Kapitel 6
XML-Anfragesprachen

- XPath ist noch keine Anfragesprache:
  selektiert nur Mengen von Knoten, nichts Tupelartiges
- weitere Funktionalität von Anfragesprachen:
  - Zusamensetzen von Tupeln aus mehreren Knoten eines Pfades
  - Joins
  - Dereferenzierung
    - via Joins
    - via ID-Direktzugriff
    - via Attributverfolgung
  - Aggregationen
  - Ergebnisformatierung und -strukturierung
  - Berücksichtigung, dass Elemente geordnet sind!

XML-ANFRAGESPRACHEN

Gesammelte Erfahrungen aus SQL, OQL, OEM/WSL/MSL, F-Logic und einigen anderen ...

- Vorformen von XPath: XSL Patterns/XPointer/XQL (1998)
- XQL erweiterte die frühe “Grundform” zu einer Anfragesprache
  - Hinzunahme diverser Konstrukte innerhalb der Pfadausdrücke
  - zunehmend kompliziert, unübersichtlich
  - keine ausreichende Ausdrucksfähigkeit
  - zeigte Grenzen und Anforderungen auf
- XML-QL (1998): pattern-matching-basierte “Extraktionssprache”
  - nicht pfad-, sondern ausschliesslich “schablonen”-basierte Bindung von Variablen
  - Semantik durch Klauselkonstrukt
  - Strukturierung des Ergebnisses durch XML-Pattern mit den gebundenen Variablen

XML-ANFRAGESPRACHEN (FORTS.)

- Quilt (2000): SQL-artige Ergänzung auf XPath aufbauend
  - Bindung von Variablen durch XPath-Ausdrücke
  - Geschachtelte Schleifen durch for-Klausel
  - Zusatzbedingungen in where-Klausel
  - Strukturierung des Ergebnisses durch return-Klausel
- XQuery (2001): “offizielle” Version von Quilt
  - W3C-Draft XPath/XQuery integriert
- Hinweis: XQuery ist case-sensitive.
  ALLE KEYWORDS MÜSSEN klein GESCHRIEBEN WERDEN!

6.1 XQL

XQL (XML Query Language; 1998) ist eine einfache Anfragesprache, die einen frühen Teil von XPath umfaßt:

- alle XPath-Ausdrücke, die ohne Verwendung von “*axis*::” gebildet sind (vgl. Folie 174 - die Achsen wurden erst später “erfunden”).
- *nodetest* war ebenfalls noch nicht erfunden; text() war eine Funktion
- Funktionsanwendungen wurden mit “!” am Ende des Pfadausdrucks angehängt:
  //country/name!text()

Weitere Anfrage-Funktionalität wurde syntaktisch in die Pfadausdrücke integriert.

XQL: Boolesche und Mengenoperatoren

- q_1 union q_2, $q_1 \mid q_2$
- q_1 intersect q_2
- $q_1 \sim q_2$ (Vereinigung, falls beide nichtleer sind)
- q_1 or q_2
- q_1 and q_2

XQL: Return-Operatoren

Operatoren zur Ausgabe des an dieser Stelle adressierten Knotens:

??: Der gesamte Knoten wird ausgegeben (mit Attributen und Subelementen)

?: Nur der direkte Inhalt wird ausgegeben

- country/city[isCountryCap]/Name

```
<name>Berlin</name>
<name>Rome</name>
```

- country?/city[isCountryCap]/Name

```
<country> <name>Berlin</name> </country>
<country> <name>Rome</name> </country>
```

- country?[@car_code?]/city[isCountryCap]/Name

```
<country car_code="D"> <name>Berlin</name> </country>
<country car_code="I"> <name>Rome</name> </country>
```

- country?[@car_code?]/city?[isCountryCap]/Name!text()

```
<country car_code="D"> <city>Berlin</city> </country>
<country car_code="I"> <city>Rome</city> </country>
```

