Implementierung eines Algorithmus zur Speicherung von XML-Daten in objektrelationalen Datenbanken

Universität Rostock
Fachbereich Informatik
Franka Reuter
(Februar 2001)
Betreuer: Dr. Meike Klettke

Inhaltsverzeichnis

Zusammenfassung

Die Verbreitung semistrukturierter Daten steigt, besonders durch Anwendungen im World Wide Web gewinnen diese an Bedeutung. Eine typische Repräsentationsform für semistrukturierte Daten ist XML (E*X*tensible *M*arkup *L*anguage).Da viele Anwendungen nur eine Teilmenge des Umfangs von XML verwenden, wäre es für diese vorteilhaft, Daten in XML in relationale oder objektrelationale Datenbanken zu speichern. Damit ließen sich diese Daten innerhalb eines Datenbanksystems verwalten und anfragen. Der aufgeführte Algorithmus zeigt, wie eine Menge von zusammengehörigen XML Dokumenten auf eine objektrelationale Datenbankstruktur (Voraussetzung ist, dass das Datenmodell der objektrelationalen Datenbank um einen XML Datentyp erweitert wurde) abgebildet werden kann. Basierend auf der XML Document Type Definition (DTD) und erstellten Statistiken berechnet der Algorithmus eine optimale Abbildung der XML-Struktur auf eine objektrelationale Datenbankstruktur. Die Statistiken werden über vorhandene XML-Daten und Beispielanfragen erhoben. In diese Studienarbeit wird der Algorithmus zur Speicherung von XML-Daten in objektrelationalen Datenbanken implementiert und damit validiert. Der Datenbankentwurf wurde spezifisch für DB2 UDB vorgenommen.

1 Motivation

Das World Wide Web ist zu einer unerschöpflichen Informationsquelle herangewachsen. Allerdings steigt damit auch die Unübersichtlichkeit und es wird schwieriger, gezielt nach Informationen zu suchen.

In vielen Anwendungsgebieten haben semistrukturierte Daten eine zunehmende Bedeutung. XML ist eine typische Repräsentationsform strukturierter und semistrukturierter Daten. XML ist eine Teilmenge der SGML Markup Language und wurde speziell dem WWW angepasst.

XML ist eine vom W3C standardisierte Sprache für den Austausch von Dokumenten. Im Vergleich zu SGML ist XML besser für automatische Prozesse geeignet. Des weiteren ist XML nicht so komplex wie SGML. Es gibt zahlreiche Anwendungen, die XML einsetzen, jedoch bisher nur wenige Werkzeuge, die die Verarbeitung von XML unterstützten.

Viele Anwendungen nutzen nur einen Teil des Umfangs von XML. Für diese wäre es vorteilhaft, Daten in XML in objektrelationalen Datenbanken zu speichern. Damit ließen sich die Daten innerhalb eines Datenbanksystems verwalten und anfragen. XML Dokumente besitzen eine wohl geformte Struktur. Das bedeutet zum Beispiel, dass die Elemente korrekt ineinander verschachtelt sind.

DTDs beschreiben den Aufbau von mehreren ähnlich aufgebauten XML Dokumenten. Ein XML-Dokument ist gültig, wenn es eine dazugehörige Dokumenttyp-Deklaration besitzt und wenn das Dokument die darin formulierten Beschränkungen erfüllt[**?**]. Die Struktur eines XML Dokumentes kann nicht nur aus der DTD, sondern auch aus dem Dokument selbst ermittelt werden. Dies führt aber zu Problemen, wie zum Beispiel dem Erkennen von strukturellen Gemeinsamkeiten von mehreren XML Dokumenten. Ohne eine vorhandene DTD ist das Finden eines optimalen allgemeinen Schemas oder die Schema-Evolution schwieriger zu bewältigen.

2 Der Algorithmus

In diesem Abschnitt wird der zu implementierende Algorithmus beschrieben. Alle Informationen dieses Abschnittes sind dem Paper [**?**] entnommen. Dieser Algorithmus wurde im Rahmen dieser Studienarbeit implementiert und dadurch validiert.

2.1 Motivation

Ein Anwendungsfeld von XML, in dem viele Standards definiert sind, ist die Verwaltung von Multimedia-Dokumenten im großem Umfang in verteilten digitalen Bibliotheken. Auch wenn die Verwaltung der Dokumente, ihre Beschreibung und zusätzlichen Informationen in einer typischen digitalen Bibliothek in einer ähnlichen DTD resultiert, kann der Inhalt und die Verwendung dieser Dokumentensammlung jedoch sehr verschieden sein. Um dies zu verdeutlichen, wurden die folgenden Beispiele populärer digitaler Bibliotheken näher betrachten.

DBLP[1]: von Michael Ley erstellt und verwaltet, beinhaltet die Verwaltung von rund 150.000 XML-Dokumenten

Springer LINK Service[2]: bietet Volltextversionen von Zeitschriftenveröffentlichungen an, z.B. Artikel des VLDB Journal

Amazon Book Store[3]: eine e-commerce Anwendung, die CDs, DVDs und Bücher verkauft

Eine einfache DTD für die oben aufgeführten Beispiele könnte die folgende DTD sein.

```
<!ELEMENT publications (book | article | conference)*>
<!ELEMENT book (front, body, references)>
<!ELEMENT front (title, author+, edition, publisher)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (first, second, e-mail?)>
<!ELEMENT first (#PCDATA)>
...
<!ELEMENT edition (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT body (part+ | chapter+)>
<!ELEMENT part (ptitle, chapter+)>
<!ATTLIST part id ID #REQUIRED>
```

---

[3] http://dblp.uni-trier.de

[3] http://link.springer.de

[3] http://www.amazon.com

```
<!ELEMENT ptitle (#PCDATA)>
<!ELEMENT chapter (ctitle, section+)>
<!ATTLIST chapter id ID #REQUIRED>
<!ELEMENT ctitle (#PCDATA)>
<!ELEMENT section (stitle, paragraph+)>
....
<!ELEMENT references (publications*)>
<!ATTLIST references reftype (book | article| conferences
  | wwwaddress) "article" >
<!ELEMENT article (meat, body, references)>
<!ELEMENT meta (author+, title, conference)>
<!ELEMENT conference (editor+, conftitle, city, year)>
<!ELEMENT editor (first, second, e-mail?)>
<!ELEMENT first (#PCDATA)>
...
```

Die DTD ist aber nur ein Faktor, der die Abbildung von XML Strukturen in Datenbanken beeinflusst. Andere wichtige Aspekte bilden sowohl die Häufigkeit der Element- und Attributvorkommen in Dokumentensammlungen, als auch deren Häufigkeit in Anfragen. So enthalten z.B. die Dokumente des DBLP Systems Elemente mit gleicher Vorkommenshäufigkeit.

Die Anwender fragen hauptsächlich den Namen des Autors an, den Titel von Konferenzen, von Journalen oder von Artikeln, oder aber der Anwender sieht sich Referenzangaben, Auszüge von Artikeln oder Inhaltsangaben einer Konferenz an. Im Gegensatz dazu enthalten die Artikel des VLDB Journals von Springer viele Auszüge, Paragraphen und einfachen Text. Dennoch werden auch diese Artikel nach Namen des Autors, Titel, Ausgabe Nr. oder Nr. des Journals abgefragt. Bei Amazon können nicht nur Informationen über Bücher, sondern auch über Filme und DVD's abgefragt werden. Diese Suche wird nach den Buchtiteln, den Keywords, den Ähnlichkeitsklassifiktionen, den Autorennamen oder den ISBN Nummern durchgeführt. Zusätzlich kann man sich auch Besprechungen und Buchanmerkungen von anderen Kunden oder Journalisten ansehen.

Diese Beispiele verdeutlichen, wie unterschiedlich Dokumentensammlungen sein können und wie breit das Anfrageprofil gestreut sein kann.

Dieser Algorithmus stellt einen Mechanismus dar, der, bei Nutzung der vorhandenen Informationen, das Abbilden von XML Strukturen in objektrelationale Datenbanken effektiver gestaltet.

2.2 "Straight Forward" Mapping Method

Objektrelationale Datenbanken unterstützen die Benutzung von strukturierten oder geschachtelten Datenbankattributen in NF^2-Relationen. NF^2 steht für *Non First Normal Form*.

