
XML und Datenbanken

Seminar 01912

Sommersemester 2002
Fernuniversität Hagen
LG Praktische Informatik IV

XML-QL

Version 1.25

basierend auf

XML-QL: A query language for XML

(A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu)

Thorsten Rood
Mat.Nr. 5.010.578
<mailto:thorsten@rood.cc>

Inhaltsverzeichnis

1	Einführung	1
1.1	Grundlagen	1
1.1.1	Einleitung	1
1.1.2	Vergleich RDBMS – XML.....	1
1.1.3	Einordnung von XML-QL und Einsatzmöglichkeiten	1
1.1.4	Features.....	2
1.2	Datenmodell	2
1.2.1	Ungeordneter Graph	2
1.2.2	Geordneter Graph.....	3
1.2.3	Navigation mit IDREF.....	3
2	Sprachumfang	5
2.1	Grundfunktionen	5
2.1.1	Syntax.....	5
2.1.2	Einfache Extraktion von Informationen	5
2.1.3	Strukturerhaltende Abfragen.....	6
2.1.4	Modifikation der Dokumentstruktur	7
2.1.5	Zusammenführung von Datenquellen	7
2.2	Geordnete Datenstrukturen.....	8
2.2.1	Indizierung.....	8
2.2.2	Sortierung	10
2.3	„Unscharfe“ Abfragen.....	10
2.3.1	Unbekannte Elementbezeichner.....	10
2.3.2	Auswertung von Alternativen	11
2.3.3	Hierarchieauswertung durch reguläre Ausdrücke	12
2.4	Sonstiges.....	13
2.4.1	Skolem-Funktion.....	13
2.4.2	Aggregation	14
2.5	Offene Punkte.....	15
2.5.1	XML-Spezialitäten	15
2.5.2	Modellierungsprobleme	16
3	Fazit.....	17
4	Anhang.....	I
4.1	XML-QL Grammatik	I
4.2	Quellen.....	II

Abbildungsverzeichnis

Abbildung 1: ungeordneter Graph	2
Abbildung 2: geordneter Graph.....	3
Abbildung 3: „pointer-chasing“ durch IDRREF(S)	4
Abbildung 4: Modellierung von XML-Grenzfällen	16

Beispiele mit XML-QL

Beispiel 1: Query als einfache Informationsextraktion	6
Beispiel 2: Query als Extraktion mit Strukturhaltung	6
Beispiel 3: Query mit Strukturmodifikation	7
Beispiel 4: Query „inner join“	8
Beispiel 5: Query mit Indizierung von XML-Tags und -attributen	9
Beispiel 6: Query mit Sortierung der Ergebnisliste.....	10
Beispiel 7: Query über flexible XML-Tags.....	11
Beispiel 8: Query über optionale XML-Tags	12
Beispiel 9: Query mit regulären Ausdrücken	13
Beispiel 10: Query „outer join“	14
Beispiel 11: Query mit Aggregation durch eingebettete Funktionen.....	15

Übersichten

Übersicht 1: Beispieldaten „studentenbelegung“	III
Übersicht 2: Beispieldaten „kursangebot“	IV
Übersicht 3: Beispieldaten „virtuelle_uni_kursangebot“	IV

1 Einführung

1.1 Grundlagen

1.1.1 Einleitung

Die zunehmende Präsenz des Internet (oder zumindest von Internettechnologien) führt zu einer Verschiebung von traditionellen IT-Schwerpunkten, der Ruf nach Interoperabilität, sowie die Schaffung und Wahrung von Standards sind bedeutende Themen geworden.

Wurde anfangs das Aufkommen von XML als Transport- und Integrationsformat für beliebige Informationen eher als Marketingargument missverstanden, so ist heute bei allen namhaften Herstellern und Unternehmen eine relativ breite Unterstützung (oder wenigstens ein starkes Interesse) für die Verarbeitung von XML-Informationen zu erkennen.

Nun werden leistungsstarke Methoden notwendig, welche XML-Dokumente verarbeiten, zusammenführen und modifizieren können – und das Ergebnis wiederum als XML bereit stellen. Gemeint ist in diesem Zusammenhang nicht die granulare Analyse einzelner Fragmente, sondern die Behandlung großer Datenmengen, also eine datenbankorientierte Sichtweise (EDI = *electronic data interchange*).

1.1.2 Vergleich RDBMS – XML

Grundlage „klassischer“ DBMS (relationales Modell) sind Relationen auf Basis eines relativ starren Datenbankschemas. In einzelnen Tabellen(zeilen) sind Werte hinterlegt und der logische Gesamtzusammenhang wird über Primär- und Fremdschlüssel gewahrt. Selbst unter Berücksichtigung von potentiell nicht belegten Informationen (zulässige „Null“-Felder) sind solche Informationstupel sehr homogen strukturiert.

XML-Dokumente sind dagegen selbstbeschreibend, zeichnen sich durch eine flexible Struktur aus¹ und sind selbst bei Forderung einer expliziten DTD – als „Datenschema“ – deutlich komplexer in Ihrer Struktur und tendenziell heterogen. Die hier mögliche Modellierung von Unregelmäßigkeiten ist in RDBMS nur unter hoher Kraftanstrengung möglich. Deshalb werden XML-Dokumente auch *semi-strukturierte Daten* genannt².

1.1.3 Einordnung von XML-QL und Einsatzmöglichkeiten

XML-QL ist eine Abfragesprache für XML-Dokumente: Einer XML-QL-Engine, welche das gesamte Befehlsspektrum implementiert, erlaubt dies, XML-Daten aus verschiedenen Quellen gleichzeitig zu verarbeiten, zusammenzuführen, neu zu strukturieren und/oder zu filtern und das Ergebnis wiederum im XML-Format abzuliefern. Maßgeblicher „Erfinder“ ist eine Forschergruppe um *Alin Deutsch*.

Es handelt sich also um eine Middleware-Technologie, welche überall dort eingesetzt werden könnte, wo das Datenvolumen eine manuelle Verarbeitung generell ausschließt und wo eine spezifische Zusammenführung der Daten in der Zielanwendung – auf Basis anderer Techniken wie XSL und XPath – deutlich zu kompliziert wäre³. Es wird ein datenbankorientierter Ansatz verfolgt, welcher auf die Besonderheiten semi-strukturierter Daten eingeht.

XML-QL kann (noch) nicht als offiziell anerkannter Standard angesehen werden, es existiert bisher „nur“ ein informeller Spezifikationsvorschlag⁴ beim W3C.

¹ Diese „intuitive“ Form der Datenrepräsentation war mit ausschlaggebend für die Akzeptanz...

² weitere Detaillierungen hierzu finden sich in [5]

³ beispielsweise in [7]

⁴ „note“ – <http://www.w3.org/Consortium/Process>

1.1.4 Features

XML-QL vermag derzeit folgendes zu leisten⁵:

- Die Syntax ist wohlgeformt; Queries können also „native“ in XML-Daten eingebettet und transportiert werden – gleiches gilt auch für die Ausgabe
- Unterstützt XML-Unregelmäßigkeiten bei gewohnter Syntax aus Muster/Filter und Konstruktor
- Ermöglicht die Restrukturierung von XML-Dokumenten durch Schachtelung, Gruppierung, Indizierung, Sortierung und Auswertung von optionalen TAGs.
- Unterstützung verschiedener Join-Methoden
- Unterstützung für komplexe XML-Daten (Schachtelung in unbekannter Tiefe, fehlende DTD) durch TAG-Analyse und reguläre Ausdrücke
- Intuitive Navigation (Auswertung von IDREFs)

1.2 Datenmodell

1.2.1 Ungeordneter Graph

Zur Abbildung und Analyse der Ein- und Ausgabedaten bedient sich XML-QL eines *XML-Graphen*, der formal wie folgt definiert ist:

- Graph $G = (V, E)$ mit ausgewiesenem Root-Element
- Kanten $\in E$ mit Elementbezeichnern (TAG-Name)
- Knoten $\in V$ mit eindeutigen Bezeichnern *object identifier* (OID) und optional zugeordneten Attribut-Wert-Paaren, sofern vorhanden
- alle Blätter sind mit #PCDATA-Inhalt belegt
- Attribute des XML-Dokuments entsprechen Knoten, Elemente den Kanten, Elementinhalt entspricht den Blättern

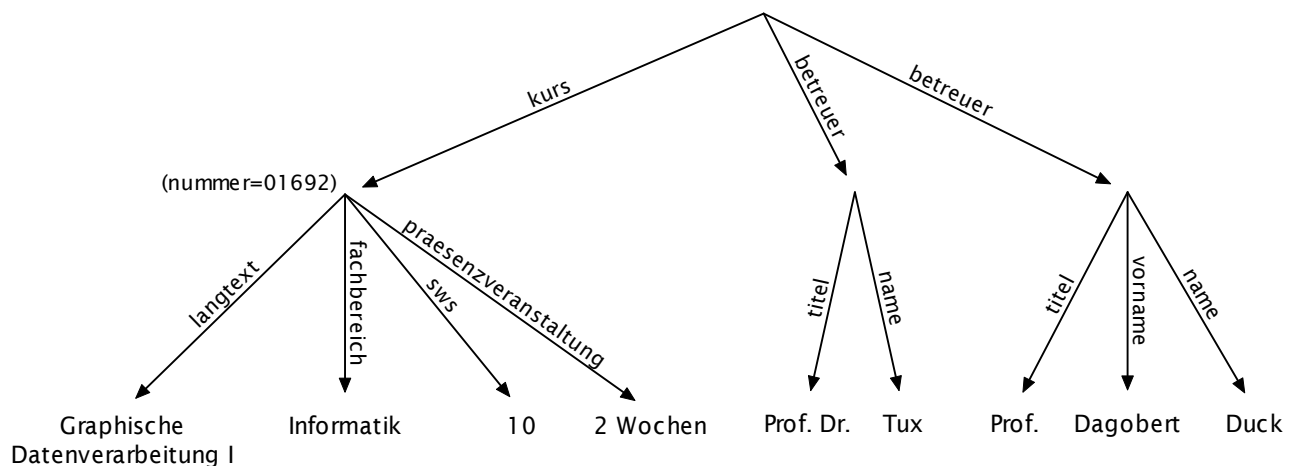


Abbildung 1: ungeordneter Graph⁶

Zusätzlich gilt im ungeordneten Modell:

- zwischen zwei Knoten gibt es höchstens eine Kante mit einer bestimmten Bezeichnung

⁵ der unter [11] verfügbare Prototyp kann noch nicht alle diese Optionen anbieten

⁶ OIDs wurden nicht notiert

- kein Knoten besitzt mehr als ein Blatt mit der gleichen Bezeichnung und Inhalt
- Die relative Reihenfolge von Elementen wird bei den Muster-/Filter-Vergleichen nicht berücksichtigt. Ein Ordnungskriterium ist also nicht abgebildet.

1.2.2 Geordneter Graph

In der dokumenten-basierten Sichtweise ist die absolute und relative Ordnung der XML-Informationen relevant für die Analyse und Aufbereitung der Daten. XML-QL unterstützt dies auf Basis eines geordneten XML-Graphen, sobald eine Query mit enthaltenen Indexvariablen (Kapitel 2.2.1) ausgeführt werden soll, d.h. es findet ein automatisches „Fallback“ statt.

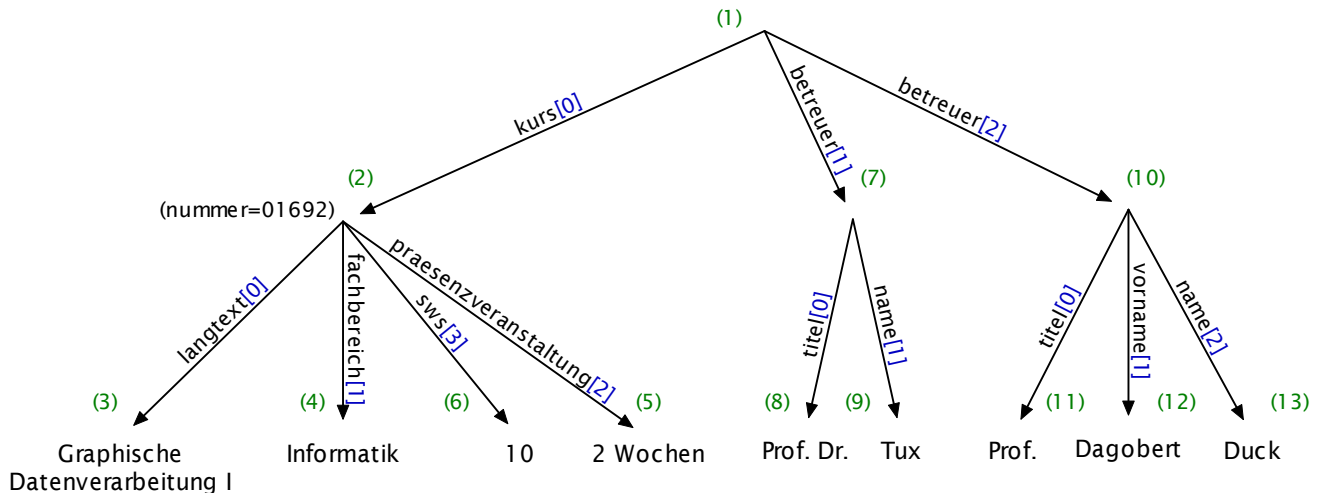


Abbildung 2: geordneter Graph⁷

Die Position der Elemente aus dem Preorder-Durchlauf ist dann auswertbar, gleichzeitig entfallen aber viele Möglichkeiten der Query-Optimization, was bei großen Datenmengen zwangsläufig zu Lasten der Ausführungsgeschwindigkeit gehen wird.

1.2.3 Navigation mit IDREF

Zur Vermeidung redundanter Daten unterstützt XML ein Konzept aus Primär- und Fremdschlüssel in Anlehnung an RDBMS-Konzepte. Ein Element kann hierbei ein Pflichtattribut des Typs *ID* enthalten und diesen ausgewiesenen Schlüssel als Referenz *IDREF* bzw. *IDREFS* in andere Elemente einbringen⁸.

Beispiel:

```
<!ATTLIST betreuer id ID #REQUIRED>
<!ATTLIST kurs betreuer IDREFS #REQUIRED>
```

Im obigen Fall können Personen mit ihren abhängigen Daten nun einmalig erfasst und dann „kostengünstig“ ihren Kursen zugeordnet werden.

XML-QL unterstützt diese Querverweise auch im XML-Graphen, sofern die DTD bekannt ist. Es entsteht jetzt der Sonderfall, dass ein Attribut als **Kante** ausgeprägt wird. Außerdem nimmt die ID die OID des betroffenen Knotens an.

⁷ Die Reihenfolge der TAGs <praesenzveranstaltung> und <sws> wurde aus Platz- bzw. Darstellungsgründen vertauscht, die geordnete Nummerierung ist richtig.

⁸ Mit Details hierzu befasst sich das Thema *document type definition* (DTD), eine Einführung findet sich beispielsweise in [12].

