs enthalten unmittelbar die konkreten Daten;
2. Container-Elemente: Elemente dieses Typs enthalten selbst wiederum Elemente, wobei die enthaltenen Elemente Daten- oder Container-Elemente sein können.

Daten- und Container-Elemente können auch als Mischform auftreten, jedoch wird gewöhnlich empfohlen, bei der Definition von Elementen eine eindeutige Zuordnung zu einer der beiden Gruppen vorzusehen.

Für Daten-Elemente muss selbstverständlich klar sein, um was für eine Art von Daten es sich handelt. Soweit nichts anderes vermerkt ist, wird davon ausgegangen, dass Daten-Elemente Zeichenfolgen enthalten, die aus den dafür zulässigen Zeichen gebildet sind. In XML sieht die Deklaration eines Daten-Elements folgendermaßen aus:

```
(2) <!ELEMENT autor (#PCDATA)>
```

Diese Definition besagt, dass ein Element vom Typ `autor` Daten vom Typ `PCDATA` enthalten kann. Das Zeichen `#` signalisiert dabei, dass es sich bei `PCDATA` um ein vordefiniertes Schlüsselwort handelt. Elemente erhalten also einen Namen, durch den es möglich wird, die Information, die in ihnen enthalten ist, zu klassifizieren und zu beschreiben.

Container-Elemente enthalten im Gegensatz zu Daten-Elementen nicht unmittelbar Daten, sondern organisieren Elemente auf einer hierarchisch höheren Ebene:

```
(3) <!ELEMENT buch (autor, titel, kapitel)>
```

Der Elementtyp `buch` ist hier als die Abfolge von drei untergeordneten Elementen definiert. Die Festlegung, in welcher Weise ein Container-Element im Einzelnen andere Elemente organisiert, geschieht durch das *Inhaltsmodell*, der rund geklammerte Teil innerhalb der Element-Deklaration. Im Inhaltsmodell werden zum einen Angaben darüber gemacht, welchen Status die einzelnen Elementtypen haben, zum anderen werden die Elemente zueinander in Beziehung gesetzt. In (3) weisen alle Elemente den Status ‚obligatorisch‘ auf, da ihrem Na-

men kein weiteres Zeichen folgt. Insgesamt gibt es vier verschiedene Statusangaben für Elemente:

- a → a muss genau einmal auftreten (obligatorisch)
- a? → a kann einmal auftreten, kann aber auch ausgelassen werden (fakultativ)
- a+ → a muss mindestens einmal, kann aber beliebig oft auftreten
- a* → a kann einmal oder beliebig oft auftreten, kann aber auch ausgelassen werden

In (3) können wir beispielsweise die folgenden Ergänzungen vornehmen:

```
(4) <!ELEMENT buch (autor+, titel, verlag?, kapitel+)>
```

Durch diese Deklaration ist es nun möglich, mehrere Autoren für ein Buch anzugeben und auch eine Verlagsangabe vorzunehmen.

Für die Festlegung der Beziehungen der Elemente zueinander stehen zwei Konnektoren zur Verfügung:

- a, b → b folgt auf a
- a | b → entweder a oder b

Die hierarchische Anordnung der Elemente ist das Grundprinzip von XML. Wollen wir z.B. das Element `autor` weiter untergliedern, so können wir dazu folgende Deklarationen verwenden:

```
(5) <!ELEMENT autor (vorname+, name)>
```

Die Vornamen-Information zum Autor eines Buches erscheint dann eingeschachtelt in das `autor`-Element und dieses schließlich im `buch`-Element.

2.3 Vorteile grammatisch strukturierter Information

Es ist bisher deutlich geworden, dass jede Element-Deklaration einer Regel entspricht, die angibt, aus welchen Informationseinheiten ein Element zusammengesetzt sein kann. Durch Element-Deklarationen werden also nicht Informationen direkt ausgedrückt, sondern sie bilden strukturelle Information.

Im Falle eines Dokuments kann man demzufolge sagen, dass Regeln elementare Textteile, gewissermaßen die Wörter des Dokuments, so miteinander in Verbindung setzen, dass die Dokumentbedeutung transferierbar und rekonstruierbar wird. Was ist nun aber die Dokumentbedeutung? Auf diese Frage kann keine generelle Antwort gegeben werden, da Dokumentgrammatiken anders als die Grammatik natürlicher Sprachen ihrem jeweiligen Verwendungszweck angepasst sind. Oft geht es allerdings darum, die Charakteristika einer bestimmten Textsorte in der Dokumentgrammatik zu beschreiben, sodass diese bei der weiteren Verarbeitung eines Dokuments berücksichtigt werden können.