Die Elementenhierarchie einer DTD kann direkt (straight forward) auf geschachtelte Attribute einer Datenbank abgebildet werden. Da es unterschiedliche objektrelationale Modelle gibt, wird hier ein Basisdatenmodel verwendet, das dann auf

Tabelle 1: Abbildungsvorschriften

DTD	objektrelationale Struktur
Sequenz von Elementen	Attribute einer Relation
optionale Elemente	Attribut mit Nullwert
Attribute	Datenbankattribute (NOT NULL, DEFAULT VALUE)
Elemente mit Quantoren +,*	Menge oder Liste von Attributen *(SET OF, LIST OF)*
verschachtelte Elemente	*TUPLE OF*

existierende Systeme, wie z.B. IBM DB2 UDB abgebildet werden kann.
Das Basisdatenmodel definiert nicht nur eine Menge von Basisdatentypen, wie *string*, *numeric*, etc., sondern auch die folgenden Typkonstruktoren: *set-of*, *list-of* und *tuple-of*.
Desweiteren wird das Datenmodell auf die Verwendung von PNF Relationen begrenzt. PNF steht für *Partioned Normal Form*. Relationen in PNF haben auf jeder Stufe der Schachtelung einen flachen Schlüssel [**?**].
Elemente mit * oder + Quantoren können als Attribute von Typ *set-of* oder *list-of* dargestellt werden. Zusätzlich wird die Existenz eines XML Datentyps mit folgender Funktionalität:

- Speichert XML Fragmente
- Unterstützt die Auswertung von Pfadausdrücken
- Besitzt Volltextoperationen

vorausgesetzt. Tabelle 1 enthält einen Überblick auf die Vorschriften zum Abbilden von DTD-Komponenten auf eine objektrelationale Struktur. Diese Vorschriften sind nicht komplett. Einige der auftretenden Probleme können damit nicht gelöst werden. Folgende Probleme sind bekannt:

Rekursionen in der DTD: Die rekursiven Elemente werden ermittelt und auf eine eigene Relation abgebildet. In der Abbildung 1 werden Rekursionen durch gepunktete Linien dargestellt.

Mixed-content type model: Elemente vom Typ Mixed-content werden auf ein Attribut des Datentyps XML abgebildet.

Alternativen: Hier existiert keine goldene Regel. Es gibt unterschiedliche Verfahren, die von den beteiligten Elementen abhängen. Wenn z.B. alle Elemente vom Type #PCDATA oder EMPTY sind, kann man den Namen und den dazugehörigen Attributwert in einem Tupel speichern. Diese Abbildungsart verbessert deutlich die Nullwertbalance einer Relation, bewirkt aber einen neuen, von der Struktur der DTD abweichenden Anfrageaufbau. Sind die Elemente komplexerer Struktur, kann man entweder jedes in einer eigenen Relation speichern oder sie alle zusammen in einem Attribut des Datentyps XML ablegen.

Dies muss von Fall zu Fall entschieden werden.

Hyperlinks: XML kennt mindestens drei unterschiedliche Standards für Hyperlinks: ID, IDREF, XLink und XPointer.

ID, IDREF können nur innerhalb eines einzelnen, logischen Dokument eingesetzt werden. Das bedeutet, dass nach dem Einlesen der XML Daten in die objektrelationale Relation die ursprüngliche Funktion der ID oder IDREF aufgehoben wird[4]. Dadurch wird die Überprüfung der IDREF Information innerhalb einer Dokumentensammlung sehr aufwendig. Komplizierter wird es, wenn man IDREFS betrachtet. Konnten die ID und IDREF auf ein Attribut von Typ *string* abgebildet werden, muss man für IDREFS eine Liste von zweielementigen Tupeln aufbauen.

2.3 Erweiterung der ”Straight forward” Abbildung

Wie in Abschnitt 2.1 erläutert wurde, ist eine nur auf der DTD basierende Abbildung nicht effektiv genug. Deshalb wurde die ”Staight forward” Abbildungsmethode durch die Verwendung von Statistiken, die auf den Betrachtungen des Abschnittes 2.1 basieren, erweitert. Es werden drei statistische Werte ermittelt, die dann zu einem so genannten signifikanten Gesamtstatistikwert unter Verwendung einer bestimmten Gewichtung zusammengefasst werden. Es werden nun die drei Statistiken betrachtet[5].

1. **DTD Statistik:** die DTD-Statistik gibt das Maß der Strukturiertheit, das sich aus der DTD ableitet, an. Der statistische Wert w_S setzt sich aus drei Teilstatistiken zusammen. Diese drei Teilstatistiken sind:

   Position des Elementes in der Hierarchie:

   $$S_H = 1 - \frac{N_P}{N_D}$$

   N_P - Anzahl der Vorgänger
   N_D - maximale Tiefe der DTD-Graphen

   Bewertung des Quantors: S_Q - Auswertung des Quantors

   Die DTD Charakteristika mit den jeweiligen Gewichten sind in der Tabelle 2 dargestellt.

---

[4] wenn man von einer Dokumentensammlung ausgeht

[5] Die verwendete Nummerierung entspricht nicht einer Reihenfolge oder einer zugeordneten Priorität.

Tabelle 2: Auswertung der Quantoren

DTD Charakteristik		S_Q
Element	no quantifier	1
	+	0.75
	?	0.5
	*	0.25
Attribute	REQUIRED	1
	IMPLIED	0.5

Tabelle 3: Auswertung der Alternativen

DTD Charakteristik	S_A
mixed content	0
alternative on the same level exists	0.5
otherwise	1

Bewertung der Alternativen: S_A - Auswertung der Alternativen

Die DTD Charakteristika mit den jeweiligen Gewichten sind in der Tabelle 3 dargestellt.

Nun kann man w_S nach der folgenden Formel abschätzen:

$$w_S = \frac{S_A + S_Q + S_H}{3}$$

Einige Informationen können nicht aus der DTD entnommen werden. Z.B. weiß man nicht, wie häufig ein Element mit dem Quantor ? oder * in der Dokumentensammlung tatsächlich vorhanden ist. Dies trifft auch auf Attribute zu, die als IMPLIED deklariert sind. Deshalb wurde die nun folgende Statistik eingeführt.

2. **Statistik über XML-Daten:** diese Statistik enthält die Anzahl der Dokumente, die ein Element oder ein Attribut enthalten. Die Formel zur Berechnung der Statistik über XML-Daten lautet:

$$w_D = \frac{D_A}{D_T}$$

D_A - Anzahl der Elemente / Attribute
D_T - Gesamtanzahl der XML-Dokumente

Die DTD enthält keine Informationen darüber, wie oft in Anfragen auf welche Elemente zurückgegriffen wird oder welche Elemente nur von geringem Interesse sind. Kennt man aber das Anfrageverhalten der Anwender und berücksichtigt diese Informationen im Abbildungprozess, so führt dies zu einem optimierten Datenbankentwurf. Sind Informationen über das Anfrageverhalten vorhanden, dann sollte die nächste Statistik berechnet werden.

3. **Statistik über Beispiel-Anfragen:** die Statistik über Beispiel-Anfragen ermittelt die Anzahl der Anfragen, die ein Element oder ein Attribut enthalten. Die Formel zur Berechnung der Statistik über Beispiel-Anfragen lautet:

$$w_Q = \frac{Q_A}{Q_T}$$

Q_A - Anzahl der Elemente / Attribute

Q_T - Gesamtanzahl der Beispiel-Anfragen

Diese Teilstatistiken werden unter Verwendung der Formel zur Signifikanz zusammengefasst:

$$w_Q = \frac{1}{4}w_D + \frac{1}{4}w_Q + \frac{1}{2}w_S$$

2.4 Algorithmusbeschreibung

Nachdem die Statistiken definiert sind, kann man nun folgenden Algorithmus entwickelt.