Eine zulässige Navigation im XML-Graphen (und damit im Quelldokument) lautet nun auch

`<kurs><betreuer><name>`

so, wie es das XML-Dokument auch informell bereits nahe legt:

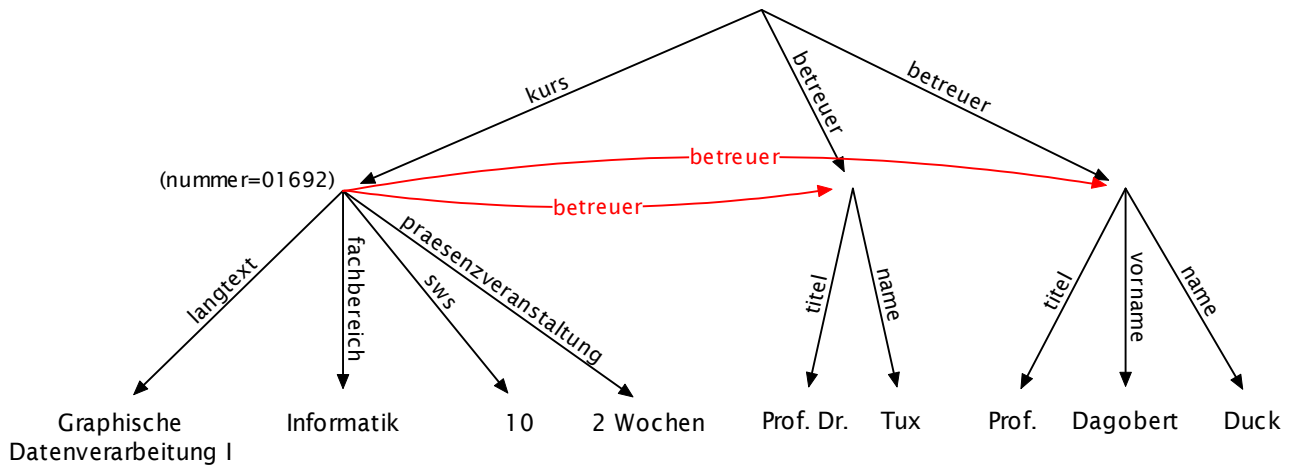


Abbildung 3: „pointer-chasing“ durch IDREF(S)

2 Sprachumfang

2.1 Grundfunktionen

2.1.1 Syntax

XML-QL hat einen syntaktischen Aufbau in Anlehnung an SQL-/OQL-Abfragesprachen, unter Berücksichtigung des Umstandes, dass für semi-strukturierte Daten weniger formale Rahmenbedingungen gelten. Jede Abfrage ist formell zu XML konform und hat (prinzipiell) die Struktur:

```
WHERE P
IN S
CONSTRUCT C
```

Dabei kennzeichnet P das in den Quelldaten S zu erkennende *Grundmuster*, in welchem Inhalte an Variablen gebunden werden. Diese wiederum können im *Konstruktionsteil C* eingesetzt und/oder weiter ausgewertet⁹ werden. Es wird also eine Schablone P mit enthaltenen Variablen $P(x_1, \dots, x_k)$ über das Quelldokument S gelegt, diese für **jeden** einzelnen „Treffer“ belegt und im Konstruktionsteil C zur Verfügung gestellt. Jede „Konstruktion“ entspricht dem klassischen „Treffer“ in Datenbanktabellen einer SQL-basierten Abfrage. Eine Filter-Klausel ist sowohl im Grundmuster, als auch (optional) in der IN-Klausel enthalten. P und C sind üblicherweise Fragmente von XML-TAGs.

Als Antwortdokument einer XML-QL-Engine wird dann die Konkatenation aller C geliefert, wobei das umgebende Wurzelement <XML> automatisch generiert wird¹⁰.

Das Grundmuster P entspricht einem XML-Fragment mit eingebetteten Platzhaltern. Variablen können dabei an

- String-Werte von Elementen (#PCDATA): <langtext>\$bezeichnung</langtext>
- String-Werte von Attributen (#CDATA): <diplomarbeit titel=\$name>...</diplomarbeit>
- Interne Knoten (OID): <kurs>\$k</kurs>¹¹
- Elemente (TAG): <\$e>

gebunden werden und werden mit \$ identifiziert¹². Für die weitere Erläuterung der Funktionalitäten wird in Anlehnung an den Originalentwurf ([1]) anhand von Beispielen der Sprachgebrauch detailliert.

Dabei beziehen sich alle Queries auf die im Anhang unter „4.2 Beispieldaten“ zur Verfügung gestellten XML-Datensätze.

2.1.2 Einfache Extraktion von Informationen

Das Grundmuster P darf sowohl statische, als auch dynamische Informationen enthalten.

XML-QL unterstützt ferner die syntaktische Vereinfachung, dass schließende Elementbezeichner mit </> abgekürzt werden dürfen. Auf diese Weise sind auch reguläre Ausdrücke über mehrere Elementebenen hinweg (vgl. 2.3.3) syntaktisch korrekt¹³, d.h. die XML-Engine ergänzt bei der Ausgabe eventuell fehlende Ebenen dann automatisch.

⁹ Verschachtelte Abfragen, vgl. ab 2.1.3

¹⁰ Auf die benutzerdefinierte Bezeichnung/Umbenennung der Wurzel wird nicht weiter eingegangen.

¹¹ \$k enthält also „XML-Content“

¹² Die Grammatik unter 4.1 schweigt sich hierzu aus, die Dokumentationen unter [1] und [11] sind ergänzend zu berücksichtigen.

¹³ Zur Erinnerung: XML-QL-Abfragen können in XML-Dokumente eingebettet werden und müssen ihrerseits deshalb den XML-Spezifikationen entsprechen! Natürlich muss sichergestellt sein, dass verwendete XML-Parser die Vereinfachung des Abschluss-TAGs </> akzeptieren.

Im vorliegenden Beispiel werden die #PCDATA an die Variablen \$l und \$n gebunden und diese tupelweise wieder ausgegeben. Die automatische Navigation über IDREF wird aus Kapitel 1.2.3 in der Form <kurs><betreuer><name> aufgegriffen.

„Finde alle Tupel (Informatik-Kursangebot,Lehrkraft)“	
<pre>[XML-QL] Where <kursangebot> <kurs> <langtext>\$l</> <fachbereich>Informatik</> <betreuer><name>\$n</></> </> </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <informatikkurs> <langtext>\$l</> <betreuer>\$n</> </></pre>	<pre>[Abfrageergebnis] <?xml version="1.0"?> <XML ID="out.genoid_0"> <informatikkurs> <langtext>Graphische Datenverarbeitung I</langtext> <betreuer>Duck</betreuer> </informatikkurs> <informatikkurs> <langtext>Graphische Datenverarbeitung I</langtext> <betreuer>Tux</betreuer> </informatikkurs> </XML></pre>

Beispiel 1: Query als einfache Informationsextraktion

Generell ist zu beachten, dass alle Elemente des Grundmusters **enthalten** sein müssen – ein XML-Fragment ohne das Element <langtext> würde hier also nicht positiv validiert und nicht in die Konstruktion aufgenommen. XML-QL unterscheidet auf diese Weise zwischen der Variablenbelegung *Null* und einem leeren *Inhalt*.

Die ursprüngliche Dokumentstruktur geht bei dieser Art der Abfrage verloren. Man spricht hier vom *unnesting* ([1]) bzw. *flattening* ([3]), d.h. ursprünglich logisch gruppierte XML-Informationen werden „voll ausmultipliziert“. Dieser Effekt ist **beabsichtigt** und ist gleichzeitig auch technische Grundlage, die Ausgabe überhaupt **neu strukturieren zu können** [vgl. [3] Kap. 2.2]!

2.1.3 Strukturerehaltende Abfragen

Aufbauend auf dem ersten Beispiel soll nun die Gruppierung der Betreuernamen beim jeweiligen Kursangebot erhalten bleiben: Es ist nötig, Inhalt aus der Schablone weiter zu interpretieren.

