Originalveröffentlichung in: Lobin, Henning/Lemnitzer, Lothar (Hrsg.): Texttechnologie. Perspektiven und Anwendungen. - Tübingen: Stauffenburg Verlag, 2004. S. 53-82. (Stauffenburg Handbücher)

Textauszeichnung und Dokumentgrammatiken

Henning Lobin

1 Einleitung

Ein zentrales Mittel der Texttechnologie bildet die Textauszeichnung. Unter Textauszeichnung wird die Einfügung von Markierungen in einen Text verstanden, aufgrund derer die auf diese Weise ausgezeichneten Textteile in spezieller Weise verarbeitet werden können (vgl. Sperberg-McQueen et al., 2000). 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. als 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 Hypertextsystem kann durch Links auf eindeutige Textstellen verwiesen werden – die Textauszeichnung wird also für Browsing-Zwecke verwendet (siehe Storrer, in diesem Band);
- eine Gruppe von gleich ausgezeichneten 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*.

Publikationsserver des Instituts für Deutsche Sprache
URN: http://nbn-resolving.de/urn:nbn:de:bsz:mh39-76184

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 Textauszeichnungsvokabularien, 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". Eine vereinfachte Version von SGML, die *Extensible Markup Language* (XML), hat heute SGML weitgehend ersetzt.

2 Baumstrukturen und Markup

Eine der grundlegenden Neuerungen im Bereich des Dokument-Markup, die durch SGML eingeführt wurde und auch die Grundlage von XML bildet, besteht darin, die durch Tags annotierten 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", drei Absatz- und einem Abbildungselement bestehen. Das Kapitelelement seinerseits konstituiert mit anderen Elementen ein Textelement „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 z. B. festgelegt werden, dass diese Eigenschaft auch allen diesem Element untergeordneten Elementen zukommt, sofern bei ihnen nicht explizit etwas anderes vermerkt ist. Das hier angewendete Prinzip der *Vererbung* von Eigenschaften ist also ein Mittel, diese 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 über-

nommen werden. Bei der Überprüfung beispielsweise, ob ein Dokument auch eine korrekt aufgebaute Baumstruktur aufweist, können Verfahren angewendet werden, die für die Analyse von Programmen entwickelt worden sind (Compilerbau).

Aus Sicht der Linguistik ist jedoch der wichtigste Vorteil der Verwendung von Baumstrukturen, dass man diese durch *Grammatiken*[1] beschreiben kann, und Grammatiktheorie und Grammatikformalismen in der Linguistik als gut untersuchte Gebiete gelten können. Den Wert einer Grammatik für die Verarbeitung strukturierter Dokumente kann man sich am besten deutlich machen, wenn man sich die Funktion von Grammatiken im Zusammenhang mit der natürlichen Sprache vor Augen führt.

Wir nutzen die Regeln der Grammatik einer Sprache, um eine komplexe, oftmals von uns in der Gesprächssituation neu erschaffene Bedeutung auf der Grundlage von Wörtern in eine Laut- oder Buchstabenfolge zu kodieren und sie in dieser Form an Kommunikationspartner zu übermitteln. Da auch dem Kommunikationspartner die Wörter und die Grammatikregeln bekannt sind, ist er in der Lage, aus den wahrgenommenen Laut- und Buchstabenfolgen eine Bedeutung zu rekonstruieren. Die grammatischen Regeln bilden also einen Rahmen zur Rekonstruktion komplexer Bedeutung ausgehend von kleinen Bedeutungseinheiten (Kompositionalitätsprinzip sprachlicher Bedeutung).

Im Falle eines Dokuments kann man demzufolgen 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, so dass diese bei der weiteren Verarbeitung eines Dokuments berücksichtigt werden können. Ein Gedicht hat einen anderen Aufbau als ein wissenschaftliches Papier, und dieser schlägt sich z. B. im Textdesign oder in der Terminologie (Überschrift, Abstract, Einleitung, Literaturliste gegenüber Titel, Strophe, Vers) nieder.

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

---

[1] Ist lediglich die Liste der Elemente eines Dokumenttyps gemeint, so spricht man auch von „Tagsets" oder „Vokabularien".

- *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 falsche Elementeinfügungen können 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 konform zur Grammatik ist.
- *Retrieval:* Das Auffinden von Informationen in Dokumenten kann durch Grammatiken unterstützt werden. Soll beispielsweise 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.
- *Textparsing:* Soll ein nicht explizit strukturiertes Dokument maschinell analysiert werden, spricht man auch von Textparsing. Eine schwächere Form des Textparsing bildet die Textkategorisierung, bei der es um die Zuordnung zu bestimmten Text- oder Dokumenttypen geht. In beiden Fällen können die beteiligten Prozesse durch Dokumentgrammatiken eingegrenzt und gesteuert werden. Wenn beispielsweise in der Dokumentgrammatik definiert ist, dass nach dem Autorennamen nur ein Abstract oder eine Überschrift erscheinen kann, können Erkennungsroutinen aktiviert werden, die Abstracts von Überschriften zu unterscheiden erlauben. Die Dokumentgrammatik enthält in diesem Fall Hypothesen über den Aufbau des Dokuments, durch die der Suchraum für die Parsing- und Kategorisierungsprozesse eingeschränkt wird.

Wenn eine wesentliche Neuerung XML-basierter Informationsmodellierung darin besteht, normierte Dokumente mit Baumstrukturen zu verwenden, so wird es wichtig, von vornherein abschätzen zu können, mit was für Arten von Bäumen in einem bestimmten Anwendungsbereich gerechnet werden kann. Für die Definition baumartiger Dokumentstrukturen gibt es verschiedene Möglichkeiten. Bäume können zum einen in ihrem Aufbau durch *Regeln* beschrieben werden. Die Regeln bilden zusammengenommen eine Grammatik. Da, anders als in der natürlichen Sprache, die beschriebenen Dokumentstrukturen in ihren Tags weiterhin die Baumstruktur eindeutig beschreiben, 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 Baumstrukturen völlig frei definiert werden, da Regeln immer auf die Zusammenhänge innerhalb eines Teilbaums beschränkt bleiben müssen. 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 bedingungsbasierten Ansatzes verstehen, allerdings gibt es auch hier interessante Aspekte, die diesen Ansatz von den anderen unterscheidet.

3 Baumgrammatiken und Schemasprachen

3.1 Mächtigkeit und Varianten von Baumgrammatiken

In der Theorie der formalen Sprachen werden Grammatiken eingesetzt, um Mengen von Zeichenketten zu definieren. Eine solche Menge wird eine *Sprache* genannt. Wenn es umgekehrt gelingt, ein Regelwerk zu finden, das eine bestimmte Menge von Zeichenketten erzeugen kann, spricht man von einer *Grammatik* für diese Sprache. Die für diesen Zweck in der Grammatik genutzten Regeln werden als *Produktionsregeln* bezeichnet (vgl. z. B. Carstensen et al., 2001).

Eine ähnliche Situation liegt vor, wenn Baumstrukturen erzeugt werden sollen. Eine Menge von Baumstrukturen heißt eine reguläre Baumsprache, eine Grammatik, die diese Baumsprache generiert, eine reguläre Baumgrammatik. Da die in SGML und XML verwendeten Baumgrammatikregeln wie die Regeln einer kontextfreien Grammatik aussehen, ist der Umstand, dass damit eine andere Art von Sprache generiert wird, oft übersehen worden.