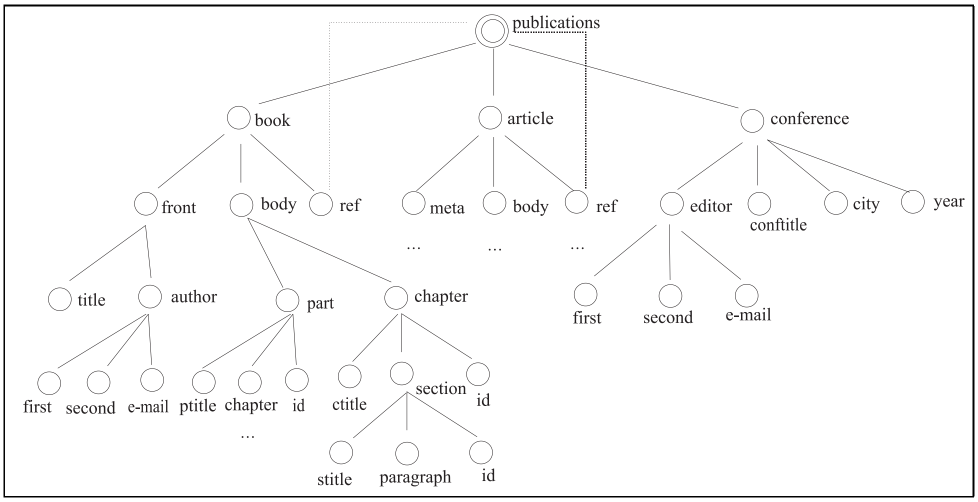


Abbildung 1: Hierarchie der Elemente / Attribute

1. **Aufbau eines DTD-Graphen:** für den DTD-Graphen wird die Baumstruktur verwendet (siehe Abbildung 1). Im Baum werden die Elemente, die Subelemente enthalten, als Knoten dargestellt, hingegen die einfachen Elemente (des Typs #PCDATA) und Attribute von Elementen als Blätter.
2. **Bestimmen der Signifikanz für jeden Knoten:** für jeden Knoten des DTD-Graphen werden die oben genannten Statistikberechnungen durchgeführt.
3. **Festlegen eines Limits:** der Anwender bestimmt ein Limit, dessen Höhe ausschlaggebend für den Abbildungsprozess der Elemente ist.
4. **XML Anteile bestimmen:** während dieses Prozesses wird der DTD-Graph nach Knoten durchsucht, die folgende Bedingungen erfüllen:
    - Der Knoten ist kein Blatt des DTD-Graphen.
    - Die Signifikanzen des Knoten und seiner Unterknoten liegen unterhalb des Limits.
    - Es existiert kein Vorgänger des Knotens, der dieselben Bedingungen erfüllt.

    Sind diese Bedingungen erfüllt, wird der Knoten und seine Unterknoten auf ein Attribut vom Datentyp XML abgebildet.
5. **Abbildung der regulären Anteile vornehmen:** ist eine der unter viertens genannten Bedingungen nicht erfüllt, wird der Knoten, in Abhängigkeit seines Quantors, abgebildet wie in Abschnitt 2.3 beschrieben.

3 Umsetzung in Modulstruktur

Der in Abschnitt 2 vorgestellte Algorithmus eignet sich sehr gut für eine Implementierung in Modulstruktur. Die Abbildung 2 stellt eine mögliche Umsetzung dar.
In Vierecken dargestellte Objekte repräsentieren Programmeingabedaten. Der Anwender muss folgende Daten angeben:

DTD Document Type Definition: Eine DTD beschreibt den strukturellen Aufbau einer Klasse von Dokumenten durch Festlegen von Regeln für die Definition von Elementen, Attributen und anderen XML-Daten. (Angabe erforderlich)

XML-Dokumentenkollektion: Eine XML-Dokumentenkollektion ist eine Sammlung von wohl geformten XML-Dokumenten, die alle Elemente derselben Klasse sind. Die Klasse wird durch die o.g. DTD beschrieben. (Angabe optional)

Beispiel-Anfragenkollektion: Eine Beispiel-Anfragenkollektion ist eine durch Statistiken ermittelte Sammlung von häufig gestellten Anfragen nach, in der XML-Dokumentenkollektion enthaltenen Daten. (Angabe optional)

Userprioritäten: User Prioritäten sind Informationen unterschiedlicher Art. Dies kann ein Limit sein oder auch eine explizite Abbildungsvorschrift für ein bestimmtes Element. (Limitangabe erforderlich, restliche Angaben optional)

Die DTD, die XML-Dokumente und die Beispielanfragen werden durch spezielle Parser eingelesen und verarbeitet. Die Parser werden durch doppelt umrahmte Vierecke dargestellt. Es werden drei Parser verwendet:

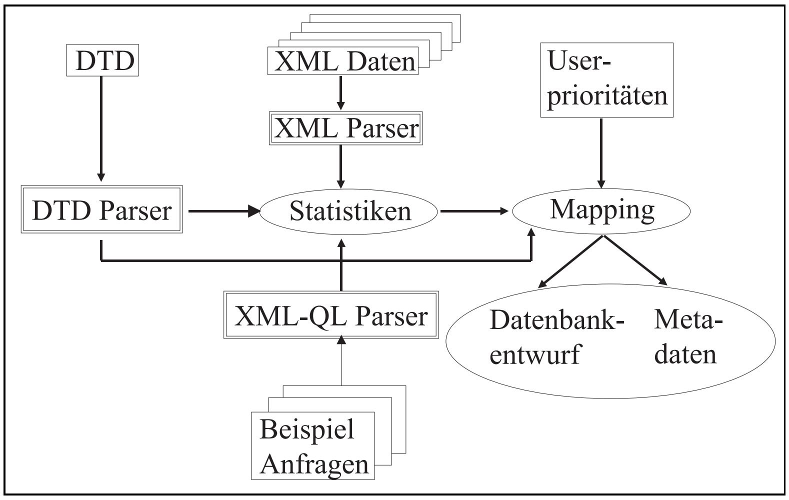


Abbildung 2: Umsetzung in Modulstruktur

DTD-Parser: Der DTD-Parser liest die DTD ein und liefert die darin enthaltenen Informationen für jedes Element zurück.

XML-Parser: Der XML-Parser liest die XML-Dokumente der XML-Dokumentenkollektion ein und liefert je Dokument die darin enthaltenen Elemente und deren Attribute zurück.[6]

XML-QL-Parser: Der XML-QL-Parser liest die Anfragendokumente der Beispiel-Anfragenkollektion ein und liefert je Anfragendokument die darin enthaltenen Elemente und deren Attribute zurück.

Diese ermittelten Informationen und die User Prioritäten können nun in den Modulen verarbeitet werden. Die Module werden als Ovale in der Abbildung 2 dargestellt. Es können drei Module gebildet werden. Ein Modul für die Erstellung der Statistiken, eins für den Abbildungsprozess und das letzte für die Aufstellung des Datenbankentwurfs zusammen mit den Metadaten. In später folgenden Abschnitten 5.2, 5.3 und 5.4 wird detaillierter auf diese Module eingegangen.

---

[6] Der XML-Parser besitzt eine weit größere Funktionalität, die hier aber nicht benötigt wird.

4 Software

Für die Implementierung wurde Python ausgewählt, da es zur Zeit der Implementierung des Algorithmus die einzige Programmiersprache war, für die ein DTD-Parser existierte. Der DTD-Parser ist Teil eines XML-Parsers, der von Lars Marius Garshol unter der Verwendung von Python programmiert wurde. Python ist eine Script Sprache. Sie kombiniert die Vorteile einer objektorientierten Sprache mit den Vorteilen einer iterativen Sprache.

4.1 Der XML-Parser xmlproc

xmlproc ist ein nicht validierender Parser. Dieser Parser wurde in Python implementiert. Nähere Informationen findet man im Internet unter:
http://www.stud.ifi.uio.no/∼lmariusg/download/python/XML.

4.2 Der xmlproc DTD-Parser

Der DTD-Parser ist das Modul, das xmlproc nutzt, um DTDs zu lesen. Der DTD-Parser lieferte für jedes Element ein Contentmodel zurück. Das Contentmodel (siehe Abbildung 3) ist ein Tupel, bestehend aus drei Elementen:

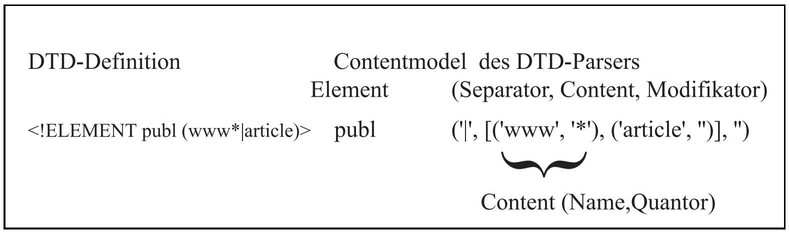


Abbildung 3: Beispiel einer DTD-Parser-Ausgabe

Separator: der Separator ist ein Element aus der Menge {|; , ; 1}.