In den einzelnen Phasen des „Lebenszyklus“ von Dokumenten erweist sich die Verfügbarkeit einer Dokumentgrammatik in der folgenden Weise als nützlich:

- Produktion: Während der inkrementellen Produktion eines Dokuments verfolgen Editoren den Aufbau der Baumstruktur und gleichen ihn mit der Dokumentgrammatik ab. Es kann angegeben werden, welche Elemente an einer bestimmten Stelle strukturell eingefügt werden können, und es können falsche Elementeneinfügungen von vornherein verhindert werden.
- Validierung: Ein vorliegendes Dokument wird dahingehend überprüft, ob seine Baumstruktur sich mit den in der Grammatik enthaltenen Regeln ableiten lässt, die Baumstruktur des Dokuments also der Grammatik entsprechend aufgebaut ist.
- Retrieval: Das Auffinden von Informationen in Dokumenten kann durch Grammatiken unterstützt werden. Soll z.B. ein Textelement „Überschrift“ mit einem bestimmten Stichwort gesucht werden, kann die Suche auf solche Abschnitte des Dokuments beschränkt werden, in denen ein Element „Überschrift“ überhaupt nur vorkommen kann.

Wenn eine wesentliche Neuerung XML-basierter Informationsmodellierung darin besteht, normierte Dokumente mit Baumstruktur zu verwenden, so wird es wichtig, von vornherein abschätzen zu können, mit was für Arten von Bäumen in einem bestimmten Anwendungsbe- reich gerechnet werden kann. Für die Definition baumartiger Dokumentstrukturen gibt es verschiedene Möglichkeiten. Bäume können, wie wir gesehen haben, zum einen in ihrem Aufbau durch *Regeln* beschrieben werden. Die Regeln bilden zusammengenommen eine Grammatik. Da, anders als bei der Beschreibung von Wortketten der natürlichen Sprache, die beschriebenen Dokumentstrukturen in ihren Tags weiterhin die Baumstruktur abbilden, spricht man in diesem Fall auch von Baumgrammatiken. Bei diesem Ansatz steht die Idee Pate, dass man komplexe Gebilde durch eine Grammatik *konstruktiv* beschreibt, also ein Verfahren angibt, wie alle zulässigen Dokumentstrukturen mit Hilfe der Grammatik gebildet werden können.

Eine zweite Möglichkeit besteht darin, *Bedingungen* anzugeben, denen eine Klasse von Dokumenten unterliegen soll. Grundlage dieses Ansatzes bildet die Vorstellung, dass bestimmte Eigenschaften gefordert werden, die Dokumente aufweisen sollen, dass aber nicht die komplette Struktur definiert wird wie durch eine Grammatik. Anders als beim Regelbasierten Ansatz können Bedingungen über einer Baumstruktur völlig frei definiert werden – durch Regeln können dagegen immer nur Zusammenhänge innerhalb eines Teilbaums beschrieben werden. Werden so viele Bedingungen aufgestellt, dass jeder Aspekt des Dokuments spezifiziert wird, kann man natürlich auch von einer Grammatik sprechen. Gleichwohl ist einem auf Bedingungen basierenden Ansatz eine gewisse Liberalität bei den zu beschreibenden Strukturen inhärent.

Die dritte Möglichkeit bildet zugleich die einfachste und naheliegendste, nämlich die Spezifikation mittels *Beispieldokumenten*. Man kann dieses als einen Sonderfall des regel- oder

bedingungs-basierten Ansatzes verstehen, allerdings gibt es auch hier interessante Aspekte, die ihn von den anderen unterscheidet.

Sowohl für die Spezifikation durch Regeln als auch die durch Beispieldokumente wurden Vorschläge für Notationen und Verarbeitungsmethoden vorgelegt (vgl. Jeliffe 2000, [Schematron], Cagle *et al.* 2001 und [Examplotron]); diese spielen bislang allerdings bislang eine marginale Rolle, so dass von einer Diskussion dieser Ansätze an dieser Stelle abgesehen wird.

3 Evolution grammatischer Textauszeichnung

3.1 SGML und XML

Man kann ohne Übertreibung sagen, dass mit der Verabschiedung der *Standard Generalized Markup Language* als internationalem Standard ISO 8879 im Jahre 1986 die Informationstechnologie aus den Fesseln der technischen Abhängigkeiten befreit und ein Weg eröffnet wurde, Information ausschließlich auf der Grundlage ihrer inneren Gesetzmäßigkeiten und ihrer Funktion zu modellieren und zu verarbeiten. Die im Web verwendete Seitenbeschreibungssprache HTML konnte spektakulär demonstrieren, wie die maschinen- und softwareunabhängige Informationsmodellierung den Austausch, die Verknüpfung und die Manipulation von Daten in weltumspannender Weise möglich machen kann.