XQL: Grouping

- übernimmt Teil der ursprünglichen Dokumentstruktur:
  $path_1 \; \{ \; path_2 \; \}$
- Ohne Gruppierung:
  country?[@car_code?]/city?/Name!text()

```
<country car_code="D"> <city>Berlin</city>
   </country>
<country car_code="D"> <city>Hamburg</city>
   </country>
<country car_code="D"> <city>Munich</city>
   </country>
```

- Mit Gruppierung:
  country?[@car_code?] {/city?/Name!text()}

```
<country car_code="D">
  <city>Berlin</city>
  <city>Hamburg</city>
  <city>Munich</city>
</country>
```


XQL: Joins

- Semi-Joins mittels Subqueries:

$$\pi[A](r \ltimes s) \; , \quad A \subset \text{attr}(r)$$

Gesucht: Der Name des Kontinents, auf dem Deutschland liegt:

```
/mondial/continent[@id =
                /mondial/country[car_code="D"]
                              /encompassed/@continent]
               /name!text()
```

Probleme

- man kann so nichts ausgeben, was in der Subquery im Filter berechnet wird.
- damit noch keine Full-Joins möglich

XQL: JOINS

- Unsymmetrische Full-Joins über *Korrelationsvariablen* und Alternative-Konstrukt: Im Filter können Variablenzuweisungen der Form

  [$\$var$:= $expr$]

  stehen, die dann irgendwo in einem anderen Filter

  [$expr'$ = $\$var$]

  wieder verwendet werden:

  //organization[$s := headq] {name | abbrev | member | //city[id=$s] }

```
<organization>
 <name>European Union</name>
 <abbrev>EU</abbrev>
 <member type="member" country="GR F E A D I B L NL DK SF S IRL P GB"/>
 <member type="membership applicant"
    country="AL CZ H SK LV LT PL BG RO EW M CY"/>
 <city> <name>Brussels</name> ... </city>
</organization>
```


XQL: FAZIT

- Ad-hoc-Konstrukte (in verschiedenen Versionen)
- halbherzige Restrukturierungs-Funkionalität
  - Baumstruktur der Eingabe wird prinzipiell beibehalten
- Probleme mit Joins,
- Ergebnisformat? Menge von Strings oder Elementen (SQL: Relation)
- nicht schachtelbar (vgl. SQL, OQL: Ergebnisse können direkt weiterverarbeitet werden)
- einer der Gründe: kein Variablenkonzept
- teilweise bis 2002 im Produkt “Tamino” der Software AG implementiert

6.2 Anfragesprachen: Anforderungen

Anforderungen an XML-Anfragesprachen [David Maier und W3C XML Query Requirements]

- Abgeschlossenheit: Ausgabe muss wieder XML sein
- Geschachtelte Query-Operationen: Selektion, Extraktion/Projektion, Restrukturierung, Kombination/Join, Verschmelzung von Elementen
- Ohne Kenntnis des Schemas anwendbar, kann aber bei Vorhandensein eines Schemas dieses nutzen,
- Ordnungserhaltend,
- Anfragen sollen in XML repräsentierbar sein, insbesondere sollen XML-Dokumente auch Anfragen enthalten können (Embedding)
- Verfolgen von XPointer und XLink
- Formale Semantik: Berechenbarkeit von Ergebnisstruktur, Äquivalenz und Query Containment (für Semantic Caching)
- Komponierbarkeit: überall, wo eine Menge von XML-Elementen erwartet wird, kann eine beliebige Anfrage stehen.

6.3 XML-QL

- `http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/`
- Einfache, Pattern-basierte XML-Anfragesprache:

  WHERE *xml-pattern* IN *url* CONSTRUCT *result*
- Verwendung von Variablenbindungen:
  xml-pattern enthält Variablen, die in *result* verwendet werden können,
- Deklarativ,
- “relational vollständig”, d.h. man kann Joins ausdrücken,

Beispiel:

```
WHERE <country car_code=$id>
        <name>$name</name>
      </country>
IN "www.../mondial.xml"
CONSTRUCT <country car_code=$id name=$name/>
```

XML-QL: Joins

Joins werden durch eine Liste von
WHERE ($expr_1$ IN doc_1, ... , $expr_n$ IN doc_n)-Klauseln formuliert:

- Equijoin innerhalb eines Dokuments:

```
WHERE
  <country car_code=$c></country>
  IN mondial.xml,
  <organization abbrev=$org>
    <membership type=$type country=$c/>
  </organization>
  IN mondial.xml,
CONSTRUCT ...
```


XML-QL: Joins

- Mehrere Quelldokumente:

```
WHERE
  <city name=$name2>
  IN www.../europe.xml,
  <city name=$name1>
  IN www.../america.xml,
  <connection from=$name1 to=$name2>
  IN www.../lufthansa.xml
CONSTRUCT
 <connection>
   <from continent=‘‘europe’’ city=‘‘name1’’/>
   <to continent=‘‘america’’ city=‘‘name2’’/>
 </connection>
```

XML-QL: Nested Queries

WHERE *xml-pattern* IN *url* CONSTRUCT *result*

- *result* kann wieder WHERE ... IN ... CONSTRUCT enthalten:

Weitere Features

- Tag-Variablen: WHERE <$tag> ... </>
- Regular Path Expressions: anstelle des // in XQL wird z.B. <*> ... </>geschrieben

Datenmodell

- Graph-basiert: XML-Baum mit gleichberechtigten IDREF-Kanten:

```
WHERE
  <country car_code=$cc>
    <capital><name>$name</name><population>$pop</population></capital>
  </country>
CONSTRUCT ...
```


XML-QL: Fazit

- Klauselbasierte Hochsprache
- Selektion und Konstruktion pattern-basiert
- Graph-Datenmodell; kein Unterschied zwischen Baumkanten und Referenzkanten
- Implementierung verfügbar
- in verschiedenen Projekten verwendet (z.B. MIX – Mediation in XML; UC San Diego)
  - Zugriff und Kombination verschiedener HTML/XML-Quellen in einer Anfrage möglich

6.4 Quilt und XQuery

- Ein Quilt ist ein Flickenteppich ...
- Beteiligt: IBM, Software AG, INRIA
- Literatur: WebDB2000-Workshop
- Wie SQL/OQL eine klauselbasierte, funktionale Sprache
  (beliebige Schachtelung von FLWR-Ausdrücken erlaubt),
- basiert auf XPath (vorher XQL/XSL Patterns) im Selektionsteil und auf XML-QL im Konstruktionsteil
- For Let Where Return-Klauseln

  for *variable* in *xpath-expr* // von XQL/XPath und XML-QL
  let *additional_variable* := *xpath-expr*
  where *filters*
  return *xml-expr* // von XML-QL

- wurde 2001 ziemlich unverändert als W3C XML Query übernommen.

XQUERY: BEISPIEL

- for-Klausel: Bindung von Variablen (vgl. SQL: FROM)
- where-Klausel: Auswertung von Bedingungen
- return-Klausel: Erzeugung der Ergebnisses (vgl. SQL: SELECT)

```
<result>
 { for $c in document("mondial-europe-2.0.xml")//country
   where $c/population > 10000000
   return <bigcountry>
            { $c/name }
            <area> { string($c/@area) } </area>
         </bigcountry>
 } </result>
```

erzeugt

```
<result>
  <bigcountry><name>France</name><area>547030</area></bigcountry>
  <bigcountry><name>Spain</name><area>504750</area></bigcountry>
  :
<result>
```

XQUERY: FUNKTIONSÜBERBLICK

- for-Klausel: Definiert geschachtelte Schleifen, wobei jede im for definierte Variable alle Werte durchläuft
- Variablen in XPath-Ausdrücken: gebunden in for/let (oder in der Umgebung bei eingebetteten Ausdrücken), verwendet als Anfangspunkte für Pfade und in Filtern
- Joins:
  - mehrere Variablen innerhalb eines FOR:
    for *$var*$_1$ in *doc*$_1$*/path*$_1$, ..., *$var*$_n$ in *doc*$_n$*/path*$_n$
  - Korrelierte Definition der Variablen innerhalb eines for:
    for *$var*$_1$ in *doc*$_1$*/path*$_1$, *$var*$_2$ in *$var*$_1$*/path*$_2$, ...
- let-Klausel zur Definition von "Konstanten":
  let *$var := expr*
  bindet *$var* an das *gesamte Ergebnis* von *expr*.
- geschachtelte/iterierte for-let-for-let-Klauseln möglich
- Erzeugung von geschachtelten gruppierten Strukturen:
  Die return-Klausel kann weitere FLWR-Klauseln enthalten (in denen auf die in der äußeren Klausel definierten Variablen zugegriffen werden kann).