Content: der Content ist ein Tupel, bestehend aus dem Namen des Elementes und dessen Quantors.

Modifikator: der Modifikator ist ein Element aus der Menge {+, *, ?, 1}

Für jedes Attribut eines Elementes liefert der DTD Parser den Namen, den Typ, die Deklaration und den Defaultwert zurück. Diese Informationen werden in einem Dictionary (Python Datenstruktur) abgelegt. Ein Dictionary besteht aus einem Schlüssel und einem zum Schlüssel gehörenden Wert. Dieser Wert kann alles beinhalten - von einem String bis zu einer Liste. Dies macht das Dictionary zu einem idealen Werkzeug und wird dementsprechend oft während der Implementierung eingesetzt.

5 Module

Im Abschnitt 3 wurde bereits eine mögliche Umsetzung in eine Modulstruktur beschrieben. Bevor man sich näher damit beschäftigen kann, ist es zwingend notwendig, erst eine andere Thematik zu besprechen.

5.1 Problematik der Rekursion

Rekursionen bilden nur in DTD's ein Problem, da sie in XML-Dokumenten nicht auftreten. Um ein reibungsloses Auswerten der DTD zu garantieren, müssen diese Rekursionen erkannt und beseitigt werden. Das folgende Beispiel stellt unterschiedliche Rekursionen in der DTD dar.

Fall 1

```
<!ELEMENT publ  (book | article |  publ )>
```

Fall 2

```
<!ELEMENT publ  (book | article )>
<!ELEMENT book ((front, body) | publ)>
```

Fall n

```
<!ELEMENT publ  (book | article )>
<!ELEMENT book ((front, body))>
...
<!ELEMENT article ( ref | (title, author+))>
...
<!ELEMENT ref (publ | library)
```

Implementiert wurde der allgemeinste Fall (Fall n), der die anderen mit erfasst.
Der Algorithmus zur Erkennung von Rekursionen durchläuft den Baum (DTD Graph) inorder, wie in Abbildung 4 mittels der Nummerierung dargestellt wird. Der Einstiegspunkt für jedes zu überprüfende Element bildet das Element selbst. Beim Durchlaufen wird das aktuelle Element mit dem zu überprüfenden Element verglichen. Im Fall einer Rekursion (Ermitteln einer Rekursion wird in der Abbildung 4 durch dicke Pfeile dargestellt.) wird das aktuelle Element aus dem Dictionarywert des Parentelementes gelöscht. Abbildung 5 zeigt das Dictionary vor und nach der Rekursionsbeseitigung.
Der Prozess der Rekursionsermittlung und -auflösung muss für alle Elemente durchgeführt werden. Nach Beendigung dieses Algorithmus liegen keine Rekursionen mehr vor.

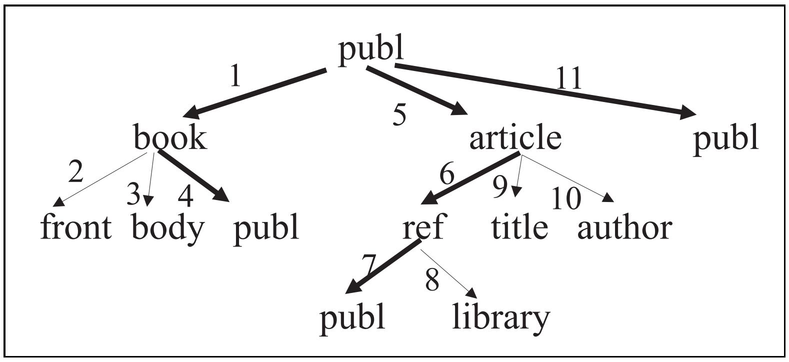
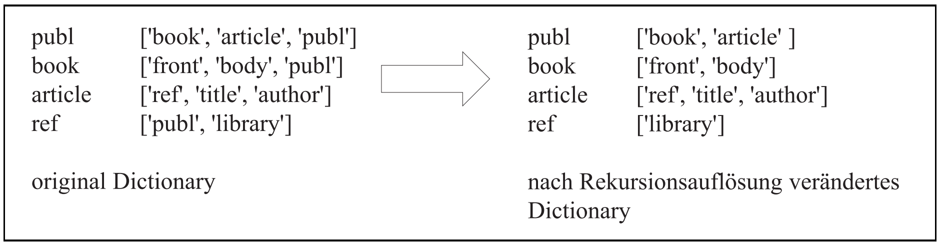


Abbildung 4: ’inorder’ Durchlauf durch den DTD-Graphen



publ	['book', 'article', 'publ']
book	['front', 'body', 'publ']
article	['ref', 'title', 'author']
ref	['publ', 'library']

original Dictionary

publ	['book', 'article']
book	['front', 'body']
article	['ref', 'title', 'author']
ref	['library']

nach Rekursionsauflösung verändertes Dictionary

Abbildung 5: Rekursionsbeseitigung im Dictionary

5.2 Das Statistikmodul

Die Verwendung von Statistiken zur Verbesserung von Resultaten ist nicht neu. So werden z.B. statistische Werte bei der Optimierung von Datenbankanfragen eingesetzt. In diesem Fall werden zur Verbesserung des Datenbankentwurfs drei statistische Werte berechnet, die dann mit einer bestimmten Gewichtung zu einem Gesamtstatistikwert, einer Signifikanz, zusammengefasst werden. Nähere Informationen findet man im Abschnitt 2.3. Zur Speicherung der ermittelten Werte wird für jedes Element und Attribut eine Instanz der Klasse Statistik generiert.

5.2.1 DTD Statistik

Zur Berechnung der DTD-Statistik müssen die im Abschnitt 2.3 genannten Teilstatistiken berechnet werden.

Für die Teilstatistik ”Position eines Elementes in der Hierarchie” muss zuerst die maximale Tiefe der DTD ermittelt werde, um dann für jedes Element, unter Verwendung des DTD-Parser, S_H berechnen zu können. Der ermittelte S_H - Wert wird dann in der zugehörigen Instanz gespeichert.

Die Berechnungen der beiden Teilstatistiken ”Bewertung des Quantors” und ”Bewertung der Alternativen” können unabhängig voneinander durchgeführt werden. Die ermittelten S_Q- und S_A-Werte werden wiederum in der zugehörigen Instanz

gespeichert.
Zum Schluss der DTD-Statistik müssen die drei Teilstatistiken unter Nutzung der Berechnungsformel von w_S zusammengefasst werden.

5.2.2 Statistik über XML-Daten

Falls der Anwender beim Programmaufruf Pfadangaben für die XML-Daten vorgenommen hat, kann man die Statistik über XML-Daten erstellen. Hierbei wird an den XML-Parser xmlproc jedes Dokument der XML-Dokumentensammlung einzeln übergeben. Der XML-Parser liest das Dokument ein und gibt alle darin enthaltenen Elemente und Attribute inklusive ihre Pfadangaben sequentiell zurück. Die jeweiligen Rückgabewerte werden in einem Dictionary als Schlüsselwerte gespeichert und dabei gleichzeitig gezählt. Abbildung 6 stellt das Ergebnis der Zählung dar. Nach der Zählung der Elemente und Attribute müssen diese in die dazugehörige Instanz der Klasse Statistik gespeichert werden.
Während der Implementierung stellte sich heraus, dass die unter Abschnitt 2.3 aufgestellte Formel für w_D nicht für alle Elemente verwendet werden konnte, sondern nur für das Wurzelelement. Für alle anderen Elemente gelten die folgenden Formeln:

$$\begin{aligned} w_D &= \frac{D_A(Element) * w_D(Parentelement)}{D_A(Parentelement)} \\ max(w_D) &= 1 \end{aligned}$$

Diese Formeln drücken eine Normierung durch den w_D-Wert des Parentelementes aus und setzen eine Obergrenze von 1 fest.
Ein spezielles Feature der Funktion zur Berechnung der Statistik über XML-Daten ist, dass das Startelement der DTD nicht in den XML-Daten enthalten sein muss [7]. Dies hat den Vorteil, dass man Dokumente mehrerer Klassen, unter Ausnutzung von Statistiken, in einer objektrelationalen Datenbank speichern kann [8].

```
article 266 time(s), attr = {'key': 266}
article.author 416 time(s), attr = {}
article.journal 266 time(s), attr = {}
article.pages 266 time(s), attr = {}
```

Abbildung 6: Ergebnisausgabe nach Parsen der XML-Daten

---

[7] Alle anderen Element müssen enthalten sein, sonst kommt es zu Verfälschungen in der Statistik.