„Gruppierere alle Lehrkräfte beim jeweiligen Informatik-Kursangebot“	
<pre>[XML-QL] Where <kursangebot> <kurs> <langtext>\$l</> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <informatikkurs> <langtext>\$l</> { where <kurs><betreuer><name>\$n</></></> in \$k construct <betreuer>\$n</> } </></pre>	<pre>[Abfrageergebnis] <?xml version="1.0"?> <XML ID="out.genoid_0"> <informatikkurs> <langtext>Graphische Datenverarbeitung I</langtext> <betreuer>Duck</betreuer> <betreuer>Tux</betreuer> </informatikkurs> </XML></pre>

Beispiel 2: Query als Extraktion mit Strukturerehaltung

Mit dem Schlüsselwort *ELEMENT_AS* wird eine weitere Möglichkeit eingeführt, Elementinhalt an eine Variable zu binden¹⁴ und zusätzlich wird der sogenannte „verschachtelte Query-Block“ gezeigt. Die IN-Klausel ist nicht nur auf externe Datenquellen beschränkt, sondern darf ebenfalls die Variablenbindung nutzen.

Das zweite WHERE-CONSTRUCT ist demnach vergleichbar mit einer inneren Schleife.

Im Zuge der Auswertung wird der Betreuername nun nicht mehr zerrissen, sondern beide Elemente in der Variable \$k bis zum zweiten WHERE-CONSTRUCT konserviert und dort ausgelesen.

Alternativ kann die Gruppierung auch über die später in Kapitel 2.4.1 vorgestellte *Skolem-Funktion* erzwungen werden, jedoch erzeugt die hier vorgestellte Query-Variante die Ausgabe kanonisch¹⁵.

2.1.4 Modifikation der Dokumentstruktur

Bei Verschachtelung von Abfragen müssen nicht zwingend ganze Elemente miteinander verknüpft werden, man kann auch ein- und dieselbe Datenquelle lose verknüpfen, also auf diese Weise einen „join“-Verbund erstellen:

„Gruppieren alle Kurse beim jeweiligen Betreuer“	
[XML-QL]	[Abfrageergebnis]
<pre> where <kursangebot> <betreuer ID=\$b> <name>\$n</> </> </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <betreuer> <name>\$n</> { where <kursangebot> <kurs betreuer=\$b> <langtext>\$k</> </> </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <kurs>\$k</> } } </> </pre>	<pre> <?xml version="1.0"?> <XML ID="out.genoid_0"> <betreuer> <name>Duesentrieb</name> <kurs>Unternehmensfuehrung</kurs> </betreuer> <betreuer> <name>Duck</name> <kurs>Graphische Datenverarbeitung I</kurs> </betreuer> <betreuer> <name>Tux</name> <kurs>Graphische Datenverarbeitung I</kurs> </betreuer> </XML> </pre>

Beispiel 3: Query mit Strukturmodifikation

Nun entsteht eine zum Ursprungsdokument „verdrehte“ Ansicht: Aus der Zuordnung *Kurs* ⇔ *Betreuer* wird die neue View *Betreuer* ⇔ *Kurs*.

Bei Einsatz von XML-QL als Konverterplattform dürfte die Erzeugung einer kompatiblen Sicht auf ähnliche Daten aus völlig verschiedenen Quellen ein recht typischer Anwendungsfall sein.

2.1.5 Zusammenführung von Datenquellen

Aus Sicht der XML-QL-Engine ist es kein **syntaktischer** Unterschied, ob der obige Join auf dieselbe Datenquelle oder auf verschiedene XML-Dokumente durchzuführen ist (aus **technischer** Sicht ist es

¹⁴ Die Möglichkeit, mit dem Schlüsselwort *CONTENT_AS* zu arbeiten, wird nicht weiter ausgeführt und auf andere Ausarbeitungen zu diesem Thema wie [3] und [5] verwiesen.

¹⁵ Das interne Optimierungspotenzial der XML-QL-Engine dürfte bei expliziter Gruppierung über geschachtelte Query-Blöcke höher ausfallen als bei Skolem-Funktionen.

durchaus interessant, z.B. Performance-Optimierungen, Pufferung, etc.), so dass ein „echter“ inner-join nach dem gleichen Anfrageschema abläuft.

„Kursbelegungsliste“	
<pre>[XML-QL] where <kursangebot> <kurs nummer=\$n> <langtext>\$l</> </> </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <kurs> <name>\$l</> { where <gesamtbelegung> <student> <kurs>\$n</> <name>\$s</> </> </> in "http://www.rood.cc/download/ 01912/studentenbelegung.xml" construct <student>\$s</> } </></pre>	<pre>[Abfrageergebnis] <?xml version="1.0"?> <XML ID="out.genoid_0"> <kurs> <name>Unternehmensfuehrung</name> <student>Thorsten Rood</student> <student>Daisy Duck</student> </kurs> <kurs> <name>Graphische Datenverarbeitung I</name> <student>Thorsten Rood</student> <student>Fritz Mustermann</student> </kurs> </XML></pre>

Beispiel 4: Query „inner join“

Es ist auch möglich, die WHERE-Schablone der geschachtelten Abfrage „nach oben zu ziehen“ und beide Teile durch Komma getrennt zu positionieren¹⁶. Um den Effekt des *flattening* zu vermeiden, wäre dann aber wiederum eine *Skolem-Funktion* bezüglich des <kurs>-Elements notwendig.

2.2 Geordnete Datenstrukturen

2.2.1 Indizierung

Bei der Auswertung semistrukturierter Daten kann es notwendig werden, eine Aussage über die Reihenfolge bestimmter Elemente zu treffen. Dies kann sich sowohl auf die Reihenfolge disjunkter TAGs, als auch auf die Abfolge namensgleicher TAGs in einem XML-Fragment beziehen¹⁷.

Um Zugriff auf diese Ordnungsinformationen zu erhalten, unterstützt XML-QL zusätzlich eine Variablenbindung, wie sie im geordneten Datenmodell (vgl. 1.2.2) gezeigt wird. Realisiert wird dies durch eine geklammerte Variable¹⁸, die an die zu analysierende XML-Information¹⁹ „angeklebt“ wird.

Die Zählweise ist in Abbildung 2 „geordneter Graph“ skizziert und zeigt in meinen Augen, dass der Einsatzbereich relativ begrenzt sein kann, da **alle** Elemente einer DOM-Ebene **gemeinsam** durchgezählt werden.

¹⁶ Vgl. 2.3.1

¹⁷ Verschiedene Szenarien hierzu sind in [1], [3] und [4] dargestellt.

¹⁸ Die Dokumentquellen [1] und [3], an denen der maßgeblich für XML-QL verantwortliche Autor Alin Deutsch in beiden Fällen mitgewirkt hat, weisen leider zueinander verschiedene syntaktische Implementierungen aus. Zudem unterstützt die öffentlich zugängliche XML-QL-Engine ([11]) Indexvariablen überhaupt nicht.

¹⁹ Es ist in der Literatur nicht genau spezifiziert, ob und wie hier sowohl TAGs, als auch Attribute „gezählt“ werden können/sollen. Insbesondere kann derzeit (noch) nicht das konkrete Verhalten einer voll ausprogrammierten XML-QL-Engine validiert werden, wenn Indizierung von XML-Daten durch „pointer-chasing“ (vgl. 1.2.3) gefordert ist. In meinen Beispieldaten wird dies als implementiert vorausgesetzt.

