

# Grammatische Restringierung von Dateninhalten in SGML/XML

Henning LOBIN

## 1. Einleitung

Bei der praktischen Anwendung von standardisierten SGML- oder XML-DTDs stellt sich oft das Problem, dass aufgrund ihrer beabsichtigten breiten Anwendbarkeit liberale Strukturvorgaben einzuschränken sind. Will ein Wörterbuch-Verlag beispielsweise das *base tag set* für Wörterbücher der TEI-Richtlinien (s. SPERBERG-MCQUEEN/BURNARD 1994) nutzen, müssen zusätzliche Regelungen getroffen werden, welche der durch die TEI-DTD erlaubten Strukturen untersagt sind und aus was für semantischen Einheiten eine Bedeutungsbeschreibung zusammengesetzt sein soll (vgl. LOBIN/WITT 1999).

Es ist verschiedentlich kritisiert worden, dass in XML/SGML nicht vorgesehen ist, Attribute und Element-Inhalte weiteren Restriktionen zu unterziehen (s. z.B. den Entwurf zu DDML). Auf der einen Seite erlauben zwar Attribute die Festlegung bestimmter Typen von Werten und die Spezifikation von Auswahllisten, es ist jedoch nicht möglich, komplexere Konstrukte z.B. durch reguläre Ausdrücke zu spezifizieren, wie sie auch als Inhaltsmodelle von Elementen erscheinen können. Inhaltsmodelle, auf der anderen Seite, erlauben zwar die Spezifikation von komplexen strukturellen Regularitäten, doch enden Element-Hierarchien immer in Inhaltsdaten, für die in XML lediglich die unspezifische Kategorisierung als #PCDATA, in SGML zusätzlich als CDATA und RCDATA möglich ist. #PCDATA kann dabei für alles zwischen einer leeren und einer aus Hunderttausenden von Zeichen bestehenden Zeichenkette stehen. Dieses Papier will zeigen, wie mit den Mitteln der sog. architektonischen Verarbeitung XML/SGML selbst zur Restringierung eingesetzt werden kann, um CDATA-Attribute und Element-Inhalte auf eine wohldefinierte Menge von möglichen Zeichenketten durch kontextfreie Regeln einzuschränken.

Architekturen, deren Struktur als eine der *SGML Extended Facilities* im HyTime-Standard kodifiziert worden ist, erlauben es, deklarativ spezifizierte Abbildungen einer DTD auf Meta-DTDs vorzunehmen, die in einem ähnlichen Verhältnis zu der sog. Client-DTD stehen, wie im Paradigma der objektorientierten Programmierung eine Klasse zu einem instanziierten Objekt (vgl. LOBIN 1999a). Die Nutzung von Architekturen ist somit unterhalb der Ebene der Daten-Konvertierung zu sehen; leistungsfähige SGML/XML-Parser, etwa NSGMLS von James Clark (siehe [www.jclark.com/sp](http://www.jclark.com/sp)), unterstützen die Möglichkeit der architektonischen Verarbeitung vollständig.

## 2. Zeichenketten und Grammatiken

Es ist sehr leicht, kontextfreie Regeln in XML/SGML-Element-Deklarationen umzusetzen (vgl. auch WITT 1999). Dabei sind lediglich alternative Regeln zu einer Kategorie durch Disjunktionkonnektoren zu vereinigen:

```
(1)  <!ELEMENT      s          (np, vp)>
      <!ELEMENT      np        (eigename)>
      <!ELEMENT      vp        (v-intr|
                                (v-trans, np-akk))>
      <!ELEMENT      np-akk    ((art-m, n-m)|
                                (art-f, n-f))>
```

Die kontextfreien Regeln können wir mit Elementen anreichern, die die Position von Leerzeichen und dem satzabschließenden Punkt markieren:

```
(2)  <!ELEMENT      s          (np, sp, vp, punkt)>
      <!ELEMENT      np        (eigename)>
      <!ELEMENT      vp        (v-intr|
                                (v-trans, sp, np-akk))>
      <!ELEMENT      np-akk    ((art-m, sp, n-m)|
                                (art-f, sp, n-f))>
```

Wenn wir die lexikalischen Regeln einer Phrasenstrukturgrammatik in SGML/XML exakt nachbilden wollen, genügt es allerdings nicht, den Wortklassenkategorien einfach als Inhaltsmodell (#PCDATA) zuzuordnen.