Schon bald aber machte die Entwicklung neuer Anwendungen und verwandter Standards sowie deren Umsetzung in Software-Systemen deutlich, dass für verteilte Informationssysteme wie dem Internet mit HTML nur ein erster Schritt bei der Informationsmodellierung unternommen worden war. Warum sollte statt dieser einen SGML-Anwendung nicht SGML insgesamt über das Internet nutzbar sein? Jede beliebige SGML-Anwendung könnte dann im Internet verfügbar werden und, darauf aufbauend, auch flankierende Standards wie HyTime oder DSSSL mit den dazugehörigen Software-Systemen¹.

So naheliegend dieser Gedanke war, so schwierig war es, ihn umzusetzen. Das World Wide Web war längst ein Massenmedium geworden, der SGML-Standard konnte darauf aber aufgrund einer Vielzahl überkomplizierter Details nicht angemessen reagieren. Darüber hinaus hatte sich ein undurchsichtiger SGML-Jargon entwickelt, der die Einarbeitung in diesen Standard zusätzlich erschwerte, da er selbst in den meisten einführenden Werken nicht vermieden wurde. Das größte Problem bestand aber darin, dass der Standard nicht nur kompliziert, sondern auch formal so komplex war, dass Online-Anwendungen Schwierigkeiten bekommen mussten, eine Verarbeitung in akzeptabler Zeit durchzuführen. Viele Eigenschaften von SGML spiegeln noch den Stand der frühen achtziger Jahre wider, in denen nicht abseh-

¹ HyTime ist ein Standard zur Spezifikation von Verlinkungen und zeitabhängigen Präsentationselementen, DSSSL ist eine Transformations- und Layout-Sprache für SGML-Dokumente.

bar war, dass SGML-Anwendungen anderswo als auf isolierten Einzelrechnern funktionieren könnten.

Das Hervortreten dieser Unzulänglichkeiten von SGML für Zwecke der Online-Anwendung war der Ursprung der *Extensible Markup Language*, die seit Anfang 1998 in einer vom *World Wide Web Consortium* verabschiedeten Fassung vorliegt. XML ist nichts anderes als eine vereinfachte Version von SGML, denn in XML kodierte Information bleibt zugleich auch SGML-kodiert. Die Definition von XML ist jedoch viel konziser, knapper und logisch überzeugender, da alles das, was in SGML zur Komplexität beitrug und ohnehin kaum genutzt wurde, weggelassen wurde, ohne dabei die Ausdrucksmöglichkeiten prinzipiell einzuschränken. Diese Reduktion war so überzeugend, dass XML bald auch dort eingesetzt wurde, wo die Online-Fähigkeit der Daten gar nicht im Vordergrund stand. Seitdem hat XML den voll ausgeformten SGML-Standard so stark zurückgedrängt, dass davon auszugehen ist, dass dieser bald nur noch von historischer Bedeutung sein wird.

3.2 Schemasprachen

Es ist bereits hervorgehoben worden, dass die Deklaration von Elementen durch Regeln eine der wichtigsten Neuerungen ist, die durch SGML und XML eingeführt worden war. Die Regeln zur strukturellen Definition eines Dokumenttyps bilden zusammengenommen eine Dokumentgrammatik, die als *Dokumenttyp-Definition* (DTD) bezeichnet wird. Das Aufkommen von sog. Schemasprachen mit dem Ziel der Ersetzung von DTDs ist auf drei Anforderungen zurückzuführen, die DTDs nicht erfüllen können:

1. XML-Notation: Da DTDs in einer proprietären Notation kodiert werden, können sie nicht selbst zum Gegenstand von XML-basierten Verarbeitungstechniken gemacht werden. Alle Schemasprachen verwenden, anders als DTDs, eine XML-Notation, sodass es möglich wird, Schemata mit dem gleichen Instrumentarium zu bearbeiten wie andere XML-Dokumente auch.
2. Reichhaltigere Datentypen: DTDs verfügen nur über ein sehr beschränktes Inventar vordefinierter Datentypen (z.B. #PCDATA). Die Definition neuer Datentypen ist mühsam und unflexibel. In vielen Schemasprachen wird dagegen ein Inventar an Datentypen angeboten, das sich an den Möglichkeiten moderner Programmiersprachen orientiert und leicht erweitert werden kann.
3. Mächtigere Strukturdefinitionen: Neuere Arbeiten zur Äquivalenz von XML und regulären Baumsprachen (vgl. z.B. Murata *et al.* 2000) haben gezeigt, dass mit DTDs nur eine sehr eingeschränkte Variante einer regulären Baumgrammatik realisiert wird. Mächtigere Varianten erlauben es, Inhaltsmodelle kontextabhängig zu spezifizieren – eine Anforderung, die oft in praktischen Strukturierungszusammenhängen besteht.