Sehen wir uns dazu ein Beispiel an. In (1) ist ein Symbol dargestellt, das in die Zeichenkette „ab“ überführt wird:

$$A \rightarrow ab \tag{1}$$

Auf der rechten Seite der Regel können auch Symbole erscheinen, die in anderen Regeln auf der linken Seite verwendet werden (non-terminale Symbole):

$$\begin{aligned} A &\rightarrow AB \\ A &\rightarrow C \\ B &\rightarrow BB \\ B &\rightarrow C \\ B &\rightarrow x \\ C &\rightarrow \emptyset \end{aligned} \tag{2}$$

Diese Grammatik erlaubt die schrittweise Ableitung z. B. der folgenden Zeichenkette:

$$\begin{aligned} &A \\ &AB \\ &CBB \\ &\emptyset xx \end{aligned} \tag{3}$$

Durch die Anwendung der Regel einer Baumgrammatik dagegen wird nicht nur eine Zeichenkette gebildet, sondern auch eine Baumstruktur aufgebaut. In XML-Notation lässt sich das folgendermaßen darstellen:

$$A \rightarrow \texttt{<A>}ab\texttt{</A>} \tag{4}$$

Gewöhnlich wird A als Elementtyp bezeichnet, `<A>` als Anfangstag, `</A>` als Endtag und „ab“ als Inhaltsmodell. Werden auf der rechten Regelseite nonterminale Symbole verwendet, führt die wiederholte Anwendung der Regeln zu einer Verschachtelung der Tags und somit zu einer Baumstruktur. Eine reguläre Baumgrammatik mit der gleichen Regelstruktur wie die kontextfreie Grammatik (1) sieht folgendermaßen aus:

1. A → <A>AB</A> (5)
2. A → <A>C</A>
3. B → <B>BB</B>
4. B → <B>C</B>
5. B → <B>x</B>
6. C → <C>∅</C>

Für Elemente, deren Inhaltsmodell leer bleibt, so dass zwischen dem Anfangs- und dem Endtag nur der leere String stehen kann, wird in XML eine besondere Notation verwendet:

6′. C → <C/> (6)

Mit (5) und (6) lässt sich somit das folgende 'Dokument' ableiten:

<A>AB</A> (7)
<A><A>C</A>B</A> (nach Regel 2)
<A><A>C</A><B>BB</B></A> (nach Regel 3)
<A><A><C/></A><B>BB</B></A> (nach Regel 6′)
<A><A><C/></A><B><B>x</B>B</B></A> (nach Regel 5)
<A><A><C/></A><B><B>x</B><B>x</B></B></A> (nach Regel 5)

Die eingerückte Darstellung lässt die Baumstruktur besser erkennen:

```
<A>                                    (8)
    <A>
        <C/>
    </A>
    <B>
        <B>x</B>
        <B>x</B>
    </B>
</A>
```

Die in (5) und (6) dargestellte Grammatik enthält ausschließlich Regeln, bei denen das auf der linken Seite erscheinende Symbol zugleich den Tagnamen bildet. Diese Beschränkung ist nicht notwendig – auf der linken Regelseite

können durchaus Symbole verwendet werden, die mit anders bezeichneten Tags kombiniert werden. In (9) etwa werden in Regel 3 die Symbole B1 und B2 eingeführt, die in den Regeln 4 und 5 beide mit dem Tag <B> verbunden werden:

1. A → <A>AB</A> (9)
2. A → <A>C</A>
3. B → <B>B1B2</B>
4. **B1** → <B>**C**</B>
5. **B2** → <B>**x**</B>
6. C → <C/>

Mit dieser Grammatik lässt sich der Baum

<A><A><C/></A><B><B><C/></B><B>**x**</B></B></A> (10)

ableiten, nicht aber der Baum (11):

*<A><A><C/></A><B><B>**x**</B><B>**x**</B></B></A> (11)

Auf diese Weise ist es möglich, für ein und dasselbe Tag – hier <B> – Ausprägungen zu definieren, da für jede dieser Ausprägungen ein anderes abstraktes Symbol in einer Regel verwendet werden kann – hier B1 und B2. Die bisher verwendeten Schema-Sprachen machen in unterschiedlichem Maße von dieser Möglichkeit Gebrauch.

3.2 Das traditionelle Konzept in SGML und XML: DTDs

Sowohl XML als auch sein Vorläuferstandard SGML definieren nicht nur, wie die Baumstruktur eines Dokuments formal zu notieren ist, beide Standards beinhalten auch eine Sprache zur Definition einer eingeschränkten regulären Baumsprache. Die Einschränkung besteht darin, dass in den Regeln der Grammatik keine abstrakten Symbole gebraucht werden dürfen, folglich durch jedes Symbol, das auf der linken Seite einer Regel erscheint, zugleich auch die Bezeichnung der Tags auf der rechten Regelseite festgelegt wird. Nach Murata et al. (2000) wird diese Einschränkung regulärer Baumgrammatiken als *Local Tree Grammar* bezeichnet.

Zusätzlich existieren in beiden Standards verschiedene Variablen, die für beliebige Dateninhalte stehen. Die wichtigste dieser Variablen ist #PCDATA, die zur Markierung beliebiger Elementinhalte verwendet werden kann. Sehen wir uns eine leicht modifizierte Beispielgrammatik an:

1. A → <A>BBCD</A> (12)
2. A → <A>#PCDATA</A>
3. B → <B>C#PCDATA</B>
4. C → <C>a</C>
5. C → <C>b</C>
6. D → <D/>

Eine Umsetzung dieser Grammatik in die Notation einer SGML-DTD (*Document Type Definition*) kann folgendermaßen aussehen:

```
1.  <!ELEMENT   A   ((B, B, D)|#PCDATA)>          (13)
2.  <!ATTLIST   A   C   (a|b)   #IMPLIED>
3.  <!ELEMENT   B   (#PCDATA)>
4.  <!ATTLIST   B   C   (a|b)   #REQUIRED>
5.  <!ELEMENT   D   EMPTY>
```

Da in DTDs non-terminale Symbole das verwendete Tag eindeutig determinieren, wird auf die explizite Definition des Tags in einer Element-Regel verzichtet, weshalb sie von einer kontextfreien Grammatikregel auf den ersten Blick nicht zu unterscheiden ist.

Zwei Besonderheiten bedürfen näherer Erläuterung: Während Regeln, die durch `<!ELEMENT` eingeleitet werden, Inhaltsmodelle (rechte Regelteile) aufweisen, die aus non-terminalen Symbolen oder der Variablen `#PCDATA` bestehen, ist eine durch `<!ATTLIST` (Attributliste) eingeleitete Regel für solche Fälle vorgesehen, in denen auf der rechten Seite nur terminale Symbole erscheinen. Derartige Regeln werden in einer sog. Attribut-Definition zusammengefasst. Im annotierten Dokument erscheinen diese Symbole innerhalb des Anfangstags des umgebenden Elements:

```
<A C="a">                                          (14)
    <B C="b">beliebiger Text</B>
    <B C="a">ebenfalls beliebiger Text</B>
    <D/>
</A>
```

Den Sinn von Attributen kann man sich am besten in dieser Form verständlich machen. Sie werden gewöhnlich verwendet, um variable Zusatz- und Metainformationen zu kodieren. Die Tatsache, dass in der DTD immer eindeutig festgelegt sein muss, ob eine Information als Elementinhalt

oder als Attributwert kodiert wird, bildet faktisch eine Reihe weiterer formaler Restriktionen der als DTD kodierbaren Baumgrammatiken, die hier nicht weiter diskutiert werden sollen. Grundsätzlich gilt aber, dass Element-Deklarationen und Attribut-Definitionen nicht miteinander 'vermischt' werden dürfen, Attribut-Definitionen dürfen aber bei einem Element kumuliert werden. Auch die Reihenfolge der Attribute untereinander gilt als undefiniert, ebenso wie das sequenzielle Verhältnis gegenüber dem Elementinhalt. Ein mit `#IMPLIED` qualifiziertes Attribut kann in einem Anfangstag auch ausgelassen werden.

