

XQuery-Anfragen

Seminar: Spezifikations- und Selektionsmethoden für Daten und Dienste

Markus Mauch

Universität Karlsruhe
Institut für Programmstrukturen und Datenorganisation (IPD)

Zusammenfassung Die *XML Query Language (XQuery)* ist eine vom World Wide Web Consortium (W3C) spezifizierte Anfragesprache für XML-basierte Datenbanken. XQuery ist eine funktionale Sprache, mit deren Hilfe insbesondere Daten aus XML-Dokumenten extrahiert und transformiert werden können. Die Ausdrucksmächtigkeit von XQuery umfasst Pfadausdrücke, Funktionsaufrufe, arithmetische, logische, bedingte Ausdrücke, Ausdrücke auf Mengen und Datentypen, sowie Ausdrücke zum Iterieren über Knotenmengen und zur Formulierung