3.2.1 XML Schema

Nachdem über längere Zeit verschiedene Entwürfe von Schemasprachen parallel zueinander verfolgt wurden (vgl. Lee/Chu 2000), liegt mit XML Schema seit 2001 eine W3C-Empfehlung

vor, die mittlerweile von vielen Software-Systemen alternativ zu DTDs unterstützt wird. Die Deklaration des Elements `buch` etwa kann in XML Schema folgendermaßen aussehen (s. z.B. Cagle *et al.* 2001):

```
(6)  <element name="buch">
      <complexType>
        <sequence>
          <element ref="autor" maxOccurs="unbounded"/>
          <element ref="titel"/>
          <element ref="verlag" minOccurs="0"/>
          <element ref="kapitel" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
```

Ein Inhaltsmodell, der rechte Teil einer Regel, wird in XML Schema als das Element `complexType` modelliert. Der Häufigkeitsindikator wird in (6) durch die Attribute `maxOccurs` und `minOccurs` ausgedrückt, dabei können neben `unbounded` beliebige Zahlenwerte eingetragen werden. Innerhalb der `sequence`-Konstruktion wird über das Konstrukt `<element ref="autor"/>` usw. auf die Deklaration des Elements `autor` verwiesen. Dabei muss es sich innerhalb des Schema-Dokuments um ein global deklariertes Element handeln.

Während (6) im wesentlichen die DTD-Syntax in XML nachzeichnet, geht eine alternative Umsetzung über die Möglichkeiten von DTDs hinaus. Dabei wird das Element `autor` nicht als ein globales Element deklariert, sondern lokal. Sein Inhaltsmodell ist dabei nur für den vorliegenden Kontext – hier innerhalb des Elements `buch` – in der angegebenen Weise definiert, in anderen Kontexten kann dagegen für `autor` ein beliebiges anderes Inhaltsmodell definiert werden:

```
(7)  <element name="buch">
      <complexType>
        <sequence>
          <element name="autor" maxOccurs="unbounded">
            <complexType>
              <sequence>
                <element name="vorname" type="string"
                  maxOccurs="unbounded"/>
                <element name="name" type="string"/>
              </sequence>
            </complexType>
          </element>
          ...
        </sequence>
      </complexType>
    </element>
```

Die konsequente Verwendung lokaler Elementdeklarationen führt dazu, dass die Struktur von Dokumenten durch Schemata beschrieben werden kann, die schon recht deutliche Ähnlichkeit mit dem spezifizierten Dokument aufweisen. Ein solches Schema kann als eine schematische Definition im engeren Sinne aufgefasst werden, also als ein Dokumentmuster, das es noch zu konkretisieren gilt.

Mit der Formulierbarkeit kontextabhängiger Inhaltsmodelle geht XML Schema über die Mächtigkeit von DTDs hinaus – allerdings nur ein wenig: wie in DTDs muss der Grundsatz eingehalten werden, dass ein Schema keine Ambiguitäten beinhalten darf. Damit ist gemeint, dass bei der Anwendung eines Schemas auf ein Dokument immer eindeutig bestimmbar sein muss, welcher Teil des Inhaltsmodells zu beachten ist. Damit sind Deklarationen wie die folgende ausgeschlossen:

```
(8) <element name="kapitel">
      <complexType>
        <choice>
          <element name="absatz">
            (Inhaltsmodell 1)
          </element>
          <element name="absatz">
            (Inhaltsmodell 2)
          </element>
        </choice>
      </complexType>
    </element>
```

Da in XML Schema also die unterschiedlich definierten Elemente gleichen Namens nicht innerhalb eines Inhaltsmodells miteinander in Konkurrenz stehen dürfen, wird diese Variante einer regulären Baumgrammatik von Murata *et al.* (2001) als *single-type tree grammar* bezeichnet.