„Gruppieren Lehrkräfte und Vertretungspersonen beim jeweiligen Informatik-Kursangebot“	
<pre>[XML-QL] Where <kursangebot> <kurs> <langtext>\${ </> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <informatikkurs> <langtext>\${ </> { where <kurs><betreuer [Si]><name>\$n</></></> in \$k construct { where [Si]=0 construct <betreuer>\$n</> } where [Si]>0 construct <vertretung>\$n</> } } </></pre>	<pre>[Abfrageergebnis] <?xml version="1.0"?> <XML ID="out.genoid_0"> <informatikkurs> <langtext>Graphische Datenverarbeitung I</langtext> <betreuer>Duck</betreuer> <vertretung>Tux</vertretung> </informatikkurs> </XML></pre>

Beispiel 5: Query mit Indizierung von XML-Tags und -attributen

Hier sieht man auch, dass der Query-Block über mehrere Ebenen verschachtelt werden kann und parallele Abfragen möglich sind. Die Ergebnisse der innersten Blöcke werden konkateniert, das entweder-oder-Prinzip entsteht hier nur durch XOR-Formulierung der Selektionskriterien bezüglich der Indexvariablen „\$i“.

2.2.2 Sortierung

Gerade bei Neustrukturierung von Dokumenten kann es interessant sein, die Ausgabereihenfolge zu beeinflussen.

Zu diesem Zweck kann das WHERE-CONSTRUCT um das Schlüsselwort ORDERED-BY ergänzt werden²⁰, hinter dem dann ein oder mehrere Variablen gelistet werden können. Im Falle mehrerer Variablen ist die Sortierung dann hierarchisch zu verstehen.

„wie 2.2.1 und sortiere nach Kursbezeichnung“	
<p>[XML-QL]</p> <pre>Where <kursangebot> <kurs> <langtext>\${}</> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml" ordered-by \${} construct <informatikkurs> <langtext>\${}</> { where <kurs><betreuer [\${}]<name>\${n}</></></> in \$k construct { where \$i=0 construct <betreuer>\${n}</> } { where \$i>0 construct <vertretung>\${n}</> } } } </></pre>	<p>[Abfrageergebnis]</p> <pre><?xml version="1.0"?> <XML ID="out.genoid_0"> <informatikkurs> <langtext>Graphische Datenverarbeitung I</langtext> <betreuer>Duck</betreuer> <vertretung>Tux</vertretung> </informatikkurs> </XML></pre>

Beispiel 6: Query mit Sortierung der Ergebnisliste

2.3 „Unscharfe“ Abfragen

2.3.1 Unbekannte Elementbezeichner

Die vorhergehenden Queries leiden an einem gravierenden Nachteil: es werden semistrukturierte Daten analysiert, die auf gleicher DOM-Ebene disjunkte TAGs beinhalten, welche jedoch aus logischer Sicht gleichermaßen betroffen sind: <kurs> und <seminar>.

Die gleichzeitige Auswertung beider Elementklassen über ein „outer-join“ (vgl. 2.4.1) wäre zwar möglich²¹, jedoch dürfte sich eine alphabetische Sortierung der tief verschachtelten Informationen sehr schwierig gestalten.

Als Ausweg bietet XML-QL die Möglichkeit, Variablen auch auf die Elementbezeichner anzuwenden und den „IN“-Zweig des WHERE-CONSTRUCT mit entsprechenden Nebenbedingungen zu versehen. Die konkrete Belegung des TAG ließe sich auch im Konstruktionsteil zur Generierung desselben Elements verwenden, jedoch ebenso als „einfache“ String-Information wieder in die Ausgabe einflechten.

²⁰ Der Zusatz DESCENDING fehlt in der Grammatik, wird in den zugehörigen Einführungen jedoch ausdrücklich erwähnt.

²¹ Leider nur in engen Grenzen, da die beabsichtigte Semantik möglicherweise nicht voll abgebildet werden könnte.

„sortiertes Informatik-Lehrangebot und zugehörige Lehrkräfte“	
<p>[XML-QL]</p> <pre>Where <kursangebot> <\$a> <langtext>\${</> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml", \$a in { kurs, seminar } ordered-by \$l construct <informatiklehrveranstaltung> <langtext>\${</> <typ>\$a</> { where <\$a><betreuer [\$i]><name>\$n</></></> in \$k construct { where \$i=0 construct <betreuer>\$n</> } { where \$i>0 construct <vertretung>\$n</> } } } </></pre>	<p>[Abfrageergebnis]</p> <pre><?xml version="1.0"?> <XML ID="out.genoid_0"> <informatiklehrveranstaltung> <langtext>Graphische Datenverarbeitung I</langtext> <typ>kurs</typ> <betreuer>Duck</betreuer> <vertretung>Tux</vertretung> </informatiklehrveranstaltung> <informatiklehrveranstaltung> <langtext>XML und Datenbanken</langtext> <typ>seminar</typ> <betreuer>Tux</betreuer> </informatiklehrveranstaltung> </XML></pre>

Beispiel 7: Query über flexible XML-Tags

2.3.2 Auswertung von Alternativen

In den einführenden Beispielen²² wurde dargestellt, dass TAGs in der WHERE-Schablone zwingend **vorhanden** sein müssen. Im Grenzbereich von optionalen oder alternativen TAGs muss hier eine Auswertung über Umwege vorgenommen werden, indem das passende XML-Fragment „unscharf“ an eine Variable gebunden wird. Eine verschachtelte Abfrage kann diese Informationen dann detaillierter auswerten und Teilinformationen wieder der Ausgabe hinzufügen.

Mit der Handhabung der Indizierung wurde bereits die Möglichkeit der parallelen Query-Blöcke gezeigt. Da XML-QL keine IF-THEN-ELSE-Logik unterstützt, muss man diese in den parallelen Query-Blöcken anhand der Selektionskriterien emulieren²³.

²² Vgl. 2.1.2

²³ Es wird Grenzbereiche geben, in denen keine zueinander disjunkten Nebenbedingungen formuliert werden können. Die parallele Abarbeitung wird dann doppelte Ausgaben in den verschachtelten Query-Blöcken liefern.

„wie 2.3.1 und ergänze Zusatzinformationen, wenn vorhanden“	
<pre>[XML-QL] Where <kursangebot> <\$a> <langtext>\${ </> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml", \$a in { kurs, seminar } ordered-by \$l construct <informatiklehrveranstaltung> <langtext>\${ </> <typ>\$a</> { where <\$a><praesenzveranstaltung>\$p</></> in \$k construct <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! \$p</> } { where <\$a><betreuer [\$i]><name>\$n</></></> in \$k construct { where \$i=0 construct <betreuer>\$n</> } { where \$i>0 construct <vertretung>\$n</> } } } </></pre>	<pre>[Abfrageergebnis] <?xml version="1.0"?> <XML ID="out.genoid_0"> <informatiklehrveranstaltung> <langtext>Graphische Datenverarbeitung I</langtext> <typ>kurs</typ> <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! 2 Wochen (2x Mo-Fr) vor Ort vom 06.05.-17.05.2002</hinweis> <betreuer>Duck</betreuer> <vertretung>Tux</vertretung> </informatiklehrveranstaltung> <informatiklehrveranstaltung> <langtext>XML und Datenbanken</langtext> <typ>seminar</typ> <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! Seminarvortraege vor Ort vom 28.06.-29.06.2002</hinweis> <betreuer>Tux</betreuer> </informatiklehrveranstaltung> </XML></pre>

Beispiel 8: Query über optionale XML-Tags

2.3.3 Hierarchieauswertung durch reguläre Ausdrücke

XML-QL bietet hierzu die Möglichkeit, reguläre Ausdrücke in den Filtern und Variablenvergleichen zu benutzen. Im komplexesten Anwendungsspektrum liegen hier die *path expressions*, d.h. die Option, auch auf TAGs eine Selektion mittels regulärer Ausdrücke auszuführen²⁴.