EINFACHSTE XQUERY-ANFRAGEN: XPATH

- jede XPath-Anfrage ist auch eine XQuery-Anfrage
  Ergebnis: eine Sequenz von Knoten oder Literalwerten.

XQUERY: FOR-KLAUSEL

for $*var* = *xpath-expr*

- iteriert über das Ergebnis von *xpath-expr*

```
for $c in document('mondial-europe-2.0.xml')//country/name
return $c
```

[Filename: for-example.quip]

XQUERY: RETURN-KLAUSEL

Ausgabe soll immer XML sein.

- einfacher Fall: Ausgabe des Variableninhalts

```
for $c in document('mondial-europe-2.0.xml')//country/name
return $c
```

- und Erzeugen strukturierter Ergebnisse (vgl. OQL)

Strukturerzeugung

- literales XML
- berechnete Element- und Attributkonstruktoren (später)

Einbringen von berechneten Werten

- jeweils innerhalb "{" ... "}" geschachtelt
- Auswertung von Variablen und XPath-Ausdrücken
- geschachtelte FLWR-Klauseln

RETURN-KLAUSEL: KONSTRUKTION VON ERGEBNISELEMENTEN

- literales XML, Variableninhalte und XPath-Ausdrücke

```
<html><table>
<tr><th>Name</th><th>Flaeche</th><th>Bevoelkerung</th></tr>
{ for $c in document('mondial-europe-2.0.xml')//country
  return
    <tr><td>{$c/name/text()}</td>
        <td>{string($c/@area)}</td>
        <td>{$c/population/text()}</td>
    </tr>
}
</table></html>
```

[Filename: table-example.quip]

gibt für jedes Land eine Tabellenzeile aus.

XQUERY: FOR-KLAUSEL

Mehrere Variablen in einer for-Klausel

- klassisches kartesisches Produkt
  (vgl. FROM-Klausel in SQL)

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country,
      $o in document('mondial-europe-2.0.xml')//organization
  where $c/@capital = $o/@headq
  return
    <answer>
      <country>{$c/name/text()}</country>
      <organization>{$o/name/text()}</organization>
    </answer>
}
</result>
```

[Filename:cartesian-example.quip]

XQUERY: FOR-KLAUSEL

Mehrere Variablen in einer for-Klausel

- “korreliertes” Join
  (vgl. FROM-Klausel in Schema-SQL und OQL)
- Teilmenge des kartesischen Produkts

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country,
      $p in $c/province
  return
    <answer>
      <country>{$c/name/text()}</country>
      <prov>{$p/name/text()}</prov>
    </answer>
}
</result>
```

[Filename:correlated-join-example.quip]

RETURN-KLAUSEL MIT GESCHACHTELTER FLWR-KLAUSEL

- innere Anfrage im Ergebnis der äußeren (vgl. OQL)

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country
  where $c/province
  return
    <answer>
      {$c/name}
      { for $p in $c/province
        return
        <prov>{$p/name/text()}</prov>
      }
    </answer>
}
</result>
```

[Filename:nested-flwr-example.quip]

erzeugt für jedes Land das Provinzen hat, ein <answer>-Element, das ein <name>-Element und eine Folge von <prov>-Elementen enthält.

LET-KLAUSEL

let $*var* := *xpath-expr*

- iteriert nicht über das Ergebnis von *xpath-expr*
- sondern bindet das gesamte Ergebnis von *xpath-expr* als Sequenz von Knoten an die Variable:

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country
  let $cities := $c//city/name
  return
    <country>
      {$c/name}
      {$cities}
    </country>
} </result>
```