[8] durch Einführen eines künstlichen Wurzelelementes

5.2.3 Statistik über Beispiel-Anfragen

Als XML-Anfragensprache wurde wegen der Ähnlichkeit zu SQL, XML-QL gewählt. Wer weitere Informationen zum Thema XML-QL benötigt, kann im Hauptseminar SS 2000 der Universität Rostock, Fachbereich Informatik, Lehrstuhl für Datenbank- und Informationssysteme [**?**] nachsehen.
XML-QL bietet eine Vielfalt von Anfragekonstruktionen (Abbildung 7), ermöglicht die Anwendung von wildcards (Abbildung 8, Beispiel A) und fordert keine vollständige Pfadangabe der Elemente zu Beginn der Anfrage (Abbildung 8, Beispiel B).
Der XML-QL Parser löst diese Probleme und liefert je eingelesener Beispiel-Anfrage (≡ eine Datei) eine Liste der darin enthaltenen Elemente und Attribute, ohne Berücksichtigung der Häufigkeit ihres Auftretens, zurück. Es werden alle in den Listen vorkommenden Elemente und Attribute in einem Dictionary gespeichert. Ist das Element bzw. das Attribut bereits Element der Schlüsselmenge des Dictionaries, wird der dazugehörige Wert um Eins erhöht. Der Endzustand eines Dictionaries wird in Abbildung 9 gezeigt.

where ... construct
construct {where ... construct }
where ... construct ... {where ... construct }
{where ... construct } {where ... Construct }
Funktion ... where ... construct End

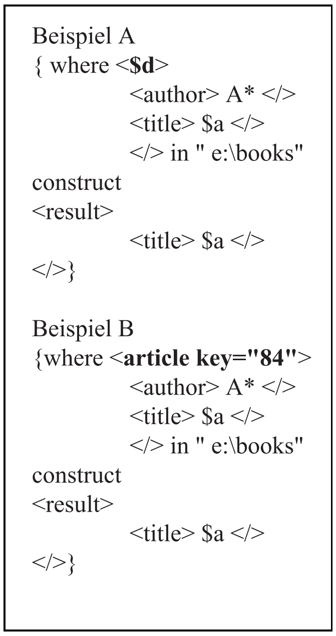
Abbildung 7: Verschiedenartigkeit von Anfragen in XML-QL

5.2.4 Das Ergebnis der Statistikberechnungen - die Signifikanz

Am Ende des Statistikmodules wird für jede Instanz der Klasse Statistik die Signifikanz berechnet. Die folgende Tabelle 4 zeigt für einen Teil der Elemente die Signifikanz. Abschließend sollte zu den Statistiken gesagt werden, dass die DTD-Statistik immer berechnet wird, die anderen beiden Statistiken nur, wenn der Anwender die benötigten Programmeingaben macht.

5.3 Das Mapping Modul

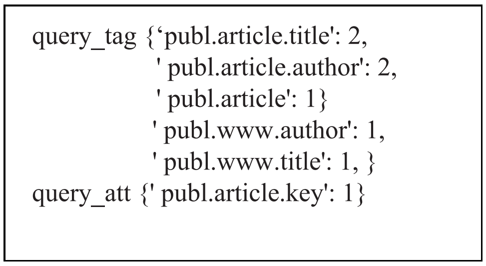
Das Mapping Modul bildet das Kernstück der Implementierung. Abbildung 10 stellt das Mapping Modul dar.
Den Abbildungsprozess kann man in zwei Phasen unterteilen. In der zweiten Phase wird die in Abschnitt 2.2 beschriebene ”Straight forward” Abbildung durchgeführt. In Phase Eins werden die in Abschnitt 2.3 erläuterten Statistiken zusammen mit den in Abschnitt 3 erläuterten User Prioritäten ausgewertet.



```
Beispiel A
{ where <$d>
            <author> A* </>
            <title> $a </>
            </> in " e:\books"
construct
<result>
            <title> $a </>
</>}

Beispiel B
{where <article key="84">
            <author> A* </>
            <title> $a </>
            </> in " e:\books"
construct
<result>
            <title> $a </>
</>}
```

Abbildung 8: Beispiel-Anfragen in XML-QL



```
query_tag {‘publ.article.title': 2,
               ' publ.article.author': 2,
               ' publ.article': 1}
              ' publ.www.author': 1,
              ' publ.www.title': 1, }
query_att {' publ.article.key': 1}
```

Abbildung 9: Dictionary nach Parsen der Beispiel-Anfragen A und B

Bevor die einzelnen Phasen näher erklärt werden, sollten einige Details des Abbildungsprozesses vorher erläutert werden. In Abschnitt 2.2 Tabelle 1 wurde dargestellt, wie die Elemente der DTD in Abhängigkeit ihrer Quantoren und Separatoren auf unterschiedliche Datentypen abgebildet werden. Diese Datentypen werden im Mapping Modul durch Klassendefinitionen modelliert. (In [**?**]wird eine andere Abbildung verwendet. Es wird die DTD auf eine Klasse abgebildet.) So wird z.B. durch ein Element mit dem Quantor * die Klasse db_list instanziert. Die folgenden Klassen sind definiert:

- db_attr: für einfache Elemente vom Typ #PCDATA oder aber Elemente, die auf den Datentyp XML abgebildet werden sollen
- db_tuple ⟨⟩ : für komplexe Elemente (Elemente, die Subelemente enthalten)

Tabelle 4: Signifikanz der Elemente

Element	Signifikanz
publ	0.666667
publ.article	0.571685
publ.article.author	0.458333
publ.article.key	0.571685
publ.article.title	0.446685
publ.article.url	0.446685
publ.article.year	0.446685

- db_list [] : für Elemente mit dem Quantor * oder + (bei Sequenzen)
- db_set {} : für Elemente mit dem Quantor * oder + (bei Alternativen)
- db_rec_attr: für rekursive Elemente und für Elemente, die mehr als einmal als Subelemente auftreten.

Jede Klasse besitzt ihre eigene Repräsentationsform, die die Grundlage für das Datenbankentwurfsmodul bildet.
Es werden nun die zwei Phasen des Abbildungsprozesses näher beschrieben.

5.3.1 Phase Eins - Die Vorselektion

In der Vorselektion werden die User-Limit-Vorgabe und die eventuell vorhandenen expliziten Abbildungsvorschriften für Elemente der DTD verarbeitet. Dies geschieht in zwei Prozessen (im weiteren Prozess A und Prozess B genannt).
Der Prozess A überprüft, ob der Anwender für das aktuelle Element eine Abbildungsvorschrift vorgegeben hat. Dabei werden drei Fälle unterschieden:

- Der Anwender wünscht eine strukturierte Abbildung. In diesem Fall kann der Prozess B übersprungen werden und man fährt mit Phase zwei fort.
- Der Anwender wünscht eine Abbildung auf dem Datentyp XML. Das Element wird auf dem Datentyp XML abgebildet und alle seine Subelemente werden markiert, so dass sie den Abbildungsprozess nicht durchlaufen können.
- Der Anwender hat keine Angaben gemacht. Es wird mit Prozess B fortgefahren.

Der Prozess B überprüft die Signifikanz des aktuellen Elementes mit der User-Limit-Vorgabe. Hierbei ist zu beachten, dass dieser Vergleich nicht nur für das aktuelle Element, sondern auch für alle seine Subelemente durchgeführt werden Muße (in Abschnitt 2.3 wurde gesagt: ”für den Knoten und alle seine Nachfolger”).

Ergibt die Überprüfung, dass das Element strukturiert abgebildet werden kann, wird dieses Element in Phase Zwei weiterverarbeitet. Im entgegengesetzten Fall wird das Element, wenn für keine seiner Subelemente strukturierte Abbildungsvorschriften vorliegen, auf den Datentyp XML abgebildet und alle seine Subelemente markiert, so dass sie den Abbildungsprozess nicht durchlaufen können. Falls für eins oder mehrere seiner Subelemente strukturierte Abbildungsvorschriften vorliegen, wird das Ergebnis des Prozesses B ignoriert und mit Phase zwei fortgefahren.

