



# Erweiterte Dokumentgrammatiken als Grundlage innovativer XML-Tools

Extended Document Grammars as a Basis for Innovative XML Tools

Henning Lobin, Universität Gießen

**Zusammenfassung** Wohlgeformte XML-Dokumente lassen sich als Bäume interpretieren und diese wiederum durch Grammatiken beschreiben. Dokumentgrammatiken weisen einige Besonderheiten auf, die sie von Grammatiken für natürliche Sprachen oder Programmiersprachen unterscheidet. Dieser Beitrag erläutert die Verarbeitungsmöglichkeiten, die aus der Nutzung von formalen Dokumentgrammatiken erwachsen.

▶▶▶ **Summary** Well-formed XML documents can be interpreted as trees, and trees can be described by grammars. Document grammars have some special features that distinguishes them from grammars for natural or programming languages. This paper shows some of the possibilities that grow out from the use of formal document grammars for processing purposes.

**KEYWORDS** XML, Dokumentgrammatik, Baumgrammatik, Schemasprachen

## 1 Einleitung

Eine der grundlegenden Neuerungen im Bereich des Dokument-  
Markups, 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 [5]. 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 Doku-

menten können wegen der Universalität von baumartig strukturierter Information in der Informationstechnologie viele Verfahren aus anderen Bereichen der Informatik übernommen werden. Bei der Überprüfung, ob ein Dokument eine korrekt aufgebaute Baumstruktur aufweist, können beispielsweise Verfahren angewendet werden, die für die Analyse von Programmen entwickelt wurden (Compilerbau).

Aus Sicht der Linguistik ist jedoch der wichtigste Vorteil der Verwendung von Baumstrukturen, dass man diese durch *Grammatiken*<sup>1</sup> beschreiben kann, und Grammatiktheorie und Grammatikformalismen in der Linguistik als ein gut untersuchtes Gebiet gelten kann.

<sup>1</sup> Ist lediglich die Liste der Elemente eines Dokumenttyps gemeint, so spricht man auch von „Tagsets“ oder „Vokabularien“.

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, 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 grammatikalischen Regeln bilden also – ausgehend von kleinen Bedeutungseinheiten (das sog. Kompositionalitätsprinzip sprachlicher Bedeutung) – einen Rahmen zur Rekonstruktion komplexer Bedeutung.

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 ihren jeweiligen Verwendungszwecken 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. 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.

## 2 Baumgrammatiken und Schemasprachen

### 2.1 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, eine Regelmenge zu finden, die 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. [2]).

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:

$$A \rightarrow a b \quad (1)$$

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

$$A \rightarrow A B \quad (2)$$

$$A \rightarrow C$$

$$B \rightarrow B B$$

$$B \rightarrow C$$

$$B \rightarrow x$$

$$C \rightarrow \emptyset$$

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

$$A \quad (3)$$

$$AB$$

$$CBB$$

$$\emptyset xx$$

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

$$A \rightarrow \langle A \rangle a b \langle /A \rangle \quad (4)$$

Gewöhnlich wird A als Elementtyp bezeichnet,  $\langle A \rangle$  als Anfangstag,  $\langle /A \rangle$  als Endtag und „a b“ als Inhaltsmodell. Werden auf der rechten Regel-seite 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:

$$1. A \rightarrow \langle A \rangle A B \langle /A \rangle \quad (5)$$

$$2. A \rightarrow \langle A \rangle C \langle /A \rangle$$

$$3. B \rightarrow \langle B \rangle B B \langle /B \rangle$$

$$4. B \rightarrow \langle B \rangle C \langle /B \rangle$$

$$5. B \rightarrow \langle B \rangle x \langle /B \rangle$$

$$6. C \rightarrow \langle C \rangle \emptyset \langle /C \rangle$$

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

$$6'. C \rightarrow \langle C \rangle \quad (6)$$

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

$$\langle A \rangle A B \langle /A \rangle$$

$$\langle A \rangle \langle A \rangle C \langle /A \rangle B \langle /A \rangle$$

(nach Regel 2.) (7)

$$\langle A \rangle \langle A \rangle C \langle /A \rangle \langle B \rangle B B \langle /B \rangle \langle /A \rangle$$

(nach Regel 3.)

$$\langle A \rangle \langle A \rangle \langle C \rangle \langle /A \rangle \langle B \rangle B B \langle /B \rangle \langle /A \rangle$$

(nach Regel 6'.)

$$\langle A \rangle \langle A \rangle \langle C \rangle \langle /A \rangle \langle B \rangle \langle B \rangle x \langle /B \rangle B \langle /B \rangle \langle /A \rangle$$

(nach Regel 5.)

$$\langle A \rangle \langle A \rangle \langle C \rangle \langle /A \rangle \langle B \rangle \langle B \rangle x \langle /B \rangle$$

(nach Regel 5.)

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 –



folgendermaßen umzuformulieren:

1. `<!ELEMENT A (#PCDATA|B)*>`
- 2.–4. wie in (12) (14)

Damit werden zwar die gleichen Konstruktionen wie in (12) 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 mächtiger Alternativen sehr weit verbreitet ist.