3.2.2 Das Typenkonzept von XML Schema

Obwohl es bei der Spezifikation von Dokumentgrammatiken in SGML und XML immer um die Definition der Zusammenhänge zwischen Informationseinheiten geht, letztlich also eine datenorientierte Perspektive eingenommen wird, wird das dafür besonders geeignete Konzept der Objektorientierung erst bei XML Schema systematisch in eine Schemasprache integriert.

Bei der objektorientierten Programmierung stehen nicht die Abläufe, sondern die Daten im Mittelpunkt. Datentypen werden zu Klassen zusammengefasst, diese durch Eigenschaften beschrieben und mit Prozeduren und Funktionen (Methoden) versehen. Klassen können hierarchisch angeordnet werden, dabei werden Eigenschaften und Methoden übergeordneter Klassen auf untergeordnete vererbt, die untergeordneten Klassen spezifizieren die übergeordneten. Klassen können aber auch in einer Teil-Ganzes- oder einer Abhängigkeitsrelation zueinander stehen. Die Klassen werden bei einer Verwendung des Programms durch Objekte, die realen Daten entsprechen, instanziiert.

Grundsätzlich besteht zwischen jeder Deklaration eines Elements in einer Schemasprache und dem durch Tags markierten Auftreten eines Elements in einem Dokument eine In-

stanzierungsbeziehung wie die zwischen Klasse und Objekt, weshalb von Element-Deklarationen genauer auch von Elementtyp-Deklarationen gesprochen werden muss. In XML Schema werden Elementtypen zusätzlich jedoch wie Klassen in einen hierarchischen Zusammenhang gebracht. Übergeordnete Klassen werden durch ein Element `complexType` definiert, aus denen durch das Element `element` solche Klassen abgeleitet werden, die sich in Dokumenten als Objekte, also konkrete Elemente, instanziierten lassen.

Im folgenden Beispiel wird zunächst der Typ `kapitelTyp` deklariert, danach ein Elementtyp `kapitel`:

```
(9) <complexType name="kapitelTyp">
    <sequence>
      <element name="überschrift" type="string"/>
      <element name="absatz" maxOccurs="unbounded"
        type="string"/>
    </sequence>
</complexType>

<element name="kapitel" type="kapitelTyp"/>
```

Ein Elementtyp kann aber auch durch Erweiterung oder Einschränkung aus einem komplexen Typ hervorgehen:

```
(10) <element name="kapitelmitbibliografie">
    <complexType>
      <complexContent>
        <extension base="kapitelTyp">
          <sequence>
            <element ref="bibliografie"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

In diesem Fall wird durch den Wert des Attributs `base` in der `extension`-Konstruktion auf den Basistyp verwiesen. Das darin definierte Inhaltsmodell wird dem Inhaltsmodell des Basistyps sequenziell angehängt. Im Dokument muss somit beim Element `kapitelmitbibliografie` die Sequenz der Elemente `überschrift`, `absatz` und `bibliografie` erscheinen. Die Deklaration des komplexen Typs geschieht in (10) ohne eine explizite Benennung des entstehenden Typs, weshalb eine derartige Konstruktion auch eine anonyme Typ-Definition genannt wird.

Eine Reihe von Kontrollattributen erlaubt es dem Schema-Designer, die Ableitung aus komplexen Typen feiner zu steuern. So kann etwa festgelegt werden, dass aus einem komplexen Typ nicht unmittelbar ein Elementtyp abgeleitet werden kann oder, im Gegenteil, genau dieses der Fall sein muss.

Bei der Ableitung aus einem komplexen Typ wird, wie wir gesehen haben, das Inhaltsmodell und die Menge der Attribute vererbt. Durch Erweiterung oder Einschränkung kann diese

Information modifiziert werden. Hinsichtlich des Inhaltsmodells ist natürlich durchaus denkbar, auch interessantere Erweiterungsformen zuzulassen als lediglich das Anhängen von Elementen. Da dieses jedoch die Vergrößerung der syntaktischen Komplexität mit sich gebracht hätte, wurde bei der Entwicklung von XML Schema bewusst darauf verzichtet.

Explizit nicht zu den Eigenschaften eines komplexen Typs gehört seine relative Stellung gegenüber anderen Elementen. Das hat zur Folge, dass die Verwendung eines Verweises auf einen komplexen Typ in einem Inhaltsmodell verboten ist:

```
(11) <sequence>
      <element ref="autor" maxOccurs="unbounded"/>
      <element ref="titel"/>
      <element ref="verlag" minOccurs="0"/>
      <complexType ref="kapitelTyp" maxOccurs="unbounded"/>
    </sequence>
```

Sinnvoll könnte eine derartige Verwendungsweise durchaus sein: an alle aus `kapitelTyp` abgeleiteten Typen könnte die sequenzielle Position vererbt werden, für abgeleitete Typen könnten darüberhinaus spezielle Verwendungsweisen definiert werden.

4 Einfache und komplexe Dokumentgrammatiken

Die Entwicklung von Dokumentgrammatiken hat sich offensichtlich in drei Schritten vollzogen. Der erste Schritt, die Entwicklung von SGML, war von der Notwendigkeit geprägt, trotz der Beschränktheit von Speicherkapazitäten eine vollständige Textauszeichnung möglich zu machen. Dazu wurde ein kompliziertes System von Minimierungsregeln entworfen, durch die die Auslassung von strukturell erschließbaren Tags erlaubt wurde. So können in SGML etwa die Tags für ein Element `überschrift` ausgelassen werden, sofern es in der DTD als obligatorisch gekennzeichnet ist:

```
(12) <kapitel>Verhaftung - Gespräch mit Frau Grubach - Dann Fräulein
      Bürstner<absatz>...</absatz>
      ...
    </kapitel>
```

Dem Vorteil der Reduktion der abzuspeichernden Daten steht eine Erhöhung des Verarbeitungsaufwands entgegen: Da durch die Auslassung der Tags für das Element `überschrift` die korrekte Baumstruktur des Dokuments nicht mehr vollständig repräsentiert wird, muss diese in einem dazwischengeschalteten Verarbeitungsschritt rekonstruiert werden.

Nicht die Effizienz der Datenkodierung, sondern die der Grammatik steht bei einer zweiten Besonderheit von SGML im Vordergrund. Dabei wird bei einem übergeordneten Element eine durch `+` gekennzeichnete Inklusion definiert:

```
(13) <!ELEMENT kapitel (überschrift, absatz+) +(kommentar)>
```

Die Inklusion des Elements `kommentar` wirkt sich dahingehend aus, dass sowohl im Element `kapitel` als auch in allen untergeordneten Elementen – hier `überschrift` und `absatz` sowie ggfs. deren Unterelemente – das Element `kommentar` beliebig oft an beliebiger Stelle erscheinen darf. Ist etwa das Element `absatz` als reines Datenelement deklariert, so wird das Inhaltsmodell durch die Inklusion in folgender Weise abgewandelt:

```
(14) <!ELEMENT absatz (#PCDATA|kommentar)*>
```

Inklusionen haben also die Wirkung von Metaregeln, sie sind Regeln, die den Aufbau anderer Regeln beeinflussen.

Sowohl die Verkürzung des Markups im Dokument als auch die Deklaration von Metaregeln bewirken in Hinsicht auf die Kosten der Ressource Speicher eine Verbilligung, in Hinsicht auf die Ressource Verarbeitungsaufwand allerdings eine extreme Verteuerung. Dass dieser Aspekt bei der Definition von SGML zu wenig berücksichtigt worden ist, wird insbesondere durch die Tatsache deutlich, dass die Tag-Minimierung sich auf die formalsprachliche Einordnung der durch SGML definierbaren Baumgrammatiken auswirkt. Werden Tags regulär ausgelassen, wird der Bereich der verhältnismäßig effizient maschinell zu analysierenden Baumsprachen verlassen und partiell in den Bereich der kontextfreien Sprachen hinübergewechselt. Für diese Kategorie von formalen Sprachen sind wiederum ganz andere Analysealgorithmen einzusetzen.

Im zweiten Schritt der Entwicklung von Dokumentgrammatiken wurde bei XML die Effizienz der Speicher-Inanspruchnahme durch die Effizienz der Verarbeitung ersetzt. XML verzichtet auf Minimierung und Metaregeln, weshalb XML-Dokumente einen erheblich größeren Umfang aufweisen können als SGML-Dokumente, in denen das Markup geschickt minimiert wurde. Die Überprüfung der grammatischen Korrektheit von XML-Dokumenten in Bezug auf eine DTD lässt sich mit sehr einfachen Algorithmen mit einem Zeitaufkommen durchführen, das sich linear zur Länge des Dokuments entwickelt und somit einen vertretbaren Verbrauch von Verarbeitungsressourcen etwa im Rahmen von Web-Anwendungen gewährleistet. Dabei stellte sich der interessante Effekt ein, dass durch die Reduktion an grammatischen Spezifikationstechniken eine formale Reinheit zu Tage trat, die überhaupt erst die Dokumentgrammatiken als Varianten der in der theoretischen Informatik schon seit längerer Zeit bekannten Baumgrammatiken erkennbar werden ließ (vgl. Mönnich/Morawietz 2003). Mit dieser Erkenntnis erschloss sich plötzlich ein reichhaltiges Inventar an Algorithmen, die auf Baumgrammatiken und -sprachen anwendbar sind (vgl. Murata 1998). Die formale Reduktion, die bei der Spezifikation von XML gegenüber SGML durchgeführt worden war, resultierte also nicht nur in einer effizienteren Verarbeitung, sondern auch in einer Ausweitung der verfügbaren Verarbeitungstechniken überhaupt.

Der dritte Schritt ist gekennzeichnet durch die bewusste Erweiterung der Ausdrucksmächtigkeit und der Übernahme weitergehender Konzepte auf der Grundlage nunmehr gesicherter Erkenntnisse zum formalen Status von Dokumentgrammatiken. Dabei sind zwei Tendenzen zu beobachten: Auf der einen Seite stehen die Bemühungen, die formalen Möglich-

keiten von Baumgrammatiken in Schemasprachen umzusetzen. Während XML Schema die eher restringierte Variante einer Baumgrammatik implementiert, gehen Sprachen wie etwa RELAX NG weit darüber hinaus. Als die unrestringierte Variante einer Baumgrammatik ist es in dieser Schemasprache möglich, mehrere Dokumentgrammatiken miteinander zu vereinigen oder voneinander abzuziehen. Welche Akzeptanz derartige Funktionen erlangen, muss sich allerdings im realen Anwendungszusammenhang erweisen.

Auf der anderen Seite ist das bereits erwähnte Typenkonzept in XML Schema zu sehen. Durch dieses Konzept wird nicht die Ausdrucksmächtigkeit der Dokumentgrammatik beeinflusst, sondern die Struktur der Grammatik selbst. Durch die Einführung von abstrakten Elementebenen, die selbst nicht im Dokument erscheinen können, können Zusammenhänge in der grammatischen Struktur dargestellt werden, die in DTDs nicht ausgedrückt werden können.

Man kann sich fragen, was gewonnen wird, wenn die Ausdrucksmächtigkeit von Dokumentgrammatiken erhöht wird. Die eigentlichen textuellen Daten, die Dokumentinstanzen, verändern sich bei all dem nicht. Auch ist es für Anwendungen in diesem Bereich unerheblich, ob eine Grammatik alle korrekten Sätze der Sprache aufzuzählen in der Lage ist, wie es von formalen Grammatiken für natürliche Sprachen gefordert wird. Die „Sätze“ von Dokumentgrammatiken sind die Dokumente, und in ihnen wird die relevante Information durch den Elementinhalt, also gewissermaßen durch die „Wörter“ des „Satzes“, repräsentiert. Diese „Wörter“ unterliegen nun aber linguistischen und inhaltlichen Regularitäten, die ihrerseits nicht in der Dokumentgrammatik repräsentiert werden. Insofern bezieht sich eine Dokumentgrammatik also lediglich auf eine strukturelle Ebene, sie macht Aussagen über mögliche Strukturen von Texten, nicht aber über mögliche Texte selbst.

