Seminar Datenbanktechnologie für das Web

XML Anfragesprachen
XML Query Languages

Autor: Tibor Dekany
Assistenz: Anca Dobre
Leitung: Prof. Dr. K. Dittrich
31.5.2001, Universität Zürich, Institut für Informatik

Inhaltsverzeichnis

1 Zusammenfassung

XML wird je länger je mehr zum wichtigsten neuen Standard für Datenrepräsentation und den elektronischen Datenaustausch. Unterdessen wurden unterschiedliche Anfragesprachen vorgeschlagen, um an gesuchte Daten in XML−Dokumenten zu kommen, um dann diese in neu geordnete XML−Dokumente auszugeben. Einige Sprachen stammen aus der Datenbankwelt und sind somit ähnlich z.B. zu SQL oder OQL, erweitert um XML−Tauglichkeit, andere verwenden als Grundlage für die Entwicklung einer Anfragesprache direkt XML.

Das World Wide Web Konsortium [W3C], an dem viele akademische Institutionen wie auch wichtige Firmen teilnehmen, hat sich bisher noch nicht für eine bestimmte Anfragesprache als Standard entschieden. Daher sollen in diesem Seminar einige Anfragesprachen mitsamt ihren Vorzügen, Nachteilen, Gemeinsamkeiten und Unterschieden vorgestellt werden[BONIF].

2 Einführung in 5 Anfragesprachen

2.1 LOREL

LOREL[LOREL] (Lightwight Object REpository Language), entwickelt und implementiert an der Stanford Universität, wurde ursprünglich für Anfragen auf

halbstrukturierte Datenbanksysteme entwickelt, und wurde dann später für XML erweitert. Die wichtigsten Eigenschaften von LOREL sind die Ähnlichkeit zu SQL/OQL, die grosse Mächtigkeit für Pfadausdrücke und die nützlichen Mechanismen zum Umgang mit verschiedenen Typen (Integer, Float, String, usw.). LOREL ist leicht lesbar und somit relativ benutzerfreundlich. Diese Anfragesprache eignet sich hervorragend, um Dokumente zu durchsuchen, wenn die Struktur nicht von vorn herein bekannt ist. LOREL wurde mit dem Ziel entwickelt, gut mit grossen Datenbeständen umgehen zu können, Daten aus heterogenen Informationsquellen zu integrieren und in austauschbare Formate umwandeln zu können.

2.2 XML−QL

XML−QL[XMLQL], entwickelt bei AT&T Labs, baut auf SQL auf und erweitert diese Sprache um den Befehl "CONSTRUCT", welcher für die Ausgabe der Resultate einer Suchanfrage zuständig ist. XML−QL erlaubt sowohl Anfragen als auch Transformationen, und kann daher für die Integration mehrerer XML Dokumente verwendet werden, um aus diesen Daten zu extrahieren und in einem neuen Dokument auszugeben.

2.3 XML−GL

XML−GL[XMLGL] verfolgt einen ganz anderen Ansatz. Es ist eine graphische Anfragesprache, und basiert auf die grafische Darstellung der XML Dokumente und DTDs, und zwar als beschriftete Graphen. Entwickelt wurde es an der Politecnico di Milano. Alle Elemente von XML−GL werden visuell dargestellt, daher ist diese Sprache geeignet, um damit ein sehr benutzerfreundliches Interface zu erstellen.

2.4 XSL

XSL[XSL] (Extensible Stylesheet Language) hat Eigenschaften, welche als Grundlage für eine Anfragesprache dienen kann. Ein XSL Programm besteht aus mehreren Regeln (template rule), wobei sich jede aus 2 Teilen zusammensetzt: Zuerst wird ein XML Dokument nach einem bestimmten Muster durchsucht und ein Ergebnisbaum gebildet (Transformation, mit XSLT[XSLT]), im zweiten Teil wird dann dieser Baum formatiert. Um Elemente auszuwählen und zu bearbeiten, um bedingte Verarbeitung und um Texte zu generieren, verwendet XSL die Sprache XPath[XPATH].

2.5 XQL

XQL[XQL] ist eine Sprache, um aus XML Dokumenten Elemente und Texte auszuwählen und zu filtern. XQL kann als eine Art Erweiterung zur Mustersyntax (pattern syntax) von XSL angesehen werden, welche um Boolsche Logik, Filter und Indizierung von Knotensammlungen erweitert wurde . Es wurde mit dem Ziel entwickelt, syntaktisch sehr einfach und kompakt zu sein, auf Kosten der Ausdruckskraft. XQL ist keine vollständige Abfragesprache. Dafür hat XQL den grossen Vorteil, so kompakt zu sein, dass eine Anfrage in eine URL passt und so direkt in einen Browser eingegeben werden kann wie auch als Lesezeichen (Bookmark) gespeichert werden kann. An der Entwicklung und Implementierung von XQL sind die Firmen Texcel Inc, webMethods Inc, und Microsoft massgeblich beteiligt.

3 Eigenschaften

In diesem Kapitel werden die verschiedenen Eigenschaften der oben genannten Anfragesprachen dargelegt.

3.1 Datenmodell

Einzig XQL setzt auf das durch XML gegebene Datenmodell auf. LOREL, XML−QL, XML−GL und XSL verwenden je eigene Datenmodelle.