5.3.2 Phase Zwei - die Abbildung

Nachdem die Vorselektion erfolgreich durchlaufen wurde, kann nun das jeweilige Element auf den entsprechenden Datentyp, wie bereits oben beschrieben, abgebildet werden. Bevor das Element verarbeitet werden kann, müssen zuerst seine eventuell vorhandenen Attribute ausgewertet werden. Die Attribute müssen vorher ermittelt werden, um später zu bestimmen, dass z.B. ein einfaches Element mit Attributen auf den Datentyp *tupel-of* abgebildet werden muss.

Jetzt kann das Element verarbeitet werden. Zuerst wird der Separator ausgewertet, dann der Quantor und zuletzt der Modifikator. Zwischen der Quantor- und Modifikator-Auswertung werden zusätzlich die in der Rekursionsermittlung gesammelten Informationen (rekursiven Elemente und multi used Elemente) verarbeitet. Ist das aktuelle Element ein mixed content type Element wird es in diesem Algorithmus auf den Datentyp XML abgebildet. Falls man zukünftig auch die mixed content type Elemente strukturiert abbilden möchte, wären nur geringe Anpassungen in der Implementierung nötig.

Besitzt das aktuell abzubildende Element Subelemente, erfolgt ein rekursiver Funktionsaufruf für jedes Subelement.

5.3.3 Ergebnis des Abbildungsprozesses

Das Resultat des Abbildungsprozesses ist ein String, der als Basis für den Datenbankentwurf dient.

```
publ: <www:{www:<key:str, author:str, title: str,
                 url:str , year: str >
           },
       article:<key:str, author:str, title:str,
                year:str >
      >
```

”Übersetzung des Strings”

Der String zeigt, dass publ ein Tupel, bestehend aus www und article ist. www ist ein Set von Tupeln, bestehend aus key, author, title, url und year, die alle vom Typ *string* sind. article ist ein Tuple bestehend aus key, author, title und year, die alle vom Typ *string* sind.

Während des gesamten Abbildungprozesses werden zusätzliche Informationen für den Datenbankentwurf gesammelt. Diese sind:

- Speichern der Elemente, die keine Null Werte enthalten dürfen
- Speichern von Defaultwerten für Attribute
- Speichern von CHECK Bedingungen für Attribute.

Diese Informationen werden später während der Tabellendefinition (Abschnit 5.4) verarbeitet.

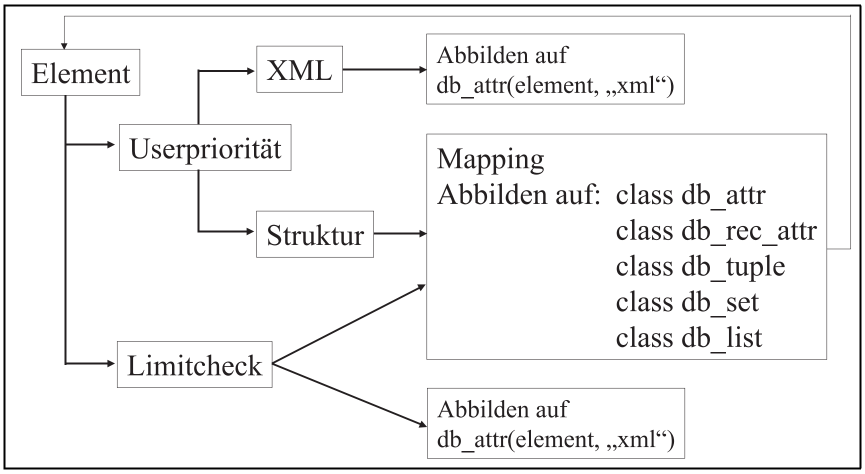


Abbildung 10: Darstellung des Mappingprozesses

5.4 Das Datenbankentwurfsmodul

In diesem Modul wird für ein konkretes Datenbanksystem, IBM DB2 UDB unter Verwendung der DDL (Data Definition Language) ein Datenbankentwurf erstellt.
Das Ergebnis des Abbildungsprozesses enthält, wie in Abschnitt 5.3.3 beschrieben, die interne Vorstufe des Datenbankentwurfsprozesses, die im Modul Datenbankentwurf analysiert wird. Darauf aufbauend werden Typdefinitionen und Tabellendefinitionen erstellt. Die objektrelationale Erweiterung von DB2 definiert keine *tuple-of-, list-of-, set-of-* Typkonstruktoren [**?**]. Wurden im Abbildungsprozess Elemente auf diese Typen abgebildet, müssen diese jetzt durch Ausgliederung in eigene Relationen und durch Einführung von Schlüsselbeziehungen modelliert werden. Das verkompliziert den Datenbankentwurfsprozess erheblich.
Im nun folgenden Abschnitt wird die Modellierung der *list-of* und *set-of* Typkonstruktoren näher beschrieben. Der *tupel-of* Typkonstruktor stellt kein Problem dar. Er kann durch eine *scope*-Beziehung in DB2 UDB dargestellt werden.

5.4.1 Modellierung der *list-of* und *set-of* Typkonstruktoren

Verschachtelte Relationen werde in DB2 UDB durch Typdefinitionen realisiert [**?**]. Das Problem der Modellierung der *list-of* und *set-of* Typkonstruktoren liegt nicht in ihrer Erkennung, sondern in der richtigen Abbildung und der Bestimmung der richtigen Schlüsselbeziehung.

5.4.1.1 Modellierung des *list-of* Typkonstruktors Wird ein Element der Typdefinition als Liste markiert, müssen die folgenden Schritte durchgeführt werden (Erläuterung an einem Beispiel):

- Entfernung des Listenelementes aus der Typ- und Tabellendefinition des Parentelementes.
- Anlegen einer Typ- und Tabellendefinition für das Listenelement.
- Hinzufügen des Parentelementes in die Typ- und Tabellendefinition des Listenelementes.
- Hinzufügen des liste_sequ Attributs in die Typ- und Tabellendefinition des Listenelementes und Erstellen einer Instanz der Klasse meta_Type (nhere Erläuterung findet man im Abschnitt /refMetafür das list_sequ Attribut.

Die Resultate der o.g. Schritte kann man in den Abbildungen 11 und 12 nachvollziehen . Die Pfeile in den Abbildungen 11 und 12 geben die *scope*-Beziehungen wieder.
Die folgende Beispiel-DTD dient zur Erläuterung der Modellierung des *list-of* Typkonstruktors:

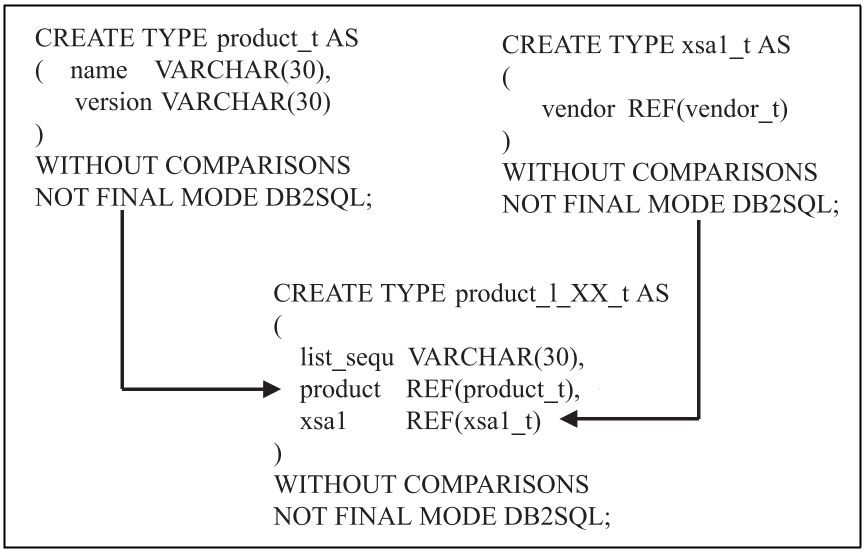
```
<!ELEMENT xsa1 (product*,vendor)>
<!ELEMENT vendor   (name)>
<!ELEMENT name     (#PCDATA)>
<!ELEMENT product  (name,version)>
<!ELEMENT version  (#PCDATA)>
```

Rückgabestring des Mapping Moduls:

```
xsa1: <product_l_XX: [product: <name: str , version: str >],
      vendor: <name: str >>
```

5.4.1.2 Modellierung des *set-of* Typkonstruktors Wird ein Element der Typdefinition als Set markiert, ist die Vorgehensweise der Modellierung in DB2 ähnlich der Behandlung des *list-of* Typekonstruktors.