[Filename: let-example.quip]

- Sinnvoll zur Zwischenspeicherung wiederverwendbarer Zwischenergebnisse (was man in SQL oft vermisst)

BEDINGUNGEN

- Vergleiche: “==” für Knotenidentität, “<<” und “>>” für Dokumentordnung, FOLLOWS und PRECEDES
- Quantoren: where some|every $*var* in *expr* satisfies *predicate*

```
<result>
  { for $c in document("mondial-europe-2.0.xml")//country
    where some $city in $c//city satisfies $city/population > 1000000
    return $c/name
  }
</result>
```

```
<result>
  { for $c in document("mondial-europe-2.0.xml")//country
    where every $city in $c//city satisfies $city/population > 100000
    return $c/name
  }
</result>
```

[Filenamen: some-example.quip und every-example.quip]

BEDINGTE AUSWERTUNG UND ALTERNATIVEN

- if-then: Alternative Auswahl von Subelementen
  if (*expr*) then *expr* else *expr*

```
<result>
 { for $c in document("mondial-europe-2.0.xml")//country
   return
     <country>
       {$c/name}
       {if ($c/province) then $c/province/city else $c/city}
     </country>
  }
</result>
```

[Filename: if-else-example.quip]

SORTIEREN

- order by: *expr* order by (*expr* [ascending|descending])
  //country order by name
- ggf. muss man dem Interpreter erst mitteilen, dass Werte als Zahlen und nicht als Strings zu interpretieren sind!
  (sonst wird alphanumerisch sortiert)

```
<result>
  { for $c in //country
    order by int(population)
    return $c/name
  }
</result>
```

[Quip: nicht implementiert]

GRUPPIERUNG UND AGGREGIERUNG

- Aggregatfunktionen über Ergebnismengen (avg, sum)
- mit for-Schleife “Gruppierung” aufbauen,
- mit let und/oder Aggregatfunktionen verarbeiten

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country
  let $cities := $c//city
  where sum($cities/population) > 10000000
  return
    <answer>
      {$c/name}
      {sum($cities/population)}
    </answer>
}
</result>
```

[Filename: aggr-1-example.quip]

Gruppierung und Aggregierung

- Maximum-Bildung:
  auch hier muss man dem Interpreter erst mitteilen, dass Werte als Zahlen und nicht als Strings zu interpretieren sind!

```
<result>
{ for $c in document('mondial-europe-2.0.xml')//country
  let $maxpop := max( for $citypop in $c//city/population/text()
                      return int($citypop))
  return
    <answer>
       {$c/name}
       {$maxpop}
    </answer>
}
</result>
```

[Filename: aggr-2-example.quip]

Attribute in der RETURN-Klausel

- Beachte: Ausdrücke der Form “@bla” ergeben *Attributknoten* - diese sind (AttrName,Wert)-Paare:

```
<result>
  {document('mondial-europe-2.0.xml')//country[name='Germany']/@car_code}
</result>
```

  erzeugt <result car_code=“D”/>.

- Attributknoten werden immer dem umgebenden Element hinzugefügt
- wenn man nur ihren Wert haben will, muss man `string()` verwenden

```
for $c in document('mondial-europe-2.0.xml')//country
return
  <country>
    {$c/@area}
    {string($c/@car_code)}
  </country>
```

[Filename: attribut-example.quip]

BERECHNETE ELEMENT- UND ATTRIBUTNAMEN

- explizite Konstruktoren
  - ELEMENT {*expr*} {*attrs-and-contents*}
    Die Auswertung von *expr* ergibt den Namen des Elements, die anderen Knoten werden dann passend eingefügt.
  - Attribute {*expr*} {*expr-wert*}
    Die Auswertung von *expr* ergibt den Namen des Attributes, *expr-wert* seinen Wert.