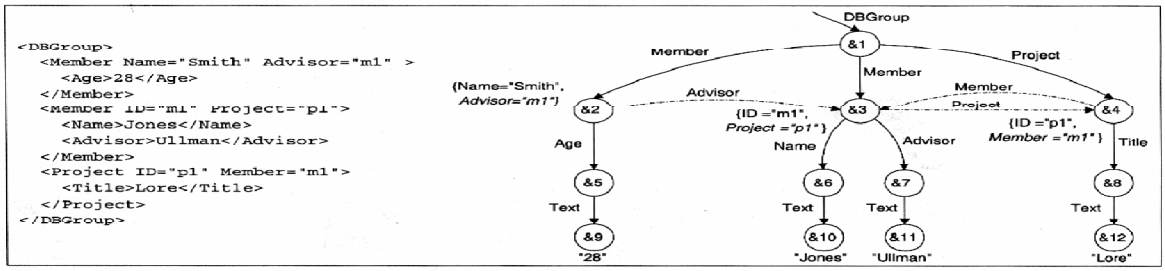
3.1.1 Lorel

Das LOREL Datenmodell ist ein Graphenmodell, wobei die Knoten des Graphen die XML (Daten−) Elemente darstellen, während die beschrifteten Kanten entweder normale Kanten des Graphen oder Querverweise darstellen. Ein XML Element ist ein Paar <eid,value>, wobei *eid* ein eindeutiger *element identifier* ist und *value* entweder ein atomarer Textstring oder ein komplexer Wert mit vier Komponenten ist:

1. String tag entsprechend dem XML tag für dieses Element
2. Geordnete (möglicherweise leere) Liste von Paaren von Attributnamen und atomaren Werten (diese Paare repräsentieren die XML Attribute), atomare Werte haben einen Typ von integer, real, string, usw. oder ID, IDREF(S)
3. Geordnete (möglicherweise leere) Liste von Querverweisen, dargestellt als <eid, value>− Paare, welche die XML IDREF Attribute darstellen.

4. Geordnete (möglicherweise leere) Liste <eid, value> normaler Subelemente, welche sich aus der normalen Verschachtelung des XML−Dokuments ergeben.

   Jeder XML Graph hat einen oder mehrere "entry point" Knoten.



3.1.2 XML−QL

Bei XML−QL ist ein XML Dokument als XML (geordneter oder auch ungeordneter) Graph modelliert. Jedem Knoten wird ein eindeutiger OID (object identifier) zugewiesen. Die Kanten werden mit den Element tags bezeichnet, die Knoten werden mit den Paaren Attribut und Wert beschriftet, und die Blätter erhalten String−Werte (z.B. CDATA, PCDATA). Jeder Graph hat genau einen mit root bezeichneten Knoten (Wurzel).

3.1.3 XML−GL

XML−GL verwendet das sogenannte "XML Graphical Data Model" (XML−GDM), welche sowohl XML Dokumente, wie auch XML DTDs repräsentieren kann. XML−GDM selber wird ebenfalls zur Formulierung von Anfragen benutzt. Das XML−GDM benutzt 3 Begriffe:

1. *Objekte*, welche als Rechteck dargestellt werden. Dies ist ein abstraktes Element ohne direkt darstellbaren Wert, d.h. mit einem Objekt wird ein nicht−terminales XML Element dargestellt, welches weitere Unterelemente hat.

2. *Eigenschaften* (Properties), dargestellt als Kreis, welches verbunden ist mit einem Objekt, deutet auf einen repräsentierbaren, zum Objekt gehörenden Wert (z.B. CDATA, PCDATA), sogenannte terminale Elemente. *Eigenschaften* haben einen Namen und einen Typ.

3. *Beziehungen* (Relationships), kommen als gerichtete Kanten zwischen 2 Objekten vor, und bedeuten eine semantische Assoziation (z.B. Verschachtelung, Querverweis) von einem Quellobjekt zu einem Zielobjekt.

3.1.4 XSL

XSL operiert über 3 Dokumenten: der Quelle, dem Resultat der Anfrage und dem Stylesheet. Dies ist ein zweistufiger Prozess, im ersten wird das XML Quelldokument (der Quellbaum) in einen Ergebnisbaum umgeformt (für die Transformation von Bäumen wird hierbei die spezielle Sprache XSLT benutzt), in einem zweiten Schritt, *xsl formatted object*, kann das Ergebnis beliebig "formatiert", d.h. ausgegeben werden, z.B. als HTML, PDF, XML, usw. XSL benutzt das gleiche Datenmodell wie in XPath spezifiziert, ein XML Dokument wird daher als Baum betrachtet, dessen Knoten aus 7 verschiedenen Typen bestehen: root node, element node, text node, attribute node, namespace node, processing instruction node und comment node. Für jeden Knotentyp existiert eine Methode, um es als einen String darzustellen.

3.1.5 XQL

Die Designer von XQL legen das "XML implizierte Datenmodell" zu Grunde. Dabei hat jeder Knoten einen bestimmten Typ und entweder einen Inhalt (content), d.h. ein Subelement, oder einen bestimmten Wert. Der semantische Bezug zwischen Knoten kann hierarchisch ("Vorfahre / Nachkomme"), örtlich (absolut, relativ, im Bereich von), und sequentiell ("geht vor", "geht unmittelbar vor") sein.