Operator	Bedeutung
<code>a b</code>	a oder b
<code><expr1.expr2></code>	<code><expr1><expr2></code>
<code>expr*</code>	Kleene-Star, also 0 bis n-fache Wiederholung von <code>expr</code> -TAGs. Wenn <code>expr</code> leer ist, können beliebige TAGs gefunden und der Platzhalter <code>\$*</code> darf durch <code>*</code> vereinfacht werden.
<code>expr+</code>	n-fache Wiederholung von <code>expr</code> -TAGs mit <code>n>0</code>
<code>()</code>	Hierarchische Operatorschachtelung im „üblichen Sinne“

Einsatzzweck: Die Kombination aus unbekanntem Bezeichnern und verschachtelten optionalen TAGs kann zu schwer handhabbaren XML-Informationen führen, insbesondere, wenn diese aus verschiede-

²⁴ Ein sehr detailliertes Beispiel hierzu findet sich in [4].

nen Quellen stammen sollten (beispielsweise im Bereich EDI). Vorstellbar sind semantisch gleichwertige Informationen in syntaktisch nur ähnlicher Darstellung.

„wie 2.3.2, syntaktisch einfacher“	
[XML-QL]	[Abfrageergebnis]
<pre> Where <kursangebot> <*> <langtext>\${l}</> <fachbereich>Informatik</> </> ELEMENT_AS \$k </> in "http://www.rood.cc/download/01912/kursangebot.xml" ordered-by \$l construct <informatiklehrveranstaltung> <langtext>\${l}</> <typ>\${a}</> { where <*.praesenzveranstaltung>\${p}</> in \$k construct <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! \${p}</> } { where <*.betreuer [\${i}]><name>\${n}</></> in \$k construct { where \$i=0 construct <betreuer>\${n}</> } { where \$i>0 construct <vertretung>\${n}</> } } } </> </pre>	<pre> <?xml version="1.0"?> <XML ID="out.genoid_0"> <informatiklehrveranstaltung> <langtext>Graphische Datenverarbeitung I</langtext> <typ>kurs</typ> <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! 2 Wochen (2x Mo-Fr) vor Ort vom 06.05.-17.05.2002</hinweis> <betreuer>Duck</betreuer> <vertretung>Tux</vertretung> </informatiklehrveranstaltung> <informatiklehrveranstaltung> <langtext>XML und Datenbanken</langtext> <typ>seminar</typ> <hinweis>Dieses Lehrangebot beinhaltet eine Praesenzveranstaltung! Seminarvortraege vor Ort vom 28.06.-29.06.2002</hinweis> <betreuer>Tux</betreuer> </informatiklehrveranstaltung> </XML> </pre>

Beispiel 9: Query mit regulären Ausdrücken

Die Dokumentstruktur wird durch path expressions abstrahiert, es fällt aber auch die Möglichkeit weg, die konkreten Elementbezeichner später in ihrer originären Bezeichnung zu referenzieren.

Gleichzeitig bietet diese Vorgehensweise aber auch den Vorteil, dass bei Einführung neuer TAGs auf der durch path expressions behandelten Ebene die erstellte Abfrage gültig bleibt und sogar die neuen Datensätze vollständig berücksichtigt werden.

2.4 Sonstiges

2.4.1 Skolem-Funktion

Die Zusammenführung verschiedener Quellen kann Datenkonstellationen hervorbringen, die im SQL als „outer-join“ verarbeitet werden, also die Vereinigung von zwei oder mehr XML-Dokumenten zu einer Gesamtinformation. Natürlich wird man auch hier Interesse an einer strukturierten Aufbereitung der Ausgabe haben.

XML-QL bietet zur automatischen Gruppierung identischer Tupel aus verschiedenen Datenquellen die *Skolem-Funktion* an. Diese n-stellige interne Funktion²⁵ errechnet für die übergebenen Argumente eine

²⁵ Der Funktionsname soll hierbei mitverantwortlich dafür sein, dass die gebildeten IDREF-Werte überschneidungsfrei sind. Somit gilt $func1(\$n) \lt \> func2(\$n)$ für alle $\$n$. Gleiches folgt für Aufrufe von Skolem-Funktionen mit unterschiedlicher Anzahl von Parametern, beispielsweise $func(\$n) \lt \> func(\$n,1)$ für alle $\$n$.

„Zeige Anzahl der Semesterwochenstunden je Student“	
<p>[XML-QL]</p> <pre> Where <gesamtbelegung> <student matrikel=\$m> <name>\$n</> <(kurs seminar)>\$nr</> </> </> in "http://www.rood.cc/download/01912/studentenbelegung.xml" construct <student ID=func(\$m)> <name>\$n</> <sws> { where <kursangebot><* nummer=\$nr><sws>\$s</></></> in "http://www.rood.cc/download/01912/kursangebot.xml" construct <classic ID=func(\$m,1)>\$sum(\$s)</> } } where <kursangebot><* nummer=\$nr><sws>\$s</></></> in "http://www.rood.cc/download/01912/virtuelle_uni_kursangebot.xml" construct <virtuell ID=func(\$m,2)>\$sum(\$s)</> } </sws> </> </pre>	<p>[Abfrageergebnis]</p> <pre> <?xml version="1.0"?> <XML ID="out.genoid_0"> <student ID="oid.0001"> <name>Thorsten Rood</name> <sws> <classic ID="oid.0005">24</classic> <virtuell ID="oid.0006">0</virtuell> </sws> </student> <student ID="oid.0002"> <name>Fritz Mustermann</name> <sws> <classic ID="oid.0007">12</classic> <virtuell ID="oid.0008">1</virtuell> </sws> </student> <student ID="oid.0003"> <name>Daisy Duck</name> <sws> <classic ID="oid.0009">12</classic> <virtuell ID="oid.0010">0</virtuell> </sws> </student> <student ID="oid.0004"> <name>Jean Luc Picard</name> <sws> <classic ID="oid.0011">0</classic> <virtuell ID="oid.0012">0</virtuell> </sws> </student> </XML> </pre>

Beispiel 11: Query mit Aggregation durch eingebettete Funktionen

2.5 Offene Punkte

2.5.1 XML-Spezialitäten

Entities

Die Auswertung von Entities, die in einer umgebenden DTD definiert wurden, ist derzeit ausgeschlossen. Denkbar wäre eine Deklaration von

```
<!ENTITY % Inf „Informatik“>
```

Queries, welche diese Texte dann analysieren sollen, funktionieren (noch) nicht.

Name-Spaces

Ferner werden Namensräume für XML-Bezeichner nicht unterstützt, die Behandlung von Fragmenten wie beispielsweise

```
<kurs>Graphische Datenverarbeitung I</kurs>
```

und

```
<feu:kurs>Graphische Datenverarbeitung I</feu:kurs>
```

ist noch nicht weiter spezifiziert worden.

2.5.2 Modellierungsprobleme

Trotz Einschränkung des Freiheitsgrades für semistrukturierte Daten durch die Validierung über DTDs lassen sich gültige Inhalte formulieren, die nicht ohne weiteres im XML-QL-Datenmodell abgebildet werden können.