Die Erhöhung der grammatischen Ausdrucksmächtigkeit wirkt sich also aus auf die Präzision der Spezifikation von Textstrukturen, der formalen Seite von Textsorten. Anders als Grammatiken für natürliche Sprachen geht es nicht darum, strukturelle Ambiguitäten erklären zu können – ein XML-Dokument darf *per definitionem* keine Ambiguitäten enthalten. Die Verbesserung grammatischer Strukturierungsmethoden zielt vielmehr auf die Erklärung der verwendeten Kategorien, der Elemente, ab, da deren Funktion und Semantik nicht aus übergeordneten Erwägungen, wie es etwa für Wortarten oder Phrasentypen der Fall ist, abgeleitet werden können. Das Typenkonzept von XML Schema bietet dafür eine partielle Antwort, wenn auch keine umfassende Methodik an. Nicht von ungefähr bildet die Erforschung der Semantik von Dokumentgrammatiken und –strukturen zur Zeit ein zentrales Forschungsfeld im Gebiet der Texttechnologie (vgl. Kimber/Heintz 2001, Erdmann 2001, Lobin/Lemnitzer 2003).

5 Fazit

Wir haben gesehen, dass Dokumentgrammatiken anderen Bedingungen unterliegen als Grammatiken für natürliche Sprachen. Der wesentliche Unterschied entsteht durch die eindeutige Auszeichnung einer baumartigen Anordnung der textuellen Daten auf der Grundlage

standardisierten Markups. Die Entwicklung der Formalismen zur Spezifikation derartiger Markup-Systeme gleicht einem evolutionären Prozess, im Laufe dessen sich das Konzept der Baumgrammatik als ein dominierendes Merkmal erwiesen hat. Durch Reduktion ursprünglicher Komplexität wurde formale Reinheit gewonnen, die sich ihrerseits durch einen Gewinn an algorithmischer Durchdringung der Dokumentverarbeitung auszahlt. Aktuelle Entwicklungen versuchen neue Komplexität auf einen unveränderten formalen Kern aufzusetzen, um das bisher ungelöste Problem der konzeptionellen Axiomatisierung der in Dokumentgrammatiken auftretenden Kategorien anzugehen.

6 Literatur

- Cagle, Kurt, Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Nikola Ozu, Ian Stokes-Rees, Jeni Tennison und Kevin Williams (2001): *Professional XML Schemas*. Birmingham: Wrox Press.
- Erdmann, Michael (2001): *Ontologien zur konzeptuellen Modellierung der Semantik von XML*. Nordestedt: Book on Demand.
- Horrocks, I., D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer und E. Motta (2000): *The Ontology Inference Layer OIL*. [<http://www.cs.vu.nl/~dieter/oil/Tr/oil.pdf>]
- Jeliffe, Rick (2000): *Schematron*. [<http://www.ascc.net/xml/resource/schematron>]
- Kimber, Eliot und John Heintz (2001): "Using UML to define XML document types". In *Markup Languages 2/3*, 295-320.
- Lee, Dongwon und Wesley W. Chu (2000): Comparative Analysis of Six XML Schema Languages. [<http://www.cs.ucla.edu/~dongwon/paper>]
- Lobin, Henning (2000): *Informationsmodellierung in XML und SGML*. Berlin, Heidelberg: Springer-Verlag [2. Aufl. 2001].
- Lobin, Henning (2003): „Textauszeichnung und Dokumentgrammatiken“. In Lobin/Lemnitzer (Hrsg.), 51-82.
- Lobin, Henning, und Lothar Lemnitzer (2003, Hrsg.): *Texttechnologie. Perspektiven und Anwendungen*. Tübingen: Stauffenburg.
- Mani, Murali, Dongwon Lee und Richard R. Muntz (2001): "Semantic Data Modeling using XML Schemas".
- Mönnich, Uwe, und Frank Morawietz (2003): „Formale Grundlagen“. In Lobin/Lemnitzer (Hrsg.), 109-141.
- Murata, Makoto (1998): "Data Model for Document Transformation and Assembly". In *Proc. of PODDP 1998*.
- Murata, Makoto, Dongwon Lee und Murali Mani (2000): "Taxonomy of XML Schema Languages using Formal Language Theory". In *Proc. of Extreme Markup Languages 2000*.
- Rehm, Georg (2002), "Towards Automatic Web Genre Identification – A Corpus-Based Approach in the Domain of Academia by Example of the Academic's Personal Homepage" In *Proceedings of the Hawai'i International Conference on System Sciences (HICSS-35)*, January 7-10, 2002, Big Island, Hawaii.
- Rehm, Georg (2003), "Texttechnologische Grundlagen". In: Ralf Klabunde, Kai-Uwe Carstensen, Christian Ebert, Cornelia Endriss, Susanne Jekat, Hagen Langer und Michael Schiehlen (Hrsg.), *Computerlinguistik und Sprachtechnologie – Eine Einführung*. Heidelberg: Spektrum Akademischer Verlag. 2. Auflage
- Sperberg-McQueen, Michael, Claus Huitfeldt und Allen Renear (2000): "Meaning and interpretation of markup". In *Markup Languages 2/3*, 215-234.

-
- [Examplotron] (2001): Examplotron. [<http://examplotron.org/>]
- [HTML] (1999): *HTML 4.01 Specification*. World Wide Web Consortium. [<http://www.w3.org/TR/html4/>]
- [RELAX NG] (2001): *RELAX NG Specification. Committee Specification 11 August 2001*. OASIS – Organization for the Advancement of Structured Information Standards. [<http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>]
- [SGML] (1986): *ISO 8879:1986. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. Genf: International Organization for Standardization.
- [XML Schema] (2001): *XML Schema*. World Wide Web Consortium. [<http://www.w3c.org/XML/Schema>]
- [XML] (2000): *Extensible Markup Language (XML) Version 1.0 (Second Edition)*. World Wide Web Consortium. [<http://www.w3c.org/TR/2000/REC-xml-20001006>]