3.1.6 Fazit

Die Datenmodelle von LOREL, XML−QL, XML−GL und XSL können geordnet wie auch ungeordnet sein. Aber einzig XML−QL ist in der Lage, die Ordnung bei einer Anfrage zu berücksichtigen. Die Datenmodelle von LOREL, XML−QL und XML−GL sind grundsätzlich äquivalent, der grösste Unterschied ist wohl die Behandlung von IDREFs, womit ein Element auf ein bereits vorhandenes Element an einer anderen Stelle verweisen kann. Hierfür gibt es 2 Betrachtungsweisen: als "Objekt Referenz" (eine Art hard−link) kann direkt von einem Element zum anderen gesprungen werden, hierbei könnten aber Zyklen entstehen, und als "Verschachtelter Verweis", welcher nur als ein String interpretiert wird. Einzig LOREL beherrscht beide Interpretationen, die Modi heissen im ersten Fall "semantic" und im zweiten "literal", d.h. im "semantic mode" wird die Datenbank als ein Graph (mit Kanten zwischen den referenzierten Elementen), im "literal mode" als ein XML−Baum betrachtet, wo die Verweise als String angesehen werden.

3.2 Syntax

In diesem Kaptitel soll die Grund−Syntax der Anfragesprachen in BNF (Backus Naur Form) dargelegt werden. Hierbei sind Terminale mit Hochkommas (’) Umschlossen, geschweifte Klammern ({}) bezeichnen keine, eine oder mehrere, durch Kommas getrennte Elemente, eckige Klammern ([]) bedeuten optionales Auftreten, und ein senkrechter Strich (|) steht für genau eine Auswahl der dadurch getrennten Elemente.

Eine Abfrage besteht generell aus 3 Teilen:

1. Muster (pattern): Der Muster−Teil der Anfrage überprüft Übereinstimmungen von verschachtelten Elementen und bindet Variablen an Elemente.
2. Filter: Die Filter−Klausel testet die gebundenen Variablen.
3. Konstruktor: Mit dem Konstruktor wir das Ergebnis spezifiziert, als Term von gebundenen Variablen.

3.2.1 LOREL

```
’select’ { select_expr }
[ ’from’ { from_expr } ]
[ ’where’ { where_expr } ]
```

Die Anfrage ist SQL−typisch "select − from − where". Die Sprache ist orthogonal, d.h. jeder Ausdruck selbst kann wieder eine Anfrage enthalten (wie dies auch in OQL möglich ist).

3.2.2 XML−QL

```
Query ::=    ’where’ { Predicate }
             ’construct’ { ’{’ Query ’}’ }
```

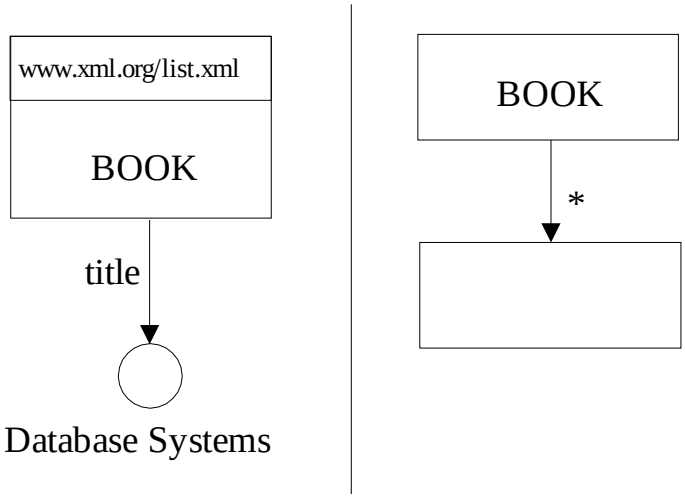
Das Resultat dieser Anfrage, welche in der construct−Klausel angegeben wird, ist ein XML Dokument.

3.2.3 XML−GL

Eine Anfrage in XML−GL besteht immer aus 2 durch einen senkrechten Strich getrennten Mengen von gerichteten, unzyklischen Graphen. Hierbei wird im linken Teil (LHS), dem

"extract part", das Ziel ausgewählt (XML Dokumente und Bereiche), im optionalen "match part" (Prädikat), welches in etwa in SQL der "where" Klausel entspricht, werden zu erfüllende logische Konditionen angegeben. Auf der rechten Seite (RHS) wird im "construct part" die Struktur des Ergebnisdokumentes angegeben (das Einfügen neuer Elemente, definieren neuer Verweise und restrukturieren von Eigenschaften von Elementen ist möglich), im optionalen "clip part" wird angegeben, welche Elemente (inkl. alle rekursiven Subelemente) des urspünglichen Dokumentes im Resultat erhalten bleiben sollen (vergleiche mit SQL: select−statement, um im Resultat nur bestimmte Attribute einer Relation auszugeben anstatt alle mit "select *").

Beispiel: Finde alle <Book> Elemente, welche den Titel "Database Systems" haben, und gib sie mit allen Subelementen (z.B. Author) aus.



3.2.4 XSL

```
’<xsl:template’ [ ’match=’ pattern_expr ] ’>’
   { xsl:directive }
   { result−elements }
’</xsl:template>’
```

In XSL wird eine template−Regel mit dem tag xsl:template eingeführt. Der *match* Teil sorgt für die Auswahl der richtigen XML Knoten im XML Baum, auf welche die Regeln angewendet werden sollen (ohne diesen Teil werden die Regeln auf das ganze XML Dokument angewendet). `pattern_expression` ist ein Ausdruck in der XPath −Sprache.

`result−elements` gibt die neuen Elemente im Ergebnisbaum an, und `xsl:directive` ist eine der folgenden Möglichkeiten (weitere siehe [XSLT]):

- `’<xsl:apply−templates’ [ ’select=’ pattern_expr ] ’>’`
- `’<xsl:for−each’ ’select=’ pattern_expr ’>’`
- `’<xsl:value−of’ [ ’select=’ pattern_exprt ] ’>’`
- `’<xsl:copy−of’ ’select=’ pattern_expr ’>’`

  Das `select` Attribut wählt hier einen speziellen Knoten zur Bearbeitung aus, damit die Nachfolgerknoten von der Bearbeitung ausgeschlossen werden. Die `xsl:apply−templates` Direktive ruft ausserhalb definierte templates auf, `xsl:for−each` iteriert über alle angegebenen Direktiven. `xsl:value−of` gewinnt alle Informationen aus den angewandten pattern_expressions und konvertiert diese in einen String, während `xsl:copy−of` die Ergebnisse nicht in einen String konvertiert, sondern das originale Format aus dem Qell−XML−Baum beibehält.