Wenn TAGs ineinander verschachtelt werden, ohne dass der Vorgänger geschlossen wurde, so entsteht #PCDATA-Content, der zu den unter 1.2 gezeigten internen Datenstrukturen nicht vollständig kompatibel ist. Nachfolgendes Beispiel zeigt, dass und wie derartige Informationen in eine plausible Form konvertiert werden können:

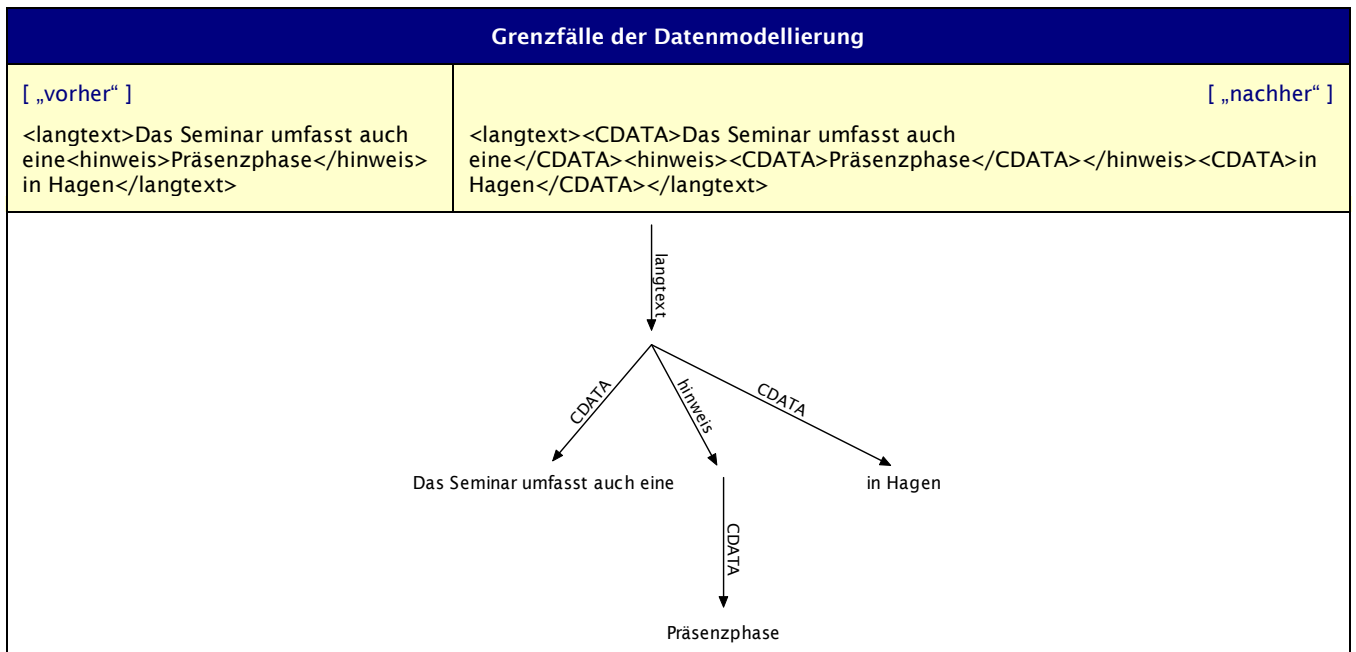


Abbildung 4: Modellierung von XML-Grenzfällen

Offen bleibt hierbei die Frage, in wie weit eine solche Konvertierung transparent ablaufen sollte, da es WHERE-Schablonen geben wird, welche dann Inhalte nicht mehr richtig analysieren können.

3 Fazit

XML-QL scheint eine praktikable Technologie zu sein, XML-Daten in Massenverarbeitung zu behandeln. Beim Experimentieren mit der Sprachsyntax ließen sich viele Anwendungsfälle relativ einfach modellieren – die Leistungsfähigkeit erscheint mir groß genug und zusammen mit der leichten Erlernbarkeit wären somit zwei wichtige Akzeptanzkriterien erfüllt.

Das vorgestellte Konzept zeigt im Originalentwurf und in Dokumenten von Autoren aus der gleichen Forschungsgruppe jedoch einige Abweichungen zueinander²⁸ – der Status „Entwurfsstadium“ lässt sich nicht übersehen.

Zudem stammt die Mehrheit der zu diesem Thema verfassten Arbeiten aus dem Zeitraum 1998/1999²⁹ und die Zahl öffentlich zugänglicher Implementierungen beschränkt sich praktisch gesehen auf den Prototypen von Alin Deutsch³⁰ – selbst dieser ist jedoch nicht in der Lage, die hier gezeigten Beispiele vollständig zu realisieren³¹. Damit entsteht ein „Stiefkind“-Eindruck, welches aber auch für die beiden anderen parallelen Entwicklungszweige *Lorel* und *YaTL*, zu gelten scheint.

Möglicherweise hat sich die hauptsächliche Forschung der letzten Zeit eher auf die **Endverarbeitung** von XML-Dokumenten³² konzentriert, so dass eine Weiterentwicklung erst realistisch erscheint, wenn ein größerer Personenkreis – möglicherweise getragen durch ein oder zwei Mitglieder aus dem kommerziellen IT-Umfeld – hier wieder Interesse und Bedarf anmeldet?

²⁸ Selbst innerhalb des W3C-Dokumentes ([1]) wird die Grammatik in Detailfragen nicht konsequent durchgehalten.

²⁹ Ausgenommen zusammenfassende Arbeiten

³⁰ Siehe [1]

³¹ Um überhaupt eine (Teil)kompatibilität zu erreichen, wurde in den Beispieldaten konsequent auf deutsche Umlaute verzichtet, da der im Prototypen enthaltene Parser erhebliche Probleme mit einem *encoding*-Zusatz in der XML-Kopfzeile hat!

³² XSL, XPath

4 Anhang

4.1 XML-QL Grammatik

XML-QL	::= (Function Query) <EOF>
Function	::= 'FUNCTION' <FUN-ID> '(' (<VAR>(':' <DTD>)?)* ')' (':' <DTD>)? Query 'END'
Query	::= Element Literal <VAR> QueryBlock
Element	::= StartTag Query EndTag
StartTag	::= '<(<ID> <VAR>) SkolemID? Attribute* '>'
SkolemID	::= <ID> '(' <VAR> (',' <VAR>)* ')'
Attribute	::= <ID> '=' ('"' <STRING> '"' <VAR>)
EndTag	::= '<' / <ID>? '>'
Literal	::= <STRING>
QueryBlock	::= Where Construct ('{' QueryBlock '}')*
Where	::= 'WHERE' Condition (',' Condition)*
Construct	::= OrderedBy? 'CONSTRUCT' Query
Condition	::= Pattern BindingAs* 'IN' DataSource Predicate
Pattern	::= StartTagPattern Pattern* EndTag
StartTagPattern	::= '<' RegularExpression Attribute* '>'
RegularExpression	::= RegularExpression '*' RegularExpression '+' RegularExpression '.' RegularExpression RegularExpression ' ' RegularExpression <VAR> <ID>
BindingAs	::= 'ELEMENT_AS' <VAR> 'CONTENT_AS' <VAR>
Predicate	::= Expression OpRel Expression
Expression	::= <VAR> <CONSTANT>
OpRel	::= '<' '<=' '>' '>=' '=' '!='
OrderedBy	::= 'ORDERED-BY' <VAR>+
DataSource	::= <VAR> <URI> <FUN-ID>(DataSource (',' DataSource)*)

4.2 Quellen

Literaturverzeichnis

- [1] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu:
XML-QL: A query language for XML, submission to w3c 19/08/1998,
<http://www.w3.org/TR/NOTE-xml-ql>
- [2] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu:
A query language for XML, leicht modifizierte Version von [1],
<http://www.cis.upenn.edu/~adeutsch/papers/xml-ql-www8.ps>
- [3] M. Fernandez, J. Siméon, P. Wadler:
XML query languages: experiences and exemplars,
<http://www-db.research.bell-labs.com/user/simeon/xquery.html>
- [4] J. Meyer, Prof. Dr. G. Lausen:
XML-QL: A query language for XML, Seminar: „Information processing on the web“, Universität Freiburg, Wintersemester 1999/2000;
<http://www.informatik.uni-freiburg.de/~dbis/lehre/seminar-ws9900/meyer.html>
- [5] S. Oldenburg, H. Meyer:
XML-QL: eine Abfragesprache für XML, Universität Rostock, Sommersemester 2000,
<http://wwwdb.informatik.uni-rostock.de/Lehre/Vorlesungen/hs-xml.html>
- [6] S. Hammermüller:
Data on the web: from relations to semistructured data and XML: XML-QL, TU Wien, Sommersemester 2000,
<http://www.dbai.tuwien.ac.at/education/agenten/index2.html>
- [7] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, S. Subramanian:
XPERANTO: A middleware for publishing object-relational data as XML-documents,
<http://www.cs.cornell.edu/People/jai/papers/XperantoDemo.pdf>
- [8] A. Møller, M. Schwartzbach:
The XML revolution: technologies for the future web, Universität Aarhus,
<http://www.javacommerce.com/tutorial/xml/index.html>
- [9] D. Suciu:
Querying XML,
http://www.cs.washington.edu/homes/suciu/files/_F337312609.ppt
- [10] G. Seidman:
Efficiently storing and querying XML documents, 14/06/2000
<http://www.cs.brown.edu/grad/seminar/gss-xml1.pdf>
- [11] A. Deutsch:
XML-QL demo site,
<http://bilbo.cis.upenn.edu/~adeutsch/xmlql-demo/html/index.html>
- [12] R. Eckstein, N. Klever:
XML, kurz & gut,
O'Reillys Taschenbibliothek, 2. Auflage 2000, ISBN 3-89721-219-6