Die zweite Besonderheit von DTDs besteht darin, dass es für einen Elementtyp nicht mehrere Deklarationen geben darf. Konkurrierende Regeln müssen deshalb wie in Regel 1. durch eine Oder-Konstruktion vereinigt werden. Grundsätzlich können im rechten Regelteil, dem Inhaltsmodell, reguläre Ausdrücke erscheinen, sofern diese lediglich durch Klammerung, den Oder-Konnektor `|`, den Sequenzkonnektor `,` sowie den Iterationskonnektoren `?` (null- oder einmal), `+` (beliebig oft) und `*` (nullmal oder beliebig oft) gebildet sind.

In (13) ist in Regel 1 auch ein Beispiel für einen der Unterschiede zwischen SGML und XML enthalten. Während es in SGML grundsätzlich keine Beschränkungen gibt, wie die Variable `#PCDATA` in einem Inhaltsmodell verwendet werden darf, ist für XML aus Gründen der Verarbeitungseffizienz festgelegt worden, dass `#PCDATA` nur allein im Inhaltsmodell oder als Teil einer iterierten Oder-Gruppe erscheinen darf. In XML wäre diese DTD also folgendermaßen umzuformulieren:

1. `<!ELEMENT A (#PCDATA|B)*>` (15)

2.–4. wie in (13)

Damit werden zwar die gleichen Konstruktionen wie in (13) zugelassen, darüber hinaus allerdings auch eine Vielzahl weiterer Kombinationen von B-Element und `#PCDATA`.

SGML- und XML-DTDs sind ein einfacher Formalismus zur Definition lokaler Baumgrammatiken, der wegen des langen Fehlens mächtigerer Alternativen sehr weit verbreitet ist.

3.3 Schemasprachen

Idee

Das Aufkommen von Schemasprachen mit dem Ziel der Ersetzung von DTDs seit Mitte der neunziger Jahre 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, so dass 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. Die Definition neuer Datentypen ist mühsam und unflexibel. In vielen Schemasprachen wird dagegen ein Inventar an Datentypen angenommen, 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.

Nachdem über längere Zeit verschiedene Entwürfe von Schemasprachen parallel zueinander verfolgt wurden (vgl. Lee und Chu, 2000), liegt mit XML Schema seit 2001 eine W3C-Empfehlung vor, die mittlerweile von vielen Software-Systemen alternativ zu DTDs unterstützt wird. Eine radikalere Variante stellt RELAX NG dar, die im Folgenden als zweite Schemasprache dargestellt werden soll.[2]

XML Schema

Die Deklaration des Elements A nach den Regeln 1 bis 4 in (15) kann in XML Schema folgendermaßen aussehen (siehe z. B. Cagle et al., 2001):

[2] Weitere Schemasprachen sind etwa XDR, SOX, DSD, TREX oder XDuce.

```
<element name="A">                                          (16)
    <complexType mixed="true">
        <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="B"/>
        </choice>
        <attribute name="C">
            <simpleType>
                <restriction base="NMTOKEN">
                    <enumeration value="a"/>
                    <enumeration value="b"/>
                </restriction>
            </simpleType>
        </attribute>
    </complexType>
</element>
```

Ein Inhaltsmodell, der rechte Teil einer Regel, wird in XML Schema als das Element `complexType` modelliert. Die Tatsache, dass es sich dabei in diesem Fall um ein gemischtes Inhaltsmodell handelt, wird explizit durch `mixed="true"` vermerkt. Der Häufigkeitsindikator wird in (16) durch die Attribute `minOccurs="0"` und `maxOccurs="unbounded"` ausgedrückt, hier können außerdem beliebige Zahlenwerte eingetragen werden. Innerhalb der `choice`-Konstruktion wird über das Konstrukt `<element ref="B"/>` auf die Deklaration des Elements B verwiesen. Dabei muss es sich innerhalb des Schema-Dokuments um ein global deklariertes Element handeln. In ähnlicher Weise wie Elemente werden auch Attribute deklariert: Das Attribut C wird durch das Element `simpleType` definiert, bei dem es darum geht, einen der vordefinierten Datentypen – hier `NMTOKEN` – zu nutzen oder abzuwandeln. In diesem Fall wird über eine `restriction`-Konstruktion die Menge der möglichen Werte des Typs `NMTOKEN` auf die zwei Werte a und b eingeschränkt.

Während (16) im Wesentlichen die DTD-Syntax in XML nachzeichnet, geht eine alternative Umsetzung über die Möglichkeiten von DTDs hinaus. Dabei wird das Element B nicht als ein globales Element deklariert, sondern als ein lokales. Sein Inhaltsmodell ist dabei nur für den vorliegenden Kontext – hier innerhalb des Elements A – in der angegebenen Weise definiert, in anderen Kontexten kann dagegen für B ein beliebiges anderes Inhaltsmodell definiert werden:

```
...                                                    (17)
<choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="B" type="string">
</choice>
...
```

Die konsequente Verwendung lokaler Element-Deklarationen 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:

```
<element name="A">                                     (18)
    <complexType>
        <choice>
            <element name="B">
                (Inhaltsmodell 1)
            </element>
            <element name="B">
                (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. (2000) als *single-type tree grammar* bezeichnet.

```
<element name="A">                                    (19)
    <choice>
        <ref name="B1"/>
        <ref name="B2"/>
    </choice>
</element>
<define name="B1">
    <element name="B">
        <ref name="C">
    </element>
</define>
<define name="B2">
    <element name="B">
        <ref name="D">
    </element>
</define>
```

Mit der Spezifikation von RELAX NG wurde versucht, die volle Mächtigkeit regulärer Baumgrammatiken für die Auszeichnung von Texten nutzbar zu machen. Das in (18) dargestellte Problem lässt sich in RELAX NG wie in (19) dargestellt lösen: Die Markierung der Elemente durch Tags ist hier – wie generell in regulären Baumgrammatiken – entkoppelt von der Bezeichnung der Kategorien. Die Deklaration für B1 und B2 lassen sich somit auch in die traditionelle Notation umsetzen:

```
A  →  <A>B1</A>                                       (20)
A  →  <A>B2</A>
B1 →  <B>C</B>
B2 →  <B>D</B>
```

Eine weitere Konsequenz aus der kompromisslosen Umsetzung einer regulären Baumgrammatik in das XML-Format besteht darin, in Inhaltsmodellen die Auswahl zwischen Element- und Attribut-Konstruktionen zuzulassen. In der folgenden Grammatik wird das Element A entweder durch mit B markierten beliebigen Text ausgeprägt oder durch die beim Attribut C erscheinenden Werte a oder b:

```
<element name="A">                                    (21)
    <choice>
        <element name="B">
            <text/>
        </element>
        <attribute name="C">
            <choice>
                <value>a</value>
                <value>b</value>
            </choice>
        </attribute>
    </choice>
</element>
```

Trotz der einfacheren Notation bildet RELAX NG gegenüber XML Schema den mächtigeren Typus einer Schemasprache. Dieses wirkt sich auf die definierbaren Kontextabhängigkeiten aus, die sich auch auf Attributwerte erstrecken können, betrifft aber auch die Kombination mehrerer Grammatiken, die nur dann miteinander vereinigt, geschnitten oder voneinander abgezogen werden können, wenn man sich im Bereich voll ausgeprägter regulärer Baumgrammatiken bewegt. Für all dieses weist RELAX NG bestimmte syntaktische Konstruktionen auf, deren Darstellung hier zu weit führen würde.

4 Typenhierarchie, Metastrukturen und Constraints

Neben der Mächtigkeit der verwendeten Grammatikvariante fallen bei der Bewertung von Schemasprachen aber auch andere Eigenschaften ins Gewicht. Zwei davon sollen in diesem Abschnitt besprochen werden: die Idee, Dokumentgrammatiken ähnlich wie Programme objektorientiert aufzubauen, und die Möglichkeit, beliebige Restriktionen von Dokumenten durch Constraints zu definieren.

4.1 Elemente als Objekte

Das Typenkonzept von XML Schema

Obwohl es bei der Spezifikation von Dokumentgrammatiken in SGML und XML immer um die Definition der Zusammenhänge zwischen Informati-

onseinheiten geht, letztlich also eine datenorientierte Perspektive eingenommen wird, wird das dafür besonders geeignete Konzept der Objektorientierung erst in XML Schema systematisch 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 gleichzeitig 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 Instanziierungsbeziehung wie die zwischen Klasse und Objekt, weshalb von Element-Deklarationen, genauer auch von Elementtyp-Deklarationen, gesprochen wird. 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, instanziieren lassen.

Im folgenden Beispiel wird zunächst der Typ `adressTyp` deklariert, danach ein Elementtyp `adresse`:

```
<complexType name="adressTyp">                                  (22)
  <sequence>
    <element name="strasse" type="string"/>
    <element name="hausnummer" type="NMTOKEN"/>
    <element name="postleitzahl" type="integer"/>
    <element name="stadt" type="string"/>
  </sequence>
  <attribute name="datum" type="date"/>
</complexType>

<element name="adresse" type="adressTyp"/>
```

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

```
<element name="adressemitland">                    (23)
  <complexType>
    <complexContent>
      <extension base="adressTyp">
        <sequence>
          <element name="land" type="string"/>
        </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 `adressemitland` die Sequenz der Elemente `strasse`, `hausnummer`, `postleitzahl` und `land` erscheinen. Zusätzlich wird dem Element `adressemitland` aus seinem Basistyp das Attribut `datum` vererbt. Die Deklaration des komplexen Typs geschieht in (23) 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 es natürlich durchaus denkbar, auch interessantere Erweiterungsformen zuzulassen als lediglich das Anhängen weiterer Elemente. 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:

```
<sequence>                                                    (24)
  <element name="personenname" type="nameType"/>
  <complexType ref="adressTyp"/>
</sequence>
```

Sinnvoll könnte eine derartige Verwendungsweise durchaus sein: An alle aus `adressTyp` abgeleiteten Typen könnte die Verwendung vererbt werden, für abgeleitete Typen könnten darüber hinaus spezielle Verwendungsweisen definiert werden.

Weitere Möglichkeiten der objektorientierten Schema-Spezifikation

Als ein wichtiger Vorläufer der Objektorientierung bei XML kann das Konzept der Architektur oder Meta-DTD bei SGML angesehen werden. Dieses Konzept ist einerseits konsequenter, andererseits schwächer als das in XML Schema realisierte. Eine Meta-DTD ist eine gewöhnliche DTD, die durch einige spezielle Deklarationen einer anderen DTD, der Client-DTD, übergeordnet wird. Die Elementtypen der Client-DTD werden über Kontrollattribute Elementtypen der Meta-DTD zugeordnet. Für unser Beispiel könnte das etwa folgendermaßen aussehen:

```
Meta-DTD:  <!ELEMENT adressTyp  (strasseTyp,                  (25)
                                 hausnummerTyp,
                                 postleitzahlTyp,
                                 stadtTyp)>

Client-DTD: <!ELEMENT adressemitland  (strasse,
                                       hausnummer,
                                       postleitzahl,
                                       stadt,
                                       land)>

            <!ATTLIST adressemitland
                      metaAF    (adressTyp)    #FIXED>
```

Da in der Client-DTD auch bei einer Erweiterung das Inhaltsmodell vollständig definiert werden muss, ist dieser Ansatz allgemeiner als das Typenkonzept in XML Schema. Der Nachteil dieses Verfahrens liegt darin, dass

in der Standardisierung von Architekturen versäumt worden ist, neben der Notation der Zuordnung auch Vererbungsprozesse zu berücksichtigen. Die Notwendigkeit, alle Vererbungsvorgänge 'per Hand' in die DTDs einzutragen, und die fehlenden Richtlinien zur Validierung der architektonischen Zuordnung haben dazu beigetragen, dass diese Technik nicht aus ihrem Nischendasein herausfinden konnte.

Andere Überlegungen gehen dahin, die objektorientierten Strukturen einer Dokumentgrammatik gleich in einem Formalismus zu erfassen, der vorrangig für diesen Zweck entwickelt worden ist. In XML Schema werden nach dieser Auffassung syntaktische Aspekte – die Anordnung der Elemente zu Bäumen – und semantische – die inhaltlichen Bezüge der einzelnen Datentypen zueinander – miteinander vermischt. Eine Entflechtung besteht darin, im Schema nur die syntaktischen Aspekte zu behandeln und die semantischen, objektorientierten Aspekte in einer Spezialsprache wie der *Unified Modeling Language* (UML, vgl. Kimber und Heintz, 2000) oder durch Entity-Relationship-Modelle (Mani et al., 2001). UML ist eine grafische Sprache zur Definition der Bezüge zwischen Klassen, sie hat trotz der Ähnlichkeit im Namen nichts mit XML gemein. Wegen der Komplementarität dieser beiden Sprachen wird für die Entwicklung großer Schemata eine Kombination der jeweiligen Modellierungsverfahren empfohlen.

4.2 Constraints und Beispieldokumente

Wir haben in Abschnitt 3.1 gesehen, wie baumartig repräsentierte Information durch Grammatiken beschrieben werden kann. Die dabei verwendeten Schema-Sprachen verfolgen generell den Ansatz, die Menge der möglichen Dokumente vollständig zu beschreiben. Ein grundlegend anderer Ansatz besteht darin, einzelne Zusammenhänge der in dem Baum enthaltenen Daten gezielt zu spezifizieren. Dieses kann alternativ zu einer grammatischen Definition erfolgen oder parallel dazu. Die grammatische Definition ist auch in der voll ausgeprägten Version einer regulären Baumgrammatik im Falle von RELAX NG auf die Spezifikation lokaler Abhängigkeiten von Elementen und Attributen untereinander beschränkt. Die Spezifikation von Abhängigkeiten beliebig weit voneinander entfernt erscheinender Informationseinheiten erfordert dagegen einen Formalismus, der weit über die Mächtigkeit regulärer Baumgrammatiken hinausgeht. Da es unsinnig ist, mögliche Dokumente mit einem solchen Formalismus vollständig beschreiben zu wollen, werden derartige Abhängigkeiten eher als Constraints über den Baumstrukturen interpretiert, deren Einhaltung zu überprüfen ist.