3.2.5 XQL

```
Query ::=    [ ’./’ | ’//’ | ’///’ | ’.//’ ] Element [ ’[’
Predicate ’]’ ] [ Path ]

Path ::=     [ ’//’ | ’///’ ] Element [ ’[’ Predicate ’]’ ] [
Path ]
```

XQL unterstützt nur *Muster* und *Filter*, keine *Konstrukroren*. Das ist ein grosser Nachteil von XQL, da viele Anfragen dadurch nicht ausdruckbar sind. Wie an der Syntax erkennbar ist, wird mit XQL hierarchisch durch einen XML−Baum navigiert, wobei Prädikate (Filter) auf Elemente und Texte entlang des Pfades angewandt werden. Das Resultat ist wieder ein XML Dokument.

3.3 Joins

Eine Join Bedingung vergleicht 2 oder mehrere XML Attribute oder Daten, welche zu einem oder zu verschiedenen Dokumenten gehören. Meistens wird auf Gleichheit überprüft (equi−join).

Lorel unterstützt Join Bedingungen vollständig, auch über mehrere Dokumente. Die zu

vergleichenden Variablen können explizit angegeben werden, die Syntax ist ähnlich zu SQL.

Auch XML−QL kann Joins verarbeiten, indem Variablen möglicherweise verschiedener Dokumente miteinander verglichen werden.

In XML−GL werden Joins graphisch dargestellt, indem 2 Endknoten (=Blätter), welche Attribute oder Daten enthalten, mit einer Kante, beschriftet mit einem Vergleichsoperator, verbunden werden.

XSL hingegen kennt Joins überhaupt nicht, XQL erlaubt nur semi−joins, d.h. kann Attribute / Daten nur im gleichen Baum miteinander vergleichen, und somit nur im gleichen XML Dokument. Z.B. ermöglicht XQL in einem Bücher−XML Dokument die Suche nach allen Büchern, welche den selben Autor haben, wie der Autor von "Moby Dick":

```
book[author=//book[title='Moby Dick']/author]
```

4 Vergleich und Beispiele

Unser Beispielszenario handelt von einer Autohandelsfirma. Hierbei kommen 2 XML Dokumente zum Einsatz. Zum einen das Dokument der Hersteller (manufacturer), in welchem verschiedene Autohersteller eines Jahres mit ihren produzierten Automodellen und deren mehrfacher Bewertung (Aussehen des Autos von vorne / von der Seite, Gesamtrang) gespeichert sind. In einem 2 Dokument (vehicle) werden die Verkäufe gespeichert, welcher Verkäufer hat welches Automodell (Name, Jahr) welcher Marke und Farbe für wieviel Geld verkauft.

Datei: www.inter.net/manufacturers.xml

```
<manufacturer>
        <mn_name>Fiat</mn_name>
        <year>1999</year>
        <model>
                <mo_name>Punto</mo_name>
                <front_rating>3.84</front_rating>
                <side_rating>2.14</side_rating>
                <rank>9</rank>
        </model>
        <model>
                ...
        </model>
        ...
</manufacturer>
<manufacturer>
        ...
</manufacturer>
```

Datei: www.inter.net/vehicles.xml

```
<vehicle>
        <vendor>Urs Müller</vendor>
        <make>Fiat</make>
        <model>Punto</model>
        <year>1999</year>
        <color>metallic blau</color>
        ...
        <price>19999</price>
</vehicle>
<vehicle>
        ...
</vehicle>
```

4.1 Selektion und Ausgabe

Dies ist das "Hello World" Programm für Anfragesprachen, eine einfache Anfrage mit der Ausgabe des Resultates, was natürlich alle Afragesprachen können müssen.

Aufgabe: Gib alle `<manufacturer>` Elemente aus, welche mindestens 1 `<model>` unter den Top 10 haben.

4.1.1 LOREL

```
select M
from db.manufacturer M
where M.model.rank <= 10
```

Für SQL Kenner ist es sicher kein Problem, diese Anfrage zu verstehen. Der Konstruktor befindet sich in der select Zeile, die from Zeile ist das Muster und die where Klausel ist der Filter. Das Resultat ist allerdings kein XML Dokument, sondern eine Menge von OIDs, welche auf die Elemente in der abgefragten Datenbank zeigt. Die Anfrage findet in der Umgebung einer Datenbank statt (im XML kompatiblen Datenmodell von LOREL), daher wird nicht direkt eine Datei als Quelle angegeben, die Datei wurde schon vorher importiert. `db` ist der sogenannte "entry point", ein Zeiger, um auf die manufacturer Datenbank zugreifen zu können.