Beispieldaten

http://www.rood.cc/download/01912/studentenbelegung.*	
[XML]	[DTD]
<pre><?xml version="1.0"?> <!DOCTYPE gesamtbelegung SYSTEM "http://www.rood.cc/download/01912/studentenbelegung.dtd"> <gesamtbelegung> <student matrikel="5010578"> <name>Thorsten Rood</name> <kurs>00507</kurs> <kurs>01692</kurs> <seminar>01912</seminar> </student> <student matrikel="4711000"> <name>Fritz Mustermann</name> <kurs>01692</kurs> <kurs>10555</kurs> <seminar>01912</seminar> <diplomarbeit titel="verteilter Datenbankzugriff mit Java"> <fachbereich>Informatik</fachbereich> <betreuer>Prof. Dr. Tux</betreuer> </diplomarbeit> </student> <student matrikel="0815000"> <name>Daisy Duck</name> <kurs>00507</kurs> <diplomarbeit titel="Kosteneinsparungen im Bestellwesen"> <fachbereich>Wiwi</fachbereich> <betreuer>Prof. Dagobert Duck</betreuer> </diplomarbeit> </student> <student matrikel="3333333"> <name>Jean Luc Picard</name> <diplomarbeit titel="Fehlertolerante Warp-Steuerung"> <fachbereich>Informatik</fachbereich> <betreuer>Prof. Dr. Daniel Duesentrieb</betreuer> </diplomarbeit> </student> </gesamtbelegung></pre>	<pre><!ELEMENT gesamtbelegung (student*)> <!ELEMENT student (name,(kurs seminar diplomarbeit)*)> <!ATTLIST student matrikel ID #REQUIRED> <!ELEMENT name (#PCDATA)> <!ELEMENT kurs (#PCDATA)> <!ELEMENT seminar (#PCDATA)> <!ELEMENT diplomarbeit (fachbereich,betreuer)> <!ATTLIST diplomarbeit titel CDATA #REQUIRED> <!ELEMENT fachbereich (#PCDATA)> <!ELEMENT betreuer (#PCDATA)></pre>

Übersicht 1: Beispieldaten „studentenbelegung“

http://www.rood.cc/download/01912/kursangebot.*	
<p>[XML]</p> <pre><?xml version="1.0"?> <!DOCTYPE kursangebot SYSTEM "http://www.rood.cc/download/01912/kursangebot.dtd"> <kursangebot> <kurs nummer="00507" betreuer="duesentrieb"> <langtext>Unternehmensfuehrung</langtext> <fachbereich>Wiwi</fachbereich> <sws>12</sws> </kurs> <seminar nummer="01912" betreuer="tux"> <langtext>XML und Datenbanken</langtext> <fachbereich>Informatik</fachbereich> <praesenzveranstaltung>Seminarvortraege vor Ort vom 28.06.-29.06.2002</praesenzveranstaltung> <sws>2</sws> </seminar> <kurs nummer="01692" betreuer="duck tux"> <langtext>Graphische Datenverarbeitung I</langtext> <fachbereich>Informatik</fachbereich> <praesenzveranstaltung>2 Wochen (2x Mo-Fr) vor Ort vom 06.05.-17.05.2002</praesenzveranstaltung> <sws>10</sws> </kurs> <betreuer ID="tux"> <titel>Prof. Dr.</titel><name>Tux</name> </betreuer> <betreuer ID="duck"> <titel>Prof.</titel> <vorname>Dagobert</vorname><name>Duck</name> </betreuer> <betreuer ID="duesentrieb"> <titel>Prof. Dr.</titel> <vorname>Daniel</vorname><name>Duesentrieb</name> </betreuer> </kursangebot></pre>	<p>[DTD]</p> <pre><!ELEMENT kursangebot ((kurs seminar betreuer)*)> <!ELEMENT kurs (langtext, fachbereich, praesenzveranstaltung?, sws)> <!ATTLIST kurs betreuer IDREFS #REQUIRED> <!ATTLIST kurs nummer CDATA #REQUIRED> <!ELEMENT seminar (langtext, fachbereich, praesenzveranstaltung?, sws)> <!ATTLIST seminar nummer CDATA #REQUIRED> <!ELEMENT betreuer (titel?, vorname?, name)> <!ATTLIST betreuer id ID #REQUIRED> <!ELEMENT langtext (#PCDATA)> <!ELEMENT fachbereich (#PCDATA)> <!ELEMENT praesenzveranstaltung (#PCDATA)> <!ELEMENT sws (#PCDATA)> <!ELEMENT titel (#PCDATA)> <!ELEMENT vorname (#PCDATA)> <!ELEMENT name (#PCDATA)></pre>

Übersicht 2: Beispieldaten „kursangebot“

http://www.rood.cc/download/01912/virtuelle_uni_kursangebot.*	
<p>[XML]</p> <pre><?xml version="1.0"?> <!DOCTYPE kursangebot SYSTEM "http://www.rood.cc/download/01912/kursangebot.dtd"> <kursangebot> <kurs nummer="10555" betreuer="witzig"> <langtext>Management des Fernstudiums</langtext> <fachbereich>uebergreifend</fachbereich> <sws>1</sws> </kurs> <kurs nummer="11692" betreuer="tux"> <langtext>Graphische Datenverarbeitung I</langtext> <fachbereich>Informatik</fachbereich> <praesenzveranstaltung>4 Wochenenden im Juli 2002</praesenzveranstaltung> <sws>10</sws> </kurs> <betreuer ID="tux"> <titel>Prof. Dr.</titel><name>Tux</name> </betreuer> <betreuer ID="witzig"> <titel>Dr.</titel><name>Witzig</name> </betreuer> </kursangebot></pre>	<p>[DTD]</p> <pre><!ELEMENT kursangebot ((kurs seminar betreuer)*)> <!ELEMENT kurs (langtext, fachbereich, praesenzveranstaltung?, sws)> <!ATTLIST kurs betreuer IDREFS #REQUIRED> <!ATTLIST kurs nummer CDATA #REQUIRED> <!ELEMENT seminar (langtext, fachbereich, praesenzveranstaltung?, sws)> <!ATTLIST seminar nummer CDATA #REQUIRED> <!ELEMENT betreuer (titel?, vorname?, name)> <!ATTLIST betreuer id ID #REQUIRED> <!ELEMENT langtext (#PCDATA)> <!ELEMENT fachbereich (#PCDATA)> <!ELEMENT praesenzveranstaltung (#PCDATA)> <!ELEMENT sws (#PCDATA)> <!ELEMENT titel (#PCDATA)> <!ELEMENT vorname (#PCDATA)> <!ELEMENT name (#PCDATA)></pre>

Übersicht 3: Beispieldaten „virtuelle_uni_kursangebot“