Die ausschließliche Nutzung von Constraints macht es erforderlich, strukturierte Dokumente entweder ganz ohne grammatische Definitionen zu verwenden – dieses ist in XML mit dem Konzept der wohlgeformten Dokumente angelegt, in SGML aber verboten – oder einzelne Inhaltsmodelle in einer Dokumentgrammatik undefiniert zu lassen und so sämtliche Unterstrukturen zuzulassen. Dieses ist sowohl in XML als auch in SGML möglich durch die Variable `ANY`, die als Inhaltsmodell erscheinen kann.

Schematron

Eine ausgereifte Konvention zur Spezifikation von Constraints über XML-Dokumenten ist Schematron (Jelliffe, 2000). Nehmen wir an, ein XML-Dokument enthält die folgende Information:

```
<bestellung>                                                    (26)
  <produkt>Rasierapparat T-310</produkt>
  <preis waehrung="EUR">120,00</preis>
  <adresse>
    <strasse>Goethestraße</strasse>
    <nr>7</nr>
    <plz>35390</plz>
    <stadt>Gießen</stadt>
    <land>Deutschland</land>
  </adresse>
</bestellung>
```

Für dieses Dokument soll der Constraint formuliert werden, dass die Währung `EUR` nur im Zusammenhang mit dem Land `Deutschland` in der Lieferadresse erscheinen darf, die Währungsbezeichnung `SFr` dagegen nur beim Zielland `Schweiz`. Es ist nicht möglich, diesen Zusammenhang in einer der bisher dargestellten Schema-Sprachen auszudrücken.

Zwei Schematron-Regeln, die diesen Constraint implementieren, sehen folgendermaßen aus:

```
<rule context="preis/@waehrung='EUR'">                    (27)
  <assert test="/bestellung/adresse/land='Deutschland'">
  <name/>: Ein Preis in Euro darf nur für eine
  Bestellung in Euro-Ländern ausgewiesen sein.
  </assert>
</rule>

<rule context="preis/@waehrung='SFr'">
  <assert test="/bestellung/adresse/land='Schweiz'">
  <name/>: Ein Preis in Franken darf nur für eine
  Bestellung in die Schweiz ausgewiesen sein.
  </assert>
</rule>
```

Jede Schematron-Regel wird durch das Element `rule` kodiert, das als Wert des Attributs `context` einen XPath-Ausdruck enthält, durch den der Anwendungskontext der Regel spezifiziert wird. In diesem Fall handelt es sich um das Attribut `waehrung` beim Element `preis`, sofern es den Wert `EUR` bzw. `SFr` aufweist.

Die Auswertung derartiger Regeln, die zu einem Schematron-Schema zusammengefasst werden, erfolgt in zwei Schritten. Zunächst wird aus dem Schematron-Schema durch eine für alle derartigen Schemata gültige Transformation ein XSLT-Skript abgeleitet, dieses ist dann auf das zu überprüfende XML-Dokument anzuwenden. Für beide Schritte kann ein gewöhnlicher XSLT-Prozessor verwendet werden. Die Ausgabe des zweiten Transformationsprozesses ist ein *Validation Report*, in dem insbesondere Fehlermeldungen für diejenigen Constraints enthalten sind, deren Nicht-Einhaltung bei der Validierung festgestellt worden ist. Die Fehlermeldungen werden nicht schematisch generiert, sondern unterliegen ebenfalls der Ausformung durch den Entwickler des Schematron-Schemas als Inhalt des Elements `assert`. Für die erste Regel in (27) würde etwa die folgende Fehlermeldung generiert werden:

```
preis/waehrung='EUR': Ein Preis in Euro darf nur für      (28)
eine Bestellung in Euro-Ländern ausgewiesen sein.
```

Eine Reihe weiterer Elemente erlauben es, Schematron-Schemata zu entwickeln, durch die sehr komplexe Überprüfungen von Dokumenten vor-

genommen werden können in Abhängigkeit vom gegenwärtigen Stand des Workflows.

Aus einer linguistischen Perspektive können derartige Constraint-Systeme verstanden werden als eine Ebene oberhalb der hierarchischen Dokumentstruktur, durch die „Long Distance Dependencies" ausgedrückt und 'lexikalische' Einflüsse auf die Baumstruktur (analog z. B. zum Konzept der Verbvalenz in der Sprache) beschrieben werden können.

Wegen der orthogonalen Konzeption von Schematron und XML Schema hinsichtlich der Spezifikation von Dokumentstrukturen ist es naheliegend, beide Ansätze miteinander zu verbinden. Dieses ist auch recht einfach möglich, da XML Schema bei jeder Element-Deklaration in einem Element `annotation` Meta-Informationen zu kodieren erlaubt. Die Schematron-Regeln, die für das definierte Element gelten sollen, können im Element `appinfo` aufgeführt werden, vgl. (29). Aus dieser integrierten Repräsentation beider Schematypen lässt sich durch einen vorgelagerten Transformationsschritt das Schematron-Schema extrahieren.

```
<xs:element name="preis">                                    (29)
  <xs:annotation>
    <xs:appinfo>
      <sch:rule context="preis/@waehrung='EUR'">
        <sch:assert
        test="/bestellung/adresse/land='Deutschland'">
        <sch:name/> : Ein Preis in Euro darf nur für
        eine Bestellung in Euro-Ländern ausgewiesen sein.
        </sch:assert>
      </sch:rule>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

Beispielbasierte Dokumentspezifikation

Ein anderer Ansatz, Mengen von Dokumenten durch Constraints zu beschreiben, besteht in der Verwendung von Beispieldokumenten. Bereits vor der Definition von XML stand mit Fred für SGML ein Tool zur Verfügung, das in der Lage war, aus einem Beispieldokument eine DTD abzuleiten.

Fred geht dabei in zwei Schritten vor. Zunächst werden die Strukturen des Dokuments Ebene für Ebene in Element-Deklarationen überführt. Werden für einen Elementtyp an verschiedenen Stellen des Dokuments unterschiedliche Inhaltsmodelle registriert, so werden diese zu einer Oder-Gruppe zusammengefasst. Im zweiten Schritt wird eine Reihe von Regeln angewandt, die die Inhaltsmodelle vereinfachen, Ambiguitäten beseitigen und Verallgemeinerungen vornehmen. Diese Regeln bewirken beispielsweise die folgenden Umformungen (siehe Lindén, 1997):

$$\begin{aligned} (a, a, a, a) &\rightarrow (a+) \\ (a|(b, a)) &\rightarrow (b?, a) \\ (a + |b)^* &\rightarrow (a|b)^* \end{aligned} \tag{30}$$

Die automatische Deduktion von Dokumentgrammatiken ist allerdings mit prinzipiellen Schwierigkeiten konfrontiert. Werden die Verallgemeinerungsregeln sehr eng ausgelegt, ist die DTD vor allem auf das Dokument beschränkt, aus dem sie abgeleitet worden ist. Dieses kann zwar durchaus sinnvoll sein, wenn abzusehen ist, dass die Dokumentstrukturen konstant bleiben und vor allem die Inhalte variieren, entspricht aber nicht der üblichen Verwendungsweise strukturierter Dokumente. Werden die Regeln so ausgelegt, dass automatisch deutliche Verallgemeinerungen vorgenommen werden, kann die DTD aufgrund ihrer zu großen strukturellen Liberalität ihren Zweck nicht mehr erfüllen.

Inzwischen verfügen Entwicklungssysteme für Dokumentgrammatiken wie XMLSpy (vgl. `http://www.xmlspy.com`) über Grammatik-Deduktion-Funktionen, die nicht nur die Ausgabe der Grammatik in verschiedenen Schema-Sprachen erlauben, sondern auch den 'Stil' der zu entwickelnden Grammatik zu parametrisieren. So kann bei der Deduktion von Grammatiken in XML Schema festgelegt werden, ob Elemente grundsätzlich global oder lokal, als benannte oder als anonyme komplexe Typen deklariert werden sollen. Eine derartige Funktionalität ist insbesondere im Umfeld eines Grammatik-Entwicklungswerkzeugs sinnvoll, da als Grundlage für die Erstellung einer komplexen Grammatik eine automatisch generierte Rohversion stehen kann. Gleichzeitig machen die Deduktionswerkzeuge deutlich, dass die Erstellung von Dokumentgrammatiken ein interpretatorischer Vorgang ist, der über die Anwendung formaler Regeln hinausgeht, da das konzeptuelle System des Gegenstandsbereichs nur zu einem geringen Teil in die Dokumentgrammatik einfließt. Die Abbildung eines solchen konzeptuellen Systems in eine Dokumentgrammatik stellt einen intellektuellen Vorgang

dar, bei dem Techniken aus dem Bereich des Software Engineering Anwendung finden (vgl. Maler und Andaloussi, 1996).

Eine konsequente Weiterentwicklung des Deduktionsansatzes ist mit Examplotron vorgestellt worden (vgl. Examplotron, 2001). Examplotron ist ein XSLT-Skript, das aus einem Beispieldokument einen Schematron-Validator ableitet. Dabei akzeptiert Examplotron einige spezielle Attribute und Elemente im Beispieldokument, die Auskunft darüber geben, welchen allgemeinen Status ein Element erhalten soll. Das Attribut `occurs` etwa erlaubt die Angabe eines Auftretensoperators wie in DTD-Inhaltsmodellen. Das Attribut `assert` stellt durch einen XPath-Ausdruck Bezüge zu anderen Teilen des Dokuments her, ähnlich dem entsprechenden Element in Schematron. Über ein Element `import` können andere Examplotron-Beispieldokumente eingebunden werden, so dass ein mehrere Beispieldokumente übergreifendes Schema abgeleitet werden kann.

Examplotron weist selbstverständlich nicht die volle Mächtigkeit der regulären Schema-Sprachen auf, stellt aber eine interessante, vor allem sehr leicht zu erlernende Alternative dar für die schnelle Spezifikation von Dokumentgrammatiken.

5 Verwendung von Dokumentgrammatiken

Die Verwendung von Dokumentgrammatiken und Constraints erstreckt sich auf alle Bereiche, in denen Vorteile aus der Vorhersagbarkeit der im Dokument vorgefundenen Strukturen gezogen werden können. Elementar ist dabei in jedem Fall die Validierung eines Dokuments in Bezug auf eine gegebene Dokumentgrammatik.

5.1 Validierung und Parsing

Bei der Validierung eines Dokuments wird überprüft, ob die Sequenz der einem Element direkt untergeordneten Tochterelemente durch das Inhaltsmodell der Element-Deklaration lizensiert ist. Sehen wir uns erneut das Beispiel einer Produktbestellung an:

```
<bestellung>                                                    (31)
   <produkt>Rasierapparat T-310</produkt>
   <preis waehrung="EUR">120,00</preis>
   <adresse>
      <strasse>Goethestraße</strasse>
      <nr>7</nr>
      <plz>35390</plz>
      <stadt>Gießen</stadt>
      <land>Deutschland</land>
   </adresse>
</bestellung>
```

Das DTD-Fragment dazu sieht folgendermaßen aus:

```
                                                                (32)
<!ELEMENT bestellung (produkt, preis, adresse, datum?)>
<!ELEMENT produkt    (#PCDATA)>
<!ELEMENT preis      (#PCDATA)>
<!ATTLIST preis      waehrung CDATA #REQUIRED>
...
```

Im Falle von DTDs ist der Validierungsalgorithmus denkbar einfach. Wird im Dokument ein Starttag gefunden, ist die entsprechende Element-Deklaration in der DTD zu aktivieren. Die bei diesem Element im Dokument aufgefundenen Tochterelemente müssen einer der im Inhaltsmodell definierten möglichen Elementreihenfolgen entsprechen. So wird in (31) zwischen dem Starttag `<bestellung>` und dem Endtag dieses Elements (`</bestellung>`) die Sequenz der Elemente `produkt`, `preis` und `adresse` vorgefunden; diese ist in der DTD (32) mit dem Inhaltsmodell (`produkt`, `preis`, `adresse`) zu matchen. Beim Matching-Prozess muss berücksichtigt werden, dass durch Iterationsoperatoren und Oder-Konstruktionen potentiell unendlich viele Sequenzen durch ein Inhaltsmodell spezifiziert sein können. Der Abgleich zwischen der tatsächlichen Sequenz und dem Inhaltsmodell entspricht formal der Anwendung eines regulären Ausdrucks auf eine Eingabe-Sequenz von Wörtern, wofür es effiziente Verfahren gibt.

Bei allen Tochterelementen ist rekursiv nach dem gleichen Schema zu verfahren, sofern nicht ein leeres Element vorgefunden wird. Dateninhalt und Attribute sind entsprechend mit der DTD abzugleichen. Murata et al.

(2000) zeigen, wie dieser einfache Algorithmus schrittweise zur Behandlung mächtigerer Schema-Sprachen erweitert werden kann.

In Dokumentgrammatiken können eine Reihe von Konstruktionen in Inhaltsmodellen erscheinen, bei denen während des Validierungsprozesses nicht entschieden werden kann, welche Variante zu wählen ist. So ist z. B. in

```
<!ELEMENT A (B*, ((B, C)|D))>                    (33)
```

nicht zu entscheiden, ob bei einem Element B als Tochterelement von A im Inhaltsmodell noch das erste B gemeint ist oder bereits das zweite. Derartige Ambiguitäten sind sowohl in XML-DTDs als auch in XML Schema explizit verboten, so dass der Grammatik-Entwickler deren Vermeidung selbst zu beachten hat. Zwar ist die Vermeidung von ambigen Inhaltsmodellen ein Merkmal von Grammatiken, das die Entwicklung von Validierungsprogrammen einfacher und deren Laufzeitverhalten effizienter macht, doch können auch ambige Inhaltsmodelle zur Validierung herangezogen werden, sofern das Validierungsprogramm über Backtracking- oder Chart-Mechanismen verfügt, die falsche Strukturzuordnungen nachträglich zu korrigieren erlauben.

Geht es bei der Validierung lediglich um eine Ja-Nein-Entscheidung, ob das Dokument der Grammatik strukturell folgt, wird beim Parsen eines Dokuments eine Ausgabe produziert, die Informationen enthalten kann, die nicht Teil des Ausgangsdokuments sind. In DTDs betrifft dieses z. B. Default-Werte von Attributen. Ist das Attribut `waehrung` im Beispiel (32) etwa mit einem Default-Wert deklariert

```
<!ATTLIST preis waehrung CDATA "EUR">            (34)
```

so kann im Dokument auf die Angabe des Attributs verzichtet werden:

```
...                                              (35)
<preis>120,00</preis>
...
```

Der Parsing-Prozess ergänzt den Default-Wert in der Ausgaberepräsentation:

```
...                                              (36)
<preis waehrung="EUR">120,00</preis>
...
```

Die Information, die in einem Dokument tatsächlich enthalten ist, wird auch als *Infoset* bezeichnet. Bei der Validierung wird das Infoset des Dokuments nicht verändert, beim Parsing hingegen vergrößert. In XML Schema werden u. a. auch die Informationen zum Typ von Daten und Elementen in das *Post Schema Validation Infoset* (PSVI) übertragen, so dass im PSVI wesentlich mehr Information der weiteren Verarbeitung zur Verfügung steht als im Ausgangsdokument. Beim PSVI handelt es sich um eine abstrakte Datenstruktur, die auf unterschiedliche Weise applikationsspezifisch konkretisiert werden kann.

5.2 Weitere Verwendungsweisen von Dokumentgrammatiken

XML-Editoren bedienen sich ebenfalls der Dokumentgrammatik, um den Produktionsprozess von XML-Dokumenten zu unterstützen. Anders als bei der Validierung wird die Grammatik eingesetzt, um für ein Dokument eine Liste derjenigen Elemente zu ermitteln, die an einer bestimmten Position eingefügt werden können. Durch eine Auswahl aus dieser Liste kann der Benutzer die Struktur des Dokuments interaktiv aufbauen; obligatorische Strukturteile werden in vielen Editoren dabei automatisch eingefügt.

```
<bestellung>                                          (37)
   <produkt>Rasierapparat T-310</produkt>
   <adresse>
      <strasse>Goethestraße</strasse>
      <nr>7</nr>
      <plz>35390</plz>
      <stadt>Gießen</stadt>
      <land>Deutschland</land>
   </adresse>
</bestellung>
```

Einer solchen Funktionalität liegt ein Prozess zugrunde, der ähnlich der Validierung arbeitet, dabei jedoch liberaler verfährt, wenn das Fehlen von Elementen registriert wird. Befindet sich der Cursor beispielsweise an der wie in Beispiel 37 dargestellten Position im Dokument, so wird kein Fehler ausgegeben, sondern gemäß (32) das Element `preis` zur Einfügung angeboten. Nur wenn allein durch Einfügung weiterer Elemente das Dokument nicht mehr DTD-konform werden kann, wird auch während des Editierprozesses ein Fehler ausgegeben. Dieses wäre etwa der Fall, wenn in (37) die Elemente `produkt` und `adresse` in umgekehrter Reihenfolge erscheinen würden.

Eine systematische Einbeziehung von Dokumentgrammatiken bei der Transformation von XML-Dokumenten wird bislang nicht vorgenommen. Murata (1998) hat gezeigt, dass nur bei der Verwendung regulärer Baumgrammatiken gewährleistet werden kann, dass nach einer Transformation in eine andere Dokumentstruktur die Validierbarkeit in Bezug auf eine andere Dokumentgrammatik gegeben ist. Insofern haftet der Dokumenttransformation immer etwas Provisorisches an, sofern sie sich auf der Ebene der weniger mächtigen Varianten regulärer Baumsprachen bewegt. Wird eine Transformation z. B. in XSLT spezifiziert, so hat der Entwickler des Transformationsskripts durch den Aufbau der Regeln sicherzustellen, ob Anwendungskontexte vorkommen können, in denen durch die Ziel-Dokumentgrammatik nicht abgedeckte Strukturen entstehen. Ein allgemeines formales, also automatisierbares Lösungsverfahren existiert für dieses Problem nicht.

In XPath 2.0, das zur Zeit noch nicht als Empfehlung vorliegt, wird aller Voraussicht nach das *Post Schema Validation Infoset* berücksichtigt werden. Dieses bedeutet, dass zumindest bei der Adressierung auch solche Informationen herangezogen werden können, die nicht im Dokument selbst enthalten sind, sondern erst bei der Validierung aus dem XML-Schema in das PSVI einfließen. Dieses betrifft vor allem Datentypen und Standardwerte.

Ein Standardbeispiel ist die Suche in einem strukturierten Dokument: Wenn der Dokumentgrammatik entnommen werden kann, dass z. B. ein Element `ueberschrift` nur als erstes Tochterelement eines Elements `kapitel` erscheinen kann, kann die Geschwindigkeit der Suche nach Elementen dieses Typs durch Auslassung großer Teile des Dokuments um ein Vielfaches erhöht werden. Bislang ist von dieser Möglichkeit beim Entwurf von Querying-Tools für XML-Dokumente nicht Gebrauch gemacht worden. Zusammen mit dem Entwurf für XPath 2.0 ist für XQuery, die in der Entwicklung befindliche Abfragesprache für XML-Dokumente, ein Datenmodell entwickelt worden (vgl. XQuery, 2002), das auch Informationen aus dem PSVI umfasst.

Eine gänzlich andere Verwendungsart von Dokumentgrammatiken ist für die Zwecke der automatischen Textkategorisierung zu beobachten (vgl. Rehm, 2002). Dabei geht es nicht darum, Dokumentstrukturen durch eine Grammatik zu definieren, sondern Strukturwissen über Dokumente zu repräsentieren. Für Dokumente, die kategorisiert werden sollen, ist diejenige Dokumentgrammatik zu finden, deren Strukturierung am besten auf dieses angewandt werden kann. Da nicht mit der Verwendung gleicher Elementnamen oder auch nur der gleichen Elementhierarchie zu rechnen ist, werden

die Dokumentgrammatiken mit Erkennungsroutinen kombiniert, mit deren Hilfe der entsprechende Strukturteil im Dokument identifiziert werden kann.

6 Ausblick

Wir haben gesehen, dass Dokumentgrammatiken schon heute eine wichtige Rolle spielen, wenn es darum geht, strukturierte Dokumente effizient und mit mächtigen Werkzeugen zu verarbeiten. In welche Richtung werden sich Dokumentgrammatiken in der näheren Zukunft entwickeln? Einige Entwicklungsperspektiven zeichnen sich bereits heute ab.

Immer häufiger werden Klassen von Dokumenten nicht nur mit Dokumentgrammatiken kombiniert, sondern auch mit einer formalisierten Beschreibung der Bedeutung der Struktur. Wird in einer Dokumentgrammatik z. B. für eine Tabelle lediglich festgelegt, dass sie aus Zeilen besteht und diese wiederum in eine bestimmte Anzahl von Zellen unterteilt sind, so beschreibt ein semantisches Modell die gegenseitige Abhängigkeit von Zeilen und Spalten, Zeilen- und Spaltenköpfen, Datenzellen usw. – oftmals Konzepte, die im Dokument und in der Dokumentgrammatik nicht explizit erscheinen. Wie bei der Analyse natürlicher Sprache, bei der zwischen Syntax und Semantik unterschieden wird, so spiegelt auch eine Dokumentgrammatik lediglich die syntaktische Seite des Dokuments wider, die durch eine semantische Repräsentation ergänzt werden kann. Derartige semantische Repräsentationen werden Ontologien genannt und ihrerseits als strukturierte Dokumente auf der Grundlage einiger inzwischen weit verbreiteter Standards dargestellt. Ein Basisformalismus für Wissensrepräsentationszwecke in XML steht mit dem *Resource Description Framework* (RDF) zur Verfügung, *XML Topic Maps* (XTM) erlauben die Spezifikation semantischer Netze, und mit dem *Ontology Interface Layer* (vgl. Horrocks et al., 2000) ist inzwischen auch eine XML-basierte Repräsentation für Ontologien verfügbar, die eine Inferenzebene umfasst.

Ausgereifte Ontologien für bestimmte Domänen können als eine integrierende Ebene oberhalb der Dokumentgrammatiken verstanden werden. Verschiedene Dokumentgrammatiken setzen das in der Ontologie repräsentierte Strukturwissen in jeweils unterschiedlicher Weise syntaktisch um. Ontologien bilden deshalb ein Basis, um zukünftig auch die Erstellung von Transformationsskripten zwischen XML-Dokumenten zu automatisieren. Auch bei den Dokumentgrammatiken selbst sind weitere Entwicklungsschritte zu erwarten. Das Verständnis der formalen Grundlagen von Schema-

sprachen hat bereits zu Entwicklungen wie RELAX NG geführt. Eine Integration von regelbasierten und Constraint-basierten Spezifikationstechniken steht bei der Weiterentwicklung von XML Schema zu erwarten.

Wichtige Erkenntnisse vermag auch die Übertragung von linguistischen Methoden und Konzepten auf Schemasprachen und Dokumentgrammatiken zu bewirken. Die linguistische Grammatiktheorie befasst sich seit einem halben Jahrhundert mit Methoden der formalen Spezifikation komplexer Grammatiken; in diesem Zusammenhang ist ein reichhaltiges Inventar von Konzepten geschaffen worden, deren Anwendung auch im nichtsprachlichen Kontext sinnvoll sein kann. Neben konzeptionellen können auch methodische Erkenntnisse aus der Linguistik übertragen werden. Gerade in den letzten Jahren haben verstärkt Forschungen dazu eingesetzt, wie ein Textkorpus automatisch mit grammatischer Information annotiert und wie aus annotierten Korpora grammatisches Wissen deduziert werden kann (siehe z. B. den Beitrag von Ule und Hinrichs, in diesem Band). Die wissenschaftliche Durchdringung dieser Zusammenhänge im Bereich beliebiger XML-annotierter Daten steht hingegen noch aus.

Literaturverzeichnis

Cagle, Kurt; Duckett, Jon; Griffin, Oliver; Mohr, Stephen; Norton, Francis; Ozu, Nikola; Stokes-Rees, Ian; Tennison, Jeni und Williams, Kevin (2001): *Professional XML Schemas.* Birmingham: Wrox Press.

Carstensen, Kai-Uwe; Ebert, Christian; Endriss, Cornelia; Jekat, Susanne; Klabunde, Ralf und Langer, Hagen (Herausgeber) (2001): *Computerlinguistik und Sprachtechnologie. Eine Einführung.* Heidelberg, Berlin: Spektrum akademischer Verlag.

Examplotron (2001): "Examplotron". `http://examplotron.org/`.

Horrocks, Ian; Fensel, Dieter; Broekstra, Jeen; Decker, Stefan; Erdmann, Michael; Goble, Carole; van Harmelen, Frank; Klein, Michael; Staab, Steffen; Studer, Rudi und Motta, Enrico (2000): "The Ontology Inference Layer OIL". `http://www.ontoknowledge.org/oil/TR/oil.long.html`.

HTML (1999): "HTML 4.01 Specification". World Wide Web Consortium (W3C), `http://www.w3.org/TR/html4/`.

Jelliffe, Rick (2000): "The Schematron – An XML Structure Validation Language using Patterns in Trees". `http://www.ascc.net/xml/resource/schematron`.

Kimber, Eliot und Heintz, John (2000): "Using UML to define XML document types". *Markup Languages* 2 (3): S. 295–320.

Lee, Dongwon und Chu, Wesley W. (2000): "Comparative Analysis of Six XML Schema Languages". http://www.cs.ucla.edu/~dongwon/paper.

Lindén, Greger (1997): *Structured Document Transformations.* Dissertation, Department of Computer Science, University of Helsinki, Finland. Report A-1997-2, http://www.cs.helsinki.fi/TR/A-1997/2/.

Lobin, Henning (2000): *Informationsmodellierung in XML und SGML.* Berlin, Heidelberg: Springer-Verlag, 2. Auflage.

Maler, Eve und Andaloussi, Jeanne El (1996): *Developing SGML DTDs. From Text to Model to Markup.* Upper Saddle River, New Jersey: Prentice Hall.

Mani, Murali; Lee, Dongwon und Muntz, Richard R. (2001): "Semantic Data Modeling using XML Schemas". In: *Proc. 20th International Conference on Conceptual Modeling (ER).* Yokohama, Japan. http://www.cobase.cs.ucla.edu/tech-docs/dongwon/er2001.ps.

Murata, Makoto (1998): "Data Model for Document Transformation and Assembly". In: *Proc. of PODDP 1998.*

Murata, Makoto; Lee, Dongwon und Mani, Murali (2000): "Taxonomy of XML Schema Languages using Formal Language Theory". In: *Proc. of Extreme Markup Languages 2000.*

RDF (1999): "Resource Description Framework (RDF). Model and Syntax Specification". World Wide Web Consortium (W3C), http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

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).* Big Island (Hawaii).

RELAX NG (2001): "RELAX NG Specification. Committee Specification". OASIS – Organization for the Advancement of Structured Information Standards. James Clark und Makoto Murata (Hrsg.), 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).* International Organization for Standardization, Genf.

Shafer, Keith E. (1995): "Creating DTDs via the GB-Engine and Fred". http://www.oclc.org/fred/docs/sgml95.html.

Sperberg-McQueen, Michael; Huitfeld, Claus und Renear, Allen (2000): "Meaning and interpretation of markup". *Markup Languages* 2 (3): S. 215–234.

XML (2000): "Extensible Markup Language (XML) Version 1.0 (Second Edition)". World Wide Web Consortium (W3C), http://www.w3c.org/TR/2000/REC-xml-20001006.

XML Schema (2001): "XML Schema". World Wide Web Consortium (W3C), http://www.w3c.org/XML/Schema.

XPath (1999): "XPath Language Version 1.0". World Wide Web Consortium (W3C), http://www.w3c.org/TR/xpath.

XQuery (2002): "XQuery 1.0: An XML Query Language". World Wide Web Consortium (W3C), http://www.w3.org/TR/xquery/.

XSLT (1999): "XSL Transformations Version 1.0". World Wide Web Consortium (W3C), http://www.w3c.org/TR/xslt.

XTM (2001): "XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification". http://www.topicmaps.org/xtm/1.0/.