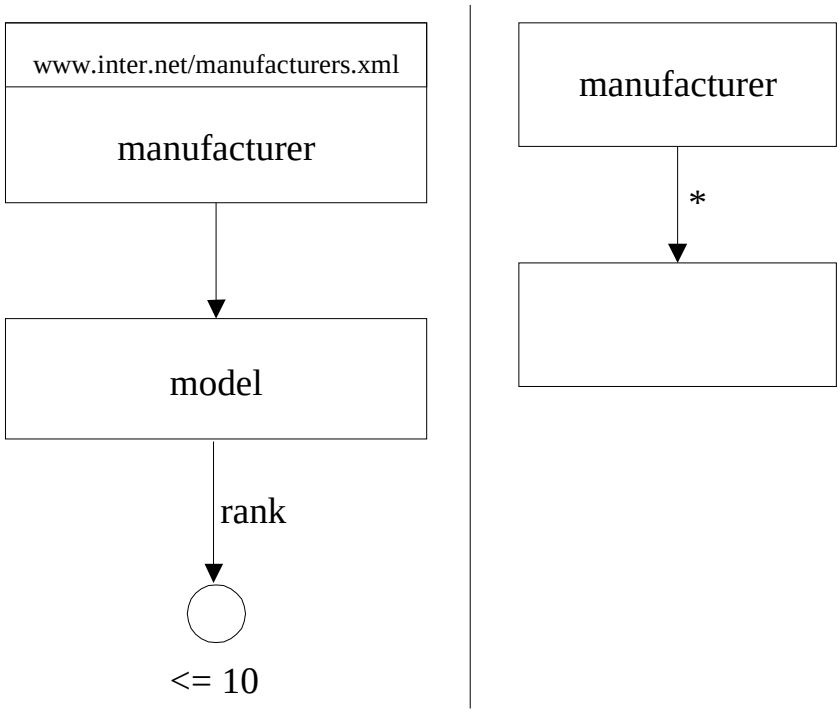
4.1.2 XML−QL

```
WHERE     <manufacturer>
               <model>
                    <rank>$r</rank>
               </model>
          </manufacturer> ELEMENT_AS $m IN
                    www.inter.net/manufacturers.xml, $r<=10
CONSTRUCT $m
```

Muster und Filter stehen hier in der where Klausel, der Konstruktor in der constructor Zeile. Die Anfrage bezieht sich direkt auf das Dokument mit seiner URL. Der Inhalt des Elementes `<rank>` wird an die Variable r gebunden, welche dann im Filterteil auf die Bedingung <= 10 getestet wird. Im Konstruktorteil werden die Elemente spezifiziert, welche Ergebnis der Anfrage sein sollen. Das Resultat ist stets eine duplikatfreie Ergebnismenge.

4.1.3 XML−GL

Auch hier wird das zu durchsuchende Dokument direkt per URL angegeben. Bei der Anfrage werden alle `<manufacturer>` Elemente ausgegeben, welche die links stehende Bedingung erfüllen. Auf der rechten Seite steht der Konstruktor, in diesem Fall also genau die selektierten `<manufacturer>` Elemente, inklusive aller Subelemente (aber ohne die per IDREF angegebenen Elemente, diese werden als String ausgegeben). Das Ergebnis ist ein neues XML Dokument, welches von `<result> ... </result>` umschlossen wird.



4.1.4 XSL

```
<xsl:template match="/">
     <xsl:for-each select="manufacturer[model/rank<=10]">
          <xsl:value-of />
     </xsl:for-each>
</xsl:template>
```

Die XSL Regel gilt hier für den Wurzelknoten (root node) des XML Dokumentes, die `xsl:for-each` Directive wird für jeden `<manufacturer>` Knoten ausgeführt, welcher mindestens ein Modell mit rank<=10 hat, xsl:value-of übernimmt eben diesen Knoten in den Ergebnisbaum.

4.1.5 XQL

```
manufacturer[model/rank<=10]
```

In XQL ist die Anfrage kurz und prägnant: auf das Navigationsmuster (pattern) folg der Filter mit der Bedingung für `<rank>`, alles auf einer Zeile. Das Ergebnis wird von `<xql:result> ... </xql:result>` umschlossen.

4.2 Reduktion

In diesem Beispiel wollen wir das XML Dokument reduzieren, d.h. gewisse Teile weglassen.

Aufgabe: Lasse vom manufacturers.xml Dokument alle `<model>` Subelemente weg, welche einen <rank> grösser als 10 haben. Ausserdem soll das Ergebnis weder `<front_rating>` noch `<side_rating>` enthalten. Leider unterstützt keines der 5 Anfragesprachen Reduktion direkt, sondern nur über Umwege, indem angegeben wird, welche Elemente wir im Resultat haben möchten, anstatt anzugeben, welche wir ausschliessen wollen. Daher funktioniert dies nur, wenn die DTDs im voraus bekannt sind.

4.2.1 LOREL

```
select Z.mn_name, Z.year,
     (select Z.model.mo_name, Z.model.rank
     where Z.model.rank <= 10)
from db.manufacturer Z
```

Hier besteht die Anfrage aus 2 geschachtelten Statements. Für jedes `<manufacturer>` Element wird die Unteranfrage ausgeführt, welche die gewünschten Modelle ohne die ratings an die Oberanfrage liefert. Beide Statements sind existentiell quantifiziert.