```
<result>   <!-- NICHT GETESTET; in QuiP nicht implementiert! -->
 { for $c in document("mondial-europe-2.0.xml")//country[name="Germany"]//city
   return
   element { $c/name/text() }
           { attribute {$c/parent::province/name/text()} {"yes"}}
 } </result>
```

[Filename: computed-constructors-example.quip]

SONSTIGES

- Benutzerdefinierte (und in XQuery oder in Java/DOM) programmierte Funktionen
- mit Hilfe von document("http:...") kann man auf beliebige XHTML- oder XML-Resourcen im Web zugreifen.
- Mit XQueryX steht eine in XML formulierte Syntax zur Verfügung
- streng getypte Sprache (siehe W3C XML Query Formal Semantics; Datentypen des XML Schema und XML Query Data Model)
  - Jedem Ausdruck (und damit sein Ergebnis) kann ein Typ (der XML Schema-Datentypen) zugewiesen werden
  - damit kann auch jeder Variable ein Typ zugewiesen werden.
- weiteres siehe Doku und Bücher ...

XQUERY: FAZIT

Funktionalität

- kombiniert die positiven Erfahrungen der vorhergehenden Ansätze
- vermeidet deren Schwächen
- intuitiv klare Syntax und Semantik

Einschränkungen

- bisher keine Verfolgung von XLink/XPointer
  (siehe auch die "Schwäche" der id(.)-Funktion; keine Navigation entlang von Referenzattributen, in vielen Systemen (u.a. quip) nicht implementiert)
- nur eine *Anfragesprache*:
  Entscheidung des W3C: erst XQuery 1.0 als Anfragesprache konsistent mit XML Schema und Datenmodell zum Status einer "Recommendation" bringen, dann offizielle Gedanken zu Updates in XQuery 2.0.

AUFGABEN

... siehe Web.

Aufgabe 5
Bestimmen Sie den niedrigsten Berg, der der höchste Berg des Kontinents ist, zu dem er gehört.

Tun Sie dies für die relationale Mondial-DB in SQL, sowie für XML in XQuery. Probieren Sie, ob es auch mit XPath geht (evtl. erst ab 2.0 wegen den Aggregationsoperationen). □

6.5 XPathLog

- Prolog-/Datalog-artig, Weiterentwicklung von F-Logic (May, DBPL und VLDB 2001; TPLP 2003)
  - Pfadsyntax von *step.step.step* zu *step/step/step* geändert
  - Bedingungssyntax mit "[...]" beibehalten
  - F-Logic-Semantik (1989) fast identisch mit XPath-Semantik
  - neu: Unterschied zwischen Attributen/Subelementen
- Binden von Variablen an *beliebigen* Stellen im Ausdruck
- Joins durch Konjunktion (wie Prolog/Datalog)

XPathLog

- automatisches Auflösen von mehrwertigen Attributen
- automatisches Auflösen von Referenzattributen

```
?- //country->C[name->N and @membership->O/name->A].
```

- Zugriff auf Signatur/Metadaten

```
?- //country[name="Germany"]/M.
?- //country[name="Germany"]/@A.
```

- Klassenmitgliedschaft und -hierarchie

```
?- C isa country[name->N]/M.
?- _C isa country/@A->_O, _O isa X.
?- country[@M=>C].      % aus DTD
```

XPathLog: Kurzzusammenfassung

- deklarative Sprache
- Iteration implizit
- (Equi-)Joinvariablen
- analoge Semantik im Regelkopf zur *Erzeugung* und *Manipulation* von XML-Daten
- Fixpunktsemantik zur Regelausführung
- kann transitive Hülle berechnen

Weitere (akademische) Anfragesprachen

- XML-GL (Comai, Politecnico Milano, 1999): grafische “Sprache”
- Lorel-XML (Stanford etc, 1999): OQL-artige Sprache, Migration von Lorel
- YATL-XML (Cluet, INRIA, 2000): termbasierte Sprache, Migration von YATL
- Lixto/Elog (Gottlob, TU Wien): grafisches Extraktionstool mit Datalog-basierten Interna
- Xcerpt (Bry et al, LMU München, 2002): term- und unifikationsbasierte Sprache

... viele Sprachdesigns/-prinzipien führen zum Ziel (im wesentlichen in Europa)
kleiner Überblick in (May, TPLP 2003)