## 2.3 Schemasprachen

**2.3.1 Idee** Das Aufkommen von Schemasprachen seit Mitte der neunziger Jahre mit dem Ziel der Ersetzung von DTDs ist auf drei Anforderungen zurückzuführen, die DTDs nicht erfüllen können:

- **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.
- **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.
- **Mächtigerer Strukturdefinitionen:** Neuere Arbeiten zur Äquivalenz von XML und regulären Baumgrammatiken z. B. [7] 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 z. B. [4], liegt mit XML Schema seit 2001 eine W3C-Empfehlung vor, die mittlerweile von vielen Software-Systemen alternativ zu DTDs unterstützt werden. Eine radikalere Variante stellt RELAX NG dar, die im Folgenden als zweite Schemasprache dargestellt werden soll.

**2.3.2 XML Schema** Die Deklaration des Elements A nach Regel (14) kann in XML Schema folgendermaßen aussehen (vgl. z. B. [1])

```
<element name="A"> (15)
  <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 (15) durch die beiden 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 Dekla-

ration des Elements B verwiesen. Dabei muss es sich innerhalb des Schema-Dokuments um ein global deklariertes Element handeln.

Attribute werden in ähnlicher Weise wie Elemente 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 (15) 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 lokal. 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:

```
... (16)
<choice minOccurs="0"
  maxOccurs="unbounded">
  <element name="B" type="string">
</choice>
...
```

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 Dokumentenmuster, 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"> (17)
  <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 in [7] als *single-type tree grammar* bezeichnet.

**2.3.3 RELAX NG** Mit der Spezifikation von RELAX NG wurde versucht, die volle Mächtigkeit regulärer Baumgrammatiken für die Auszeichnung von Texten nutzbar zu machen. Das in (17) dargestellte Problem lässt sich in RELAX NG folgendermaßen lösen:

```
<element name="A"> (18)
  <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>
```

Die Markierung der Elemente durch Tags ist hier – wie generell in regulären Baumgrammatiken – entkoppelt von der Bezeichnung der Kategorien. Die Deklarationen für B1 und B2 lassen sich somit auch in die traditionelle Notation umsetzen:

```
A → <A>B1</A> (19)
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"> (20)
  <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.

### 3 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.

#### 3.1 Validierung und Parsing

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

```
<bestellung> (21)
  <produkt>Rasierapparat
    T-310</produkt>
  <preis waehrung="EUR">120,00</preis>
  <adresse>
    <strasse>Goethestraße</strasse>
    <nr>7</nr>
    <plz>35390</plz>
    <stadt>Gießen</stadt>
  </adresse>
</bestellung>
```

Das DTD-Fragment dazu sieht folgendermaßen aus:

```
<!ELEMENT bestellung (produkt, (22)
  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 Elementdeklaration 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 (21) zwischen dem Starttag <bestellung> und dem Endtag dieses Elements (</bestellung>)

die Sequenz der Elemente `produkt`, `preis` und `adresse` vorgefunden; diese ist in der DTD (22) mit dem Inhaltsmodell (`produkt`, `preis`, `adresse`) zu „matchen“. Beim Matching-Prozess muss berücksichtigt werden, dass durch Iterationsoperatoren und Oder-Konstruktionen potenziell 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. Wie dieser einfache Algorithmus schrittweise zur Behandlung mächtigerer Schema-Sprachen erweitert werden kann zeigt [7].

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))> (23)
```

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, sodass 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 (22) etwa mit einem Default-Wert deklariert

```
<!ATTLIST preis (24)
    waehrung CDATA "EUR">
```

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

```
... (25)
<preis>120,00</preis>
...
```

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

```
... (26)
<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.

### 3.2 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.

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 folgenden Position im Dokument, so wird kein Fehler ausgegeben, sondern gemäß (22) das Element `preis` zur Einfügung angeboten:

```
<bestellung> (27)
  <produkt>Rasierapparat T-310
  </produkt>|
  <adresse>
    <strasse>Goethestraße</strasse>
    <nr>7</nr>
    <plz>35390</plz>
    <stadt>Gießen</stadt>
  </adresse>
</bestellung>
```

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 (27) 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. Nur bei der Verwendung regulärer Baumgrammatiken kann gewährleistet werden, dass nach einer Transformation in eine andere Dokumentstruktur die Validierbarkeit in Bezug auf eine andere Dokumentgrammatik gegeben ist [6]. 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 überschreibt nur als erstes Tochterelement eines Elements *kapitel* möglich ist, kann die Geschwindigkeit der Suche nach Elementen dieses Typs durch Auslassung großer Teile des Dokuments um ein Vielfaches erhöht werden. Bislang wurde von dieser Möglichkeit beim Entwurf von Querying-Tools für XML-Dokumente nicht Gebrauch gemacht. Zusammen mit dem Entwurf für XPath 2.0 ist für XQuery, der in der Entwicklung befindlichen Abfragesprache für XML-Dokumente, ein Datenmodell entwickelt worden [18], das auch Informationen aus dem PSVI umfasst.

Eine gänzlich andere Verwendungsart von Dokumentgrammatiken ist für die Zwecke der automatischen Textkategorisierung zu beobachten [8]. Dabei geht es nicht darum, Dokumentstrukturen durch

eine Grammatik zu definieren, sondern Strukturwissen über Dokumente zu repräsentieren. Für ein Dokument, das kategorisiert werden soll, 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.

#### 4 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 aber 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 ge-

nannt 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* [3] 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 eine 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 Schemasprachen hat bereits zu Entwicklungen wie RELAX NG geführt. Eine Integration von regel- und constraint-basierten Spezifikationstechniken steht bei der Weiterentwicklung von XML Schema zu erwarten.

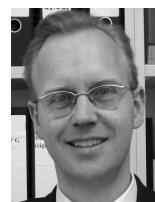
Wichtige Erkenntnisse vermag auch die Übertragung von linguistischen Methoden und Konzepten auf Schemasprachen und Dokumentgrammatiken zu bewirken. Die linguistische Grammatiktheorie befasst sich seit einem halben Jahrhundert mit Methoden der formalen Spezifikation komplexer Grammatiken; in diesem Zusammenhang 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 darüber eingesetzt, wie ein Textkorpus automatisch mit grammatischer Information annotiert und wie aus annotierten Korpora grammatisches Wissen deduziert werden kann. Die wissenschaftliche Durchdringung dieser Zusammenhänge im Bereich beliebiger XML-annotierter Daten steht hingegen noch aus.

### Literatur

- [1] K. Cagle, J. Duckett, O. Griffin, S. Mohr, F. Norton, N. Ozu, I. Stokes-Rees, J. Tennison, K. Williams (2001): Professional XML Schemas. Birmingham: Wrox Press.
- [2] K.-U. Carstensen, C. Ebert, C. Endriss, S. Jekat, R. Klabunde, H. Langer (2001): Computerlinguistik und Sprachtechnologie. Eine Einführung. Heidelberg, Berlin: Spektrum akademischer Verlag.
- [3] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, E. Motta (2000): The Ontology Inference Layer OIL. [<http://www.cs.vu.nl/~dieter/oil/Tr/oil.pdf>]
- [4] D. Lee, W.W. Chu (2000): Comparative Analysis of Six XML Schema Languages. [<http://www.cs.ucla.edu/~dongwon/paper>]
- [5] H. Lobin (2000): Informationsmodellierung in XML und SGML. Berlin, Heidelberg: Springer-Verlag [2. Aufl. 2001].
- [6] M. Murata: „Data Model for Document Transformation and Assembly“. In Proc. of PODDP 1998. [<http://www.w3c.org/TR/2000/REC-xml-20001006>].
- [7] M. Murata, D. Lee, M. Mani (2000): „Taxonomy of XML Schema Languages using Formal Language Theory“. In Proc. of Extreme Markup Languages 2000.
- [8] G. Rehm (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.
- [9] K.E. Shafer (1995): Creating DTDs via the GB-Engine and Fred. [<http://www.oclc.org/fred/docs/sgml95.html>]
- [10] (1999) Resource Description Framework (RDF). Model and Syntax Specification. World Wide Web Consortium. [<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>]
- [11] (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>]
- [12] (1986) ISO 8879:1986. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Genf: International Organization for Standardization.
- [13] (2001) XML Schema. World Wide Web Consortium. [<http://www.w3c.org/XML/Schema>]
- [14] (2000) Extensible Markup Language (XML) Version 1.0 (Second Edition). World Wide Web Consortium. [<http://www.w3c.org/TR/2000/REC-xml-20001006>].
- [15] (1999) XPath Language Version 1.0. World Wide Web Consortium. [<http://www.w3c.org/TR/xpath>]
- [16] (1999) XSL Transformations Version 1.0. World Wide Web Consortium. [<http://www.w3c.org/TR/xslt>]
- [17] (2001) XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification. [<http://www.topicmaps.org/xtm/1.0/>]
- [18] (2002) XQuery 1.0: An XML Query Language. World Wide Web Consortium. [<http://www.w3.org/TR/xquery/>]



**Dr. Henning Lobin** ist seit 1999 Professor für Angewandte Sprachwissenschaft und Computerlinguistik an der Universität Gießen. Er studierte Linguistik und Informatik, promovierte 1991 an der Universität Bonn und habilitierte sich 1996 an der Universität Bielefeld. Aktuelle Forschungsvorhaben befassen sich mit der semantischen Modellierung von Dokumentstrukturen und der Übertragung von grammatiktheoretischen Modellen der Linguistik auf die XML-basierte Informationsstrukturierung. Adresse: Justus-Liebig-Universität Gießen, Angewandte Sprachwissenschaft und Computerlinguistik, Otto-Behaghel-Str. 10 D, D-35394 Gießen, E-Mail: [Henning.Lobin@uni-giessen.de](mailto:Henning.Lobin@uni-giessen.de)