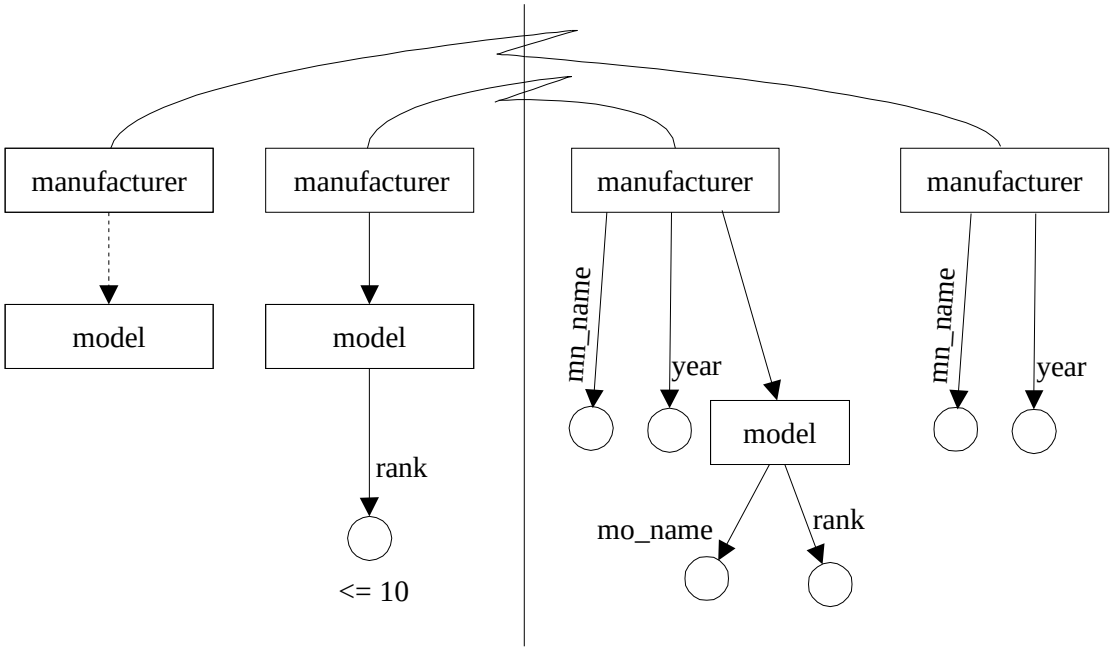
4.2.2 XML−QL

```
WHERE
     <manufacturer>
          <mn_name>$mn</mn_name>
          <year>$y</year>
     </manufacturer> CONTENT_AS $m IN
                     www.inter.net/manufacturers.xml
CONSTRUCT
     <manufacturer>
          <mn_name>$mn</mn_name>
          <year>$y</year>
          { WHERE
               <model>
                    <mo_name>$mon</mo_name>
                    <rank>$r</rank>
               </model> IN $m, $r<=10
          CONSTRUCT
               <model>
                    <mo_name>$mon</mo_name>
                    <rank>$r</rank>
               </model>
          }
     </manufacturer>
```

Auch in XML−QL wird die Anfrage mit Hilfe von verschachtelten Anfragen bewerkstelligt, die Verschachtelung findet im `CONSTRUCT` Teil statt.

4.2.3 XML−GL

XML−GL bietet keine Möglichkeit zur Verschachtelung, somit werden zuerst alle <manufacturer> ausgewählt, welche überhaupt kein <model> Subelement enthalten, und übergibt diese ins Resultat. Zweitens werden diejenigen <manufacturer> seelktiert, welche mindestens ein <model> unter den Top 10 haben. Von diesen Subelementen werden dann nur die gewünschten Angaben ins Ergebnis übernommen.



4.2.4 XSL

```
<xsl:template match="manufacturer[model/rank<=10]">
     <model>
          <xsl:value−of select="mo_name"/>
          <xsl:value−of select="rank"/>
     </model>
</xsl:template>
```

Hier wird für jeden <manufacturer> die Bedingung auf alle Modelle überprüft. Trifft die Bedingung zu, werden <mo_name> und <rank> ins Resultat übernommen.

4.2.5 XQL: nicht möglich

Die Anfrage kann in XQL weder als Reduktion, noch als Konstruktion ausgedrückt werden. XQL erlaubt weder Restrukturierung, verschachtelte Anfragen, noch Joins.

4.3 Joins

Aufgabe: finde Paare von `<manufacturer>` und `<vehicle>`, unter der Bedingung, dass `<mn_name> = <make>`, `<mo_name> = <model>` und `<year> = <year>`.

4.3.1 LOREL

```
temp:=    select (M,V) as pair
          from db.manufacturer M, db.vehicle V
          where M.mn_name = V.make
               and M.model.mo_name = V.model
               and M.year = V.year
```

Hier werden nach dem Join Paare von OID gebildet, welche mit einem neuen entry point (Zeiger) "temp" versehen werden (Zugriff: `select T from temp.pair P` usw.).