- Entfernung des Setelementes aus der Typ- und Tabellendefinition des Parentelementes.
- Anlegen einer Typ- und Tabellendefinition für das Setelement.

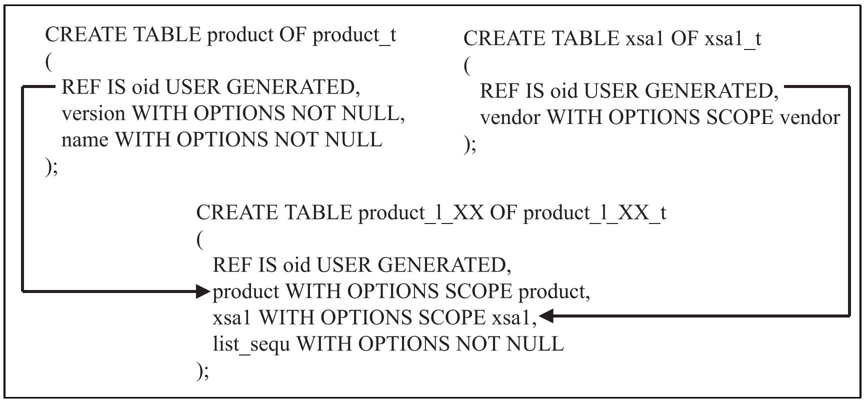


```
CREATE TYPE product_t AS
(   name    VARCHAR(30),
    version VARCHAR(30)
)
WITHOUT COMPARISONS
NOT FINAL MODE DB2SQL;

CREATE TYPE xsa1_t AS
(
    vendor  REF(vendor_t)
)
WITHOUT COMPARISONS
NOT FINAL MODE DB2SQL;

CREATE TYPE product_l_XX_t AS
(
   list_sequ  VARCHAR(30),
   product    REF(product_t),
   xsa1       REF(xsa1_t)
)
WITHOUT COMPARISONS
NOT FINAL MODE DB2SQL;
```

Abbildung 11: Modellierung des *list-of* Typkonstruktors in der Typdefinition



```
CREATE TABLE product OF product_t
(
  REF IS oid USER GENERATED,
  version WITH OPTIONS NOT NULL,
  name WITH OPTIONS NOT NULL
);

CREATE TABLE xsa1 OF xsa1_t
(
  REF IS oid USER GENERATED,
  vendor WITH OPTIONS SCOPE vendor
);

CREATE TABLE product_l_XX OF product_l_XX_t
(
  REF IS oid USER GENERATED,
  product WITH OPTIONS SCOPE product,
  xsa1 WITH OPTIONS SCOPE xsa1,
  list_sequ WITH OPTIONS NOT NULL
);
```

Abbildung 12: Modellierung des *list-of* Typkonstruktors in der Tabellendefinition

- Hinzufügen des Parentelementes in die Typ- und Tabellendefinition des Setelementes.
- Definieren eines Primary Key in der Tabellendefinition des Setelements zur Vermeidung von Duplikaten.

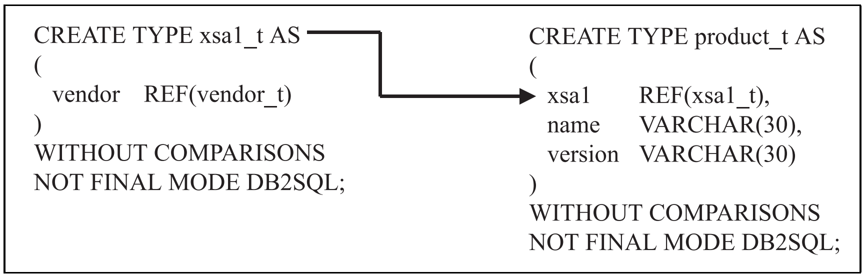
Die Resultate der o.g. Schritte kann man in den Abbildungen 13 und 14 nachvollziehen. Die Pfeile in den Abbildungen 13 und 14 geben die *scope*-Beziehung wieder.
Beispiel-DTD zur Erläuterung der Modellierung des *set-of* Typkonstruktors:

```
<!ELEMENT xsa1 (product,vendor)>
```

```
<!ELEMENT vendor   (name)>
<!ELEMENT name     (#PCDATA)>
<!ELEMENT product  (name|version)*>
<!ELEMENT version  (#PCDATA)>
```

Rückgabestring des Mapping Moduls:

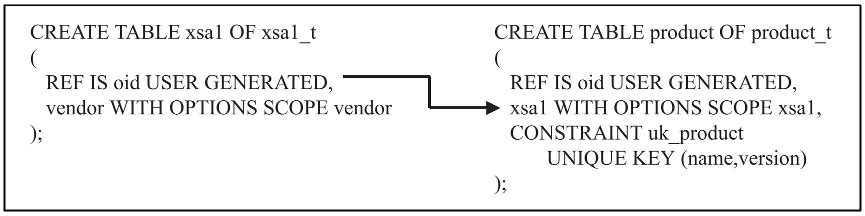
```
xsa1: <product: <product: {name: str , version: str }>,
       vendor: <name: str >>
```



```
CREATE TYPE xsa1_t AS
(
  vendor   REF(vendor_t)
)
WITHOUT COMPARISONS
NOT FINAL MODE DB2SQL;
```

```
CREATE TYPE product_t AS
(
  xsa1     REF(xsa1_t),
  name     VARCHAR(30),
  version  VARCHAR(30)
)
WITHOUT COMPARISONS
NOT FINAL MODE DB2SQL;
```

Abbildung 13: Modellierung des *set-of* Typkonstruktors in der Typdefinition



```
CREATE TABLE xsa1 OF xsa1_t
(
  REF IS oid USER GENERATED,
  vendor WITH OPTIONS SCOPE vendor
);
```

```
CREATE TABLE product OF product_t
(
  REF IS oid USER GENERATED,
  xsa1 WITH OPTIONS SCOPE xsa1,
  CONSTRAINT uk_product
      UNIQUE KEY (name,version)
);
```

Abbildung 14: Modellierung des *set-of* Typkonstruktors+– in der Tabellendefinition

5.4.2 Allgemeiner Programmablauf des Datenbankentwurfs

Um die Modellierung des Abschnitts 5.4.1 durchführen zu können, ist es notwendig, den String (Ergebnis des Abbildungsprozesses) nicht nur einmal, sondern zweimal zu analysieren. Dies ist notwendig, da im ersten Durchlauf Informationen gesammelt werden, die den zweiten Durchlauf steuern.

In der ersten Analyse werden Informationen über Elementebeziehungen gesammelt. Es werden z.B. die Informationen gesammelt, welches Element mit einem anderen Element eine *scope*-Beziehung eingeht oder welches Element mit einem anderen Element eine Listen- oder Set-Beziehung eingeht. Zusätzlich werden alle Elemente in einer Liste t_sequ gespeichert, für die eine Typdefinition erstellt wird. Die Liste t_sequ ist von großer Bedeutung für die Reihenfolge der späteren Ausgabe der Typ- und Tabellendefinitionen.

Aus diesem Grund werden vor dem Start der zweiten Analyse die, im ersten Durchlauf des Ergebnisstrings, gesammelten *scope*-Beziehungen analysiert. Auftretende gegenseitige Referenzierungen werden erkannt und durch Löschen der *scope*-Beziehung und Erstellen der *ALTER TYPE* und *ALTER TABLE*- Definitionen verhindert. Des weiteren werden durch die Analyse der *scope*-Beziehungen die Elemente der Liste t_sequ so sortiert, dass die Ausgabe der Typdefinition in umgekehrter Reihenfolge zur Referenzierung der Elemente untereinander erfolgt. Das heißt, wird in Typ A Typ B referenziert, so muss Typ B vor Typ A definiert werden.