Die lexikalischen Kategorien müssen wir stattdessen weiter in Elemente untergliedern, die jeweils für eine spezifische lexikalische Einheit stehen:

```
(3)  <!ELEMENT      eigenname  (hans|katrin)>
      <!ELEMENT      v-intr    (lacht|singt)>
      <!ELEMENT      v-trans   (isst|holt)>
      <!ELEMENT      art-m     (einen|den)>
      <!ELEMENT      art-f     (eine|die)>
      <!ELEMENT      n-m      (apfel|braten)>
      <!ELEMENT      n-f      (birne|aprikose)>
```

Die lexikalischen Einheiten können wir nun als leere Elemente definieren, für die als ein fixiertes CDATA-Attribut genau die Zeichenkette spezifiziert ist, durch die sie zu realisieren sind:

```
(4)  <!ELEMENT      hans      EMPTY>
      <!ELEMENT      katrin    EMPTY>
      <!ELEMENT      lacht     EMPTY>
      ...
      <!ATTLIST      katrin
                string      CDATA      #FIXED "Katrin">
      <!ATTLIST      hans
                string      CDATA      #FIXED "Hans">
      <!ATTLIST      lacht
                string      CDATA      #FIXED "lacht">
      ...
```

Eine Dokument-Instanz, die beispielsweise die Zeichenkette "Hans isst einen Apfel" beschreibt, enthält somit an keiner Stelle direkt die Teilketten, aus denen dieser Satz zusammengesetzt ist:

```
(5)  <s><np><eigenname><hans/></eigenname></np><sp/>
      <vp><v-trans><isst/></v-trans><sp/><np><art-m>
      <einen/></art-m><sp/><n-m><apfel/></n-m></np></vp>
      <punkt/></s>
```

Wenn wir in dieser Dokument-Instanz nur diejenigen Elemente herausnehmen, die lexikalische Einheiten bezeichnen, also die abgeleitete Instanz

```
<hans/><sp/><isst/><sp/><einen/><sp/><apfel/><sp/>
<punkt/>
```

bilden, so ist klar, dass sich der konkrete Satz aus der Konkatenation der Zeichenketten ergibt, die bei allen diesen Elementen als fixiertes Attribut `string` vermerkt sind:

```
"Hans"+" "+"isst"+" "einen"+" "+"Apfel"+"."
→ "Hans isst einen Apfel."
```

Es ist evident, dass sich durch diese Methode alle Zeichenketten definieren lassen, die durch kontextfreie Grammatiken beschreibbar sind.

### 3. Restrangierung von CDATA-Attributen

Nehmen wir nun an, unsere DTD besteht aus lediglich einer Element-Deklaration:

```
(6)  <!ELEMENT      satz      EMPTY>
      <!ATTLIST    satz
              inhalt      CDATA      #REQUIRED>
```

Eine korrekte Dokument-Instanz zu dieser DTD wäre z.B. die folgende:

```
(7)  <!DOCTYPE satz SYSTEM "satz.dtd">
      <satz inhalt="Hans isst einen Apfel."/>
```

Wenn für das Attribut `inhalt` nur ganz bestimmte Zeichenketten als Wert angegeben werden sollen, können diese Ketten durch eine Grammatik definiert und durch entsprechende Deklarationen nach XML/SGML umgesetzt werden. Auf der Grundlage der Überlegungen zur Repräsentation von kontextfreien Regeln können wir leicht eine DTD erstellen, die statt des Attributs `inhalt` beim Element `satz` genau die als Beispiel aufgeführten Zeichenketten zu definieren erlaubt:

```
(8) <!ELEMENT      satz      (s)>
     <!ELEMENT      s      (np, sp, vp, punkt)>
     <!ELEMENT      np      (eigename)>
     ...
     <!ELEMENT      eigename (hans|katrin)>
     <!ELEMENT      v-intr  (lacht|singt)>
     <!ELEMENT      v-trans (isst|holt)>
     ...
     <!ELEMENT      hans    EMPTY>
     <!ELEMENT      katrin  EMPTY>
     <!ELEMENT      lacht   EMPTY>
     ...
     <!ATTLIST      hans
string            CDATA      #FIXED "Hans">
     <!ATTLIST      katrin
string            CDATA      #FIXED "Katrin">
     <!ATTLIST      lacht
string            CDATA      #FIXED "lacht">
     ...
```

Eine Dokument-Instanz dieser DTD für den gleichen Satz wie in (5) sieht dann folgendermaßen aus:

```
(9) <!DOCTYPE satz SYSTEM "satz-rstr.dtd">
     <satz><s><np><eigename><hans/></eigename></np>
     <sp/><vp><v-trans><isst/></v-trans><sp/><np>
     <art-m><einen/></art-m><sp/><n-m><apfel/></n-m>
     </np></vp><punkt/></s>
```

Wie kommen wir nun von dieser Dokument-Instanz zu der in (7)? Die architektonische Verarbeitung erlaubt uns, über Kontroll-Attribut-Werte zu Element-Inhalten zu machen. Weiterhin ist es möglich, architektonische CDATA-Attribute durch den Inhalt aller eingebetteten Elemente zu bilden. Wenn wir also die string-Attribute der lexikalischen Elemente der Instanz zunächst in Element-Inhalt überführen, kann auf der nächsten Stufe der gesamte Inhalt im inhalt-Attribut von satz 'zusammengesammelt' werden. Dazu ist es notwendig, dass wir zwischen die DTD in (6) und die in (8) eine temporäre DTD zwischenschalten. In dieser DTD brauchen nur noch die Elemente zu erscheinen, die string-Attribute

tragen, da die übrigen Elemente hier zu Restriktionszwecken keine Rolle mehr spielen. Darüber hinaus kann jede beliebige Kombination dieser Elemente zugelassen werden, da die korrekte, eingeschränkte Abfolge ja schon durch die DTD in (8) gewährleistet ist:

```
(10) <!ELEMENT   satz (hans|katrin|lacht|singt|isst|
                    holt|einen|den|eine|die|apfel|
                    braten|birne|aprikose|sp|
                    punkt)*>
      <!ELEMENT   hans   (#PCDATA)>
      <!ELEMENT   katrin  (#PCDATA)>
      <!ELEMENT   lacht   (#PCDATA)>
      <!ELEMENT   singt   (#PCDATA)>
      <!ELEMENT   isst    (#PCDATA)>
      ...
```

Soll nun die DTD (8) dieser Zwischen-DTD als Meta-DTD zugeordnet werden, müssen die entsprechenden architektonischen Deklarationen vorgenommen werden (s. LOBIN 1999a).

#### 4. Ein allgemeines Verfahren

Wir wollen uns im folgenden ansehen, wie für die Restringierung von Attributen und Daten-Elementen ein allgemeines Verfahren aussehen muss. Gegeben sei eine DTD  $D$ , in der eine Element-Deklaration vom Typ

```
<!ELEMENT      x          EMPTY>
<!ATTLIST     x
      att          CDATA          #REQUIRED>
```

vorkommt. Es sei weiterhin  $G$  eine Grammatik mit dem Startsymbol  $S$ , den Phrasenstrukturregeln  $P$  und den lexikalischen Regeln  $L$ . Aus  $D$  und  $S$  kann dann folgendermaßen eine restringierende DTD  $D'$  abgeleitet werden:

##### 1. Ersetze

```
<!ELEMENT      x          EMPTY>
<!ATTLIST     x
      att          CDATA          #REQUIRED>
```

durch

```
<!ELEMENT          x          (S)>
```

Füge weiterhin für alle Regelgruppen aus P der Form

```
A    →    B11 ... B1i
A    →    B21 ... B2j
...
A    →    Bn1 ... Bnk
```

die Element-Deklarationen

```
<!ELEMENT          a          ((b11, ..., b1i) |
                               (b21, ..., b2j) |
                               ...
                               (bn1, ..., bnk))>
```

in D' ein. Für alle Regeln aus L der Form

```
a    →    {s1, ..., sn}
```

füge die folgenden Deklarationen in D' ein:

```
<!ELEMENT          a          (t1, ..., tn)
<!ELEMENT          t1        EMPTY>
<!ELEMENT          t2        EMPTY>
...
<!ELEMENT          tn        EMPTY>
<!ATTLIST          t1
    string          CDATA #FIXED "s1">
    ArcNamrA       CDATA #FIXED "#ARCCONT string">
<!ATTLIST          t2
    string          CDATA #FIXED "s2">
    ArcNamrA       CDATA #FIXED "#ARCCONT string">
...
<!ATTLIST          tn
    string          CDATA #FIXED "sn">
    ArcNamrA       CDATA #FIXED "#ARCCONT string">
```

2. Erstelle aus D' die folgende intermediäre DTD D\*. Ersetze dazu in D die Deklarationen

```
<!ELEMENT          x          EMPTY>
<!ATTLIST          x
    att            CDATA          #REQUIRED>
```

durch

```

<!ELEMENT          x          (t1 | ... | tq)*>
<!ATTLIST
    ArcNamrA      CDATA #FIXED "att #CONTENT">
<!ELEMENT          t1        (#PCDATA)>
<!ELEMENT          t2        (#PCDATA)>
...
<!ELEMENT          tn        (#PCDATA)>

```

Dabei ist  $\{t_1, \dots, t_n\}$  die Menge der lexikalischen Elemente, die durch die Umsetzung der lexikalischen Regeln in Element-Deklarationen in  $D'$  entstehen.

3. Deklariere  $D^*$  zur Architektur von  $D'$ .

4. Deklariere  $D$  zur Architektur von  $D^*$ .

Wird aus einer validierbaren Dokument-Instanz von  $D'$  zunächst die architektonische Instanz  $i'$  bezüglich  $D^*$  generiert, dann aus dieser die architektonische Instanz  $i$  bezüglich  $D$ , so ist  $i$  eine validierbare Instanz von  $D$ .<sup>1</sup> Die Restringierung von Attribut-Werten läßt sich schematisch somit folgendermaßen darstellen:

DTD		Instanz
D	→	2. architektonische Instanz; Ziel-Dokument
↑		
$D^*$	→	1. architektonische Instanz
↑		
$D'$	↔	restringierende Dokument-Instanz

Restringierung von Attribut-Werten, schematisch

<sup>1</sup> Dies ist evident: Das CDATA-Attribut beim restringierten Element aus  $D$  wird in  $D'$  durch einen Teilbaum repräsentiert, dessen Blätter ebenfalls alle CDATA-Attribute aufweisen. Bei der Ableitung der architektonischen Instanz bezüglich  $D^*$  werden alle diese Attribut-Inhalte der vormals leeren Elemente zum CDATA-kodierten Element-Inhalt. Der nächste Schritt der architektonischen Verarbeitung fasst alle diese Elemente wieder in einem CDATA-Attribut zusammen.

## 5. Schlußbemerkung

Dieses einfache Beispiel hat gezeigt, dass es möglich ist, zu einer bestehenden Deklaration eines Daten-Elements oder eines Attributs in einer DTD eine Grammatik der möglichen Daten-Inhalte zu spezifizieren, diese in ein DTD-Fragment umzusetzen und die bestehende DTD damit zu modifizieren. Instanzen, die gemäß dieser modifizierten DTD korrekt gebildet sind, lassen sich allein durch architektonische Verarbeitung in korrekte Instanzen der ursprünglichen DTD überführen. Es ist also nicht möglich, auf diese Weise zu prüfen, ob Daten einer existierenden Instanz der Ausgangs-DTD die Restriktionen erfüllen oder nicht. Diese Eigenschaft der beschriebenen Technik prädestiniert sie somit für die Restringierung des Eingabevorgangs. Wird die Dokument-Erstellung auf der Grundlage der restringierten DTD in einem XML/SGML-Editor durchgeführt, ist es nicht möglich, inkorrekte Inhaltsdaten oder Attribut-Werte zu erzeugen. Jede validierbare Instanz der restringierten DTD ist auch eine validierbare Instanz der ursprünglichen DTD.

Der wesentliche Vorteil des beschriebenen Verfahrens liegt darin, dass es Standardverfahren der Linguistik direkt in den Bereich der SGML/XML-basierten Informationsverarbeitung zu übertragen erlaubt, ohne dass dazu ansonsten kaum motivierbare formale und notationelle Erweiterungen notwendig werden.

## Literatur

- DDML (1999): Document Definition Markup Language (DDML) Version 1.0. World Wide Web Consortium (<http://www.w3.org/TR/NOTE-ddml>).
- HyTime (1997): ISO/IEC 10744:1997. Information Technology – Hypermedia/Time Based Structuring Language (HyTime). Genf: International Organization for Standardization.
- LOBIN, Henning (1999a): Informationsmodellierung in XML und SGML. Heidelberg u.a.: Springer.
- LOBIN, Henning (Hrsg., 1999b): Text im digitalen Medium. Linguistische Aspekte von Textdesign, Texttechnologie und Hypertext Engineering. Wiesbaden: Westdeutscher Verlag.

- LOBIN, Henning und WITT, Andreas (1999): "Semantic and Thematic Navigation in Electronic Encyclopedias". Erscheint in: *Proceedings of Electronic Publishing 99*. Rønneby.
- SGML (1986): ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Genf: International Organization for Standardization.
- SPERBERG-MCQUEEN, C.M. and BURNARD, Lou (1994): Guidelines for Electronic Text Encoding and Interchange. Chicago und Oxford: Text Encoding Initiative.
- WITT, Andreas (1999): SGML und Linguistik. In: LOBIN (1999b), 121-153.
- XML (1998): Extensible Markup Language (XML) Version 1.0. World Wide Web Consortium.