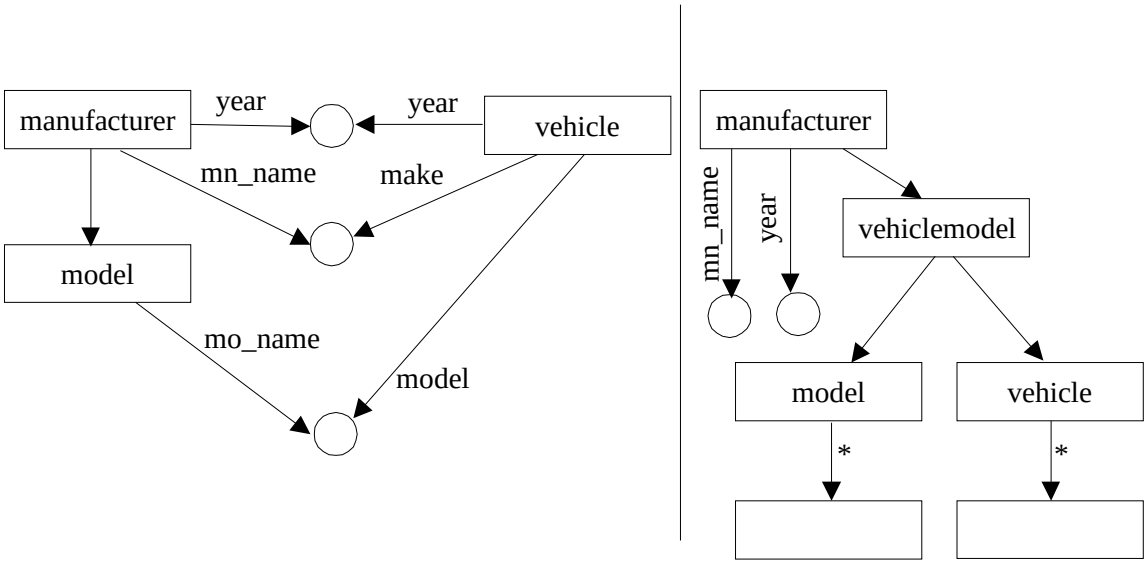
4.3.2 XML−QL

```
WHERE
     <manufacturer>
          <mn_name>$mn</mn_name>
          <year>$y</year>
          <model>
               <mo_name>$mon</mo_name>
          </model> CONTENT_AS $mo
     </manufacturer> CONTENT_AS $m IN
          www.inter.net/manufacturers.xml
     <vehicle>
          <model>$mon</model>
          <year>$y</year>
          <make>$mn</make>
     </vehicle> CONTENT_AS $v IN
          www.inter.net/vehicles.xml
CONSTRUCT
     <manufacturer>
          <mn_name>$mn</mn_name>
          <year>$y</year>
          <vehiclemodel>
               $mo,$v
          </vehiclemodel>
     </manufacturer>
```

Hier wird ein neues XML Stück erstellt, der Inhalt der übereinstimmender `<model>` und `<vehicle>` Elemente wird mit den `<vehiclemodel>` Tags umschlossen.

4.3.3 XML−GL

Hier werden die gewonnenen Paare `<model>` und `<vehicle>` zum Subelement vom neuen `<vehiclemodel>` Element welches wiederum ins `<manufacturer>` Element eingefügt werden.



4.3.4 XSL, XQL: nicht möglich

Keine dieser Sprachen unterstützt Joins (Ausnahme XQL: semijoins), ausserdem fehlt bei beiden die Möglichkeit, 2 unterschiedliche Dokumente verarbeiten zu können.

4.4 Restrukturierung

In diesem Beispiel möchten wir das Ergebnis aus dem letzten Beispiel verwenden, um als Resultat `<car>` Elemente zu erhalten, welche `<make>`, `<model>`, `<vendor>`, `<rank>` und `<price>` genau in dieser Reihenfolge enthalten.

4.4.1 LOREL

```
select xml(car: (select X.vehicle.make, X.vehicle.model,
                        X.vehicle.vendor,
X.manufacturer.rank,
                        X.vehicle.price
                from temp.pair X))
```

Die Elemente werden in der selben Reihenfolge aus dem Dokument extrahiert, wie sie in der verschachtelten select−Klausel erscheinen. Die Funktion `xml(car: querystring)`

erzeugt ein XML Element <car>, welches die im select−statement spezifizierten Elemente beinhaltet.

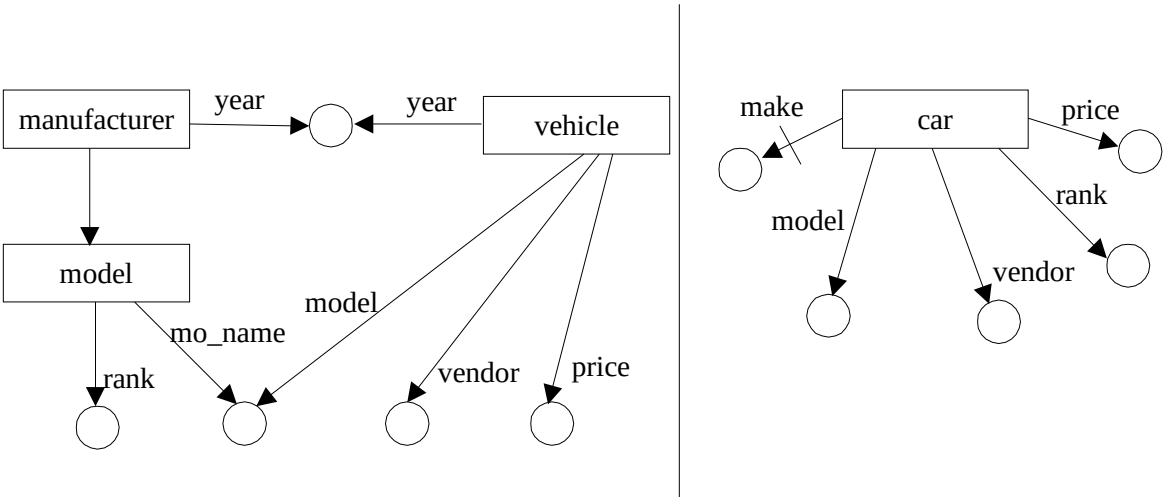
4.4.2 XML−QL

```
WHERE
     <manufacturer>
          <mn_name>$mn</mn_name>
          <vehiclemodle>
               <model>
                    <mo_name>$mon</mo_name>
                    <rank>$r</rank>
               </model>
               <vehicle>
                    <price>$p</price>
                    <vendor>$mn</vendor>
               </vehicle>
          </vehiclemodel>
     </manufacturer> IN www.inter.net/queryresult3.xml
CONSTRUCT
     <car>
          <make>$mn</make>
          <mo_name>$mon</mo_name>
          <vendor>$v</vendor>
          <rank>$r</rank>
          <price>$p</price>
     </car>
```

Die neue Struktur des Dokumentes entsteht in der CONSTRUCT−Klausel. Neue Elemente werden dort definiert und eingefügt, eine Funktion hierfür ist unnötig. Die Benutzung ist sehr intuitiv, das Muster aus dem CONSTRUCTOR−Teil spiegelt das neue XML Dokument wieder, einzig die Variablen werden ersetzt.