Nun kann im zweiten Durchlauf des Ergebnisstrings des Abbildungsprozesses die Typdefinition, unter Berücksichtigung der in der ersten Analyse gesammelten Informationen und der veränderten *scope*-Beziehungen, erstellt werden. Dabei wird für jede Typdefinition ein String erstellt, der die einzelnen Attributdeklarationen des Typs enthält.

Für die Erstellung der Tabellendefinition sind bereits alle Vorarbeiten geleistet, teilweise im Abbildungsprozess und teilweise bei der Typdefinition. Die Tabellendefinition und Erstellung unterscheidet sich von der Typdefinitionserstellung darin, dass anstelle des String der Typ Definition nun Dictionaries verwendet werden. Das bedeutet, dass in einem übergeordneten Dictionary die Tabellennamen die Schlüsselwerte darstellen und die dazugehörigen Werte wiederum Dictionaries sind, die die Tabellenattribute als Schlüssel und die jeweiligen Attributdefinitionen als Werte enthalten. Abbildung 15 stellt ein Dictionary für das folgende Beispiel dar.

```
<!ELEMENT xsa1 (product*,vendor)>
<!ELEMENT vendor   (name)>
<!ELEMENT name     (#PCDATA)>
<!ELEMENT product  (name,version)>
<!ELEMENT version  (#PCDATA)>
```

```
{'product_1_XX': {'product':'product WITH OPTIONS SCOPE product',
                  'xsa1': 'xsa1 WITH OPTIONS SCOPE xsa1'},
          'xsa1': {'vendor': 'vendor WITH OPTIONS SCOPE vendor'},
        'vendor': {'name': 'name WITH OPTIONS NOT NULL'},
       'product': {'version': 'version WITH OPTIONS NOT NULL',
                   'name': 'name WITH OPTIONS NOT NULL'}
```

Abbildung 15: Dictionary der Tabellenattributdefinitionen

Zum Schluss werden die erstellten Typdefinitionen und Tabellendefinitionen in die Datei db2_syntax.txt geschrieben.

5.5 Das Metadatenmodul

Metadaten stellen eine wichtige Information für den Einleseprozess der XML-Daten in die Datenbank, fr das Anfragen von XML-Daten sowie fr weiter An-

wendungen dar.
Die Metadaten enthalten die Abbildungsvorschriften für alle XML Elemente und Attribute bezüglich der Datenbanktabellen und ihrer Datenbankattribute.
Implementiert wurden die Metadaten durch eine Klasse meta_Type (Abbildung 16). Wie auch schon für die Statistik wird für jedes Element und Attribut der DTD eine Instanz der Klasse meta_Type generiert. Die Generierung der Instanz erfolgt während des Abbildungsprozesses im Mapping Modul, da dort bereits die ersten Informationen, wie Name, Type und Vorgänger gesammelt werden. Die noch fehlenden Informationen der Metadaten (Art der Abbildung, Tabelle und Spalte) werden dann im zweiten Durchlauf der Analyse des Ergebnisstrings des Abbildungsprozesses im Datenbankentwurfsmodul vervollständigt. Wie bereits im Abschnitt 5.4.1 angedeutet, können auch während des zweiten Durchlaufs des Datenbankentwurfsmoduls noch neue Instanzen der Klasse meta_Type generiert werden.
Am Ende werden die erstellten Metadaten in die Datei db2_meta.txt geschrieben und so dem Anwender verfügbar gemacht.

```
class  meta_Type
  def __init__( self, name, element,pred):
            self.name = name
            self.element = element
            self.pred = pred
            self.table = ""
            self.art = ""
            self.type = ""
            self.col = ""
            d_mType[name]= self
```

Abbildung 16: Klassendefinition in Python

6 Programmaufruf

Des Programm kann mit den folgenden Parameteroptionen aufgerufen werde:

```
dtd2db.py dtd  [-v] [-l] {-o tagname}[-q Queryfile/Verzeichnis]
               {-s Tagname}{-x Tagname} [XML-Daten]
        dtd Pfadangabe der DTD
        -v  Anzeige der ermittelten Daten
        -l  nur Durchfuehren und Anzeigen der Statistik
        -o  nur dieses Element verarbeiten
        -q  Pfadangabe der Anfragedatei
        -s  Element mit vollstaendiger Pfadangabe,
            das auf Struktur gemapped werden soll
        -x  Element mit vollstaendiger Pfadangabe,
            das auf XML gemapped werden soll
  XML-Daten Pfadangabe oder XML-Datei
            (dieser Parameter muss als letzter eingegeben werden)
```

7 Zusammenfassung

Für die Implementierung existieren einige Einschränkungen, die der Anwender berücksichtigen muss. Die Einschränkungen, die in den folgenden Unterabschnitten genannt werden, beziehen sich auf die DTD und auf die Vorgabe einer expliziten Abbildungsvorschrift eines Elementes durch den Anwender.

7.1 DTD-Einschränkungen

Die Implementierung ist nicht für alle DTDs zugelassen. Deshalb existiert eine DTD Einschränkung, die folgende DTD als nicht zugelassen definiert:

```
1. <!ELEMENT a ( b , c , b+)>
```

Doppeldefinitionen werden nicht berücksichtigt. Der allgemeinste Quantor des doppelt definierten Elementes wird für die weitere Verarbeitung verwendet.

7.2 User Prioritäten

Fordert der Anwender eine strukturierte Abbildung eines Blattelementes des DTD-Graphen und setzt die User-Limit-Vorgabe gleichzeitig auf 1,macht der Benutzer also widersprüchliche Angaben,so wird das Blattelement dennoch auf den Datentyp XML abgebildet.

7.3 Aufgabenumsetzung

Ziel der Implementierung war es, den Algorithmus zu validieren und falls erforderlich zu ändern. Bis auf die Änderung der Formel für die Berechnung der Statistik über XML-Daten[9] waren keine anderen Änderungen der Statistiken notwendig.
Der Datenbankenwurf wurde um das Modellieren von Listen und Sets in DB2 erweitert. Gefordert war auch eine Interaktion zwischen dem Benutzer und dem Programm zu konzipieren und zu implementieren. Die Interaktion soll eine Beeinflussung des Abbildungsprozesses durch den Benutzer erlauben. Diese Interaktion wurde durch die Implementierung User Prioritäten umgesetzt.
Durch Variieren der User-Limit-Angabe und die Vorgabe expliziter Abbildungsvorschriften besitzt der Benutzer zwei leistungsstarke Kontrollmechanismen um den Abbildungsprozess zu beeinflussen. Die Abbildung 17 stellt den Kreislauf dar, der durch die Interaktion zwischen dem Benutzer und dem Programm entsteht.

8 Ausblick

Dieses Studienarbeit ist in keinem Projekt konkret eingebunden. Man könnte das Verfahren als allgemeine Methode zur Speicherung von XML-Dokumenten, für

---

[9] Diese Änderung basierte auf der detaillierten Zählung der Elemente und Attribute in den XML-Daten.

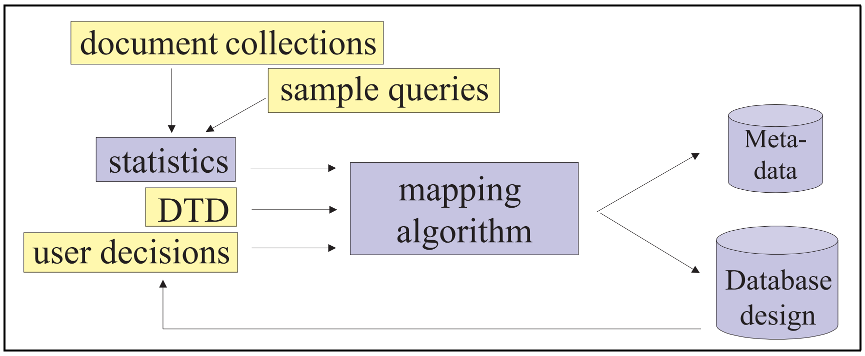


Abbildung 17: Kreislauf der Interaktion

die es vielfältige Einsatzmöglichkeiten gibt, beschreiben. Weiterführende Studienarbeiten befassen sich mit dem Einleseprozess der XML-Daten in die Datenbank, sowie dem Anfragen der gespeicherten Daten unter Ausnutzung der Metainformation. Nach Beendigung dieser Studienarbeiten hätte man dann ein umfangreiches Werkzeug zur Verarbeitung von XML-Daten in objektrelationalen Datenbanken zur Verfügung.