4.4.3 XML−GL

Das neue <car> Element wird im Resultat, auf der rechten Seite, definert, welches mit den entsprechenden Subelementen ausgestattet wird. Diese werden dem Gegenuhrzeigersinn nach sortiert, angefangen beim markierten Element.



4.4.4 XSL

```
<xsl:template match="manufacturer">
     <car>
          <xsl:value-of select="vehiclemodel/vehicle/make"/>
          <xsl:value-of select="vehiclemodel/model/mo_name"/>
          <xsl:value-of select="vehiclemodel/vehicle/vendor"/>
          <xsl:value-of select="vehiclemodel/model/rank"/>
          <xsl:value-of select="vehiclemodel/vehicle/price"/>
     </car>
</xsl:template>
```

Hier wird die Anfrage über die <manufacturer> Elemente ausgeführt, und die neuen <car> Elemente gebildet, mit allen gewünschten Subelementen.

4.4.5 XQL: nicht möglich

Ohne Konstruktor ist eine Neuordnung nicht möglich.

5 Schlussbemerkung

	LOREL	*XML−QL*	*XML−GL*	*XSL*	*XQL*
Eigenes Datenmodell	X	X	X	X	−
Joins	X	X	X	−	−
Abbruch bei Zyklen	X	undef.	X	−	undef.
Existentielle Quantifizierung	X	X	X	X	X
Universelle Quantifizierung	X	−	−	−	X
Negation	X	−	X	X	X
Reduktion	−	−	−	−	−
Konstruktion neuer Elemente	X	X	X	X	−
Verschachtelte Anfragen	X	X	−	X	−
Resultat sortieren	X	X	X	X	−
Typenunterstützung	X	−	−	−	−
Unterstützung XPointer & XLink	−	−	−	−	−

Diese Tabelle gibt noch einmal einen Überblick und führt ein paar weitere Eigenschaften auf, welche von einer Anfragesprache erwartet werden.

Anfragesprachen sollen möglichst deklarativ sein, eine hohe Ausdruckskraft haben und einfach zu benutzen sein.

Deklarativheit: Eine Anfragesprache ist deklarativ, wenn die Anfrage das gewünschte Resultat beschreibt, und nicht etwa den Weg, wie es zu diesem Resultat kommen soll. So gesehen sind alle 5 Sprachen deklarativ

Ausdrucksmächtigkeit: Dies zeigt, wie gut eine Sprache möglichst viele Anfragemöglichkeiten abdecken kann. Von diesem Standpunkt aus gesehen sind vor allem LOREL, aber auch XML−QL die mächtigsten Sprachen. XML−GL ist etwas weniger mächtig, aber durch seinen graphischen Formalismus sehr natürlich. XSL und XQL sind ähnlich stark, beiden fehlt aber die Möglichkeit, mehrere Dokumente zu verarbeiten. XSL ist eine Stylesheet Sprache, welche gut geeignet ist, Umstrukturierungen von XML Dokumenten vorzunehmen, was wohl auch das wichtigste Ziel dieser Sprache war. Mit XQL wollten die Entwickler eine Möglichkeit schaffen, Elemente aus XML Dokumenten auszuwählen und auch möglichst effektiv Volltextsuchen auf sehr grosse Dokumente durchzuführen, daher ist XQL eine sogenannte dokumentorientierte Sprache, welche keine vollständige

Anfragesprache ist, was sie auch nie sein wollte. Es ist vielmehr als der nützlichste Teil einer Anfragesprache anzusehen.

Einfache Benutzbarkeit: XML−GL ist sowohl einfach zu lesen und leicht verständlich. Zusammen mit XQL sind das die wohl einfachsten erlernbaren und benutzbaren Sprachen. Demgegenüber steht XSL, welche mit den vielen Tags und Regeln auf den ersten Blick wenig verständlich aussieht. Mit LOREL und XML−QL kann man, nach ein Wenig Einarbeitungszeit, relativ einfach auch kompliziertere Anfragen formulieren. Datenbankexperten werden wohl eher mit LOREL Vorliebe nehmen wegen der Ähnlichkeit zu SQL, XML Spezialisten werden sich wahrscheinlich mit XML−QL schneller anfreunden.

Literaturverzeichnis

BONIF: Bonifati, Ceri, Comparative Analysis of Five XML Query Languages, SIGMOD Record 29(1), 2000, 68−79
LOREL: www−db.stanford.edu/lorel
W3C: www.w3c.org
XMLGL: Stefano Ceri, Sara Comai, Piero Fraternali, Stefano Paraboschi, Letizia Tanca, Ernesto Damiani, XML−GL: A Graphical Language for Querying and Restructuring XML Documents, 1999, www.w3.org/TandS/QL/QL98/pp/xml−gl.html
XMLQL: www.research.att.com/sw/tools/xmlql
XPATH: www.w3.org/TR/xpath
XQL: Jonathan Robie, Joe Lapp, David Schach, XML Query Language (XQL), 1998, http://www.w3.org/TandS/QL/QL98/pp/xql.html
XSL: www.w3.org/TR/xsl
XSLT: www.w3.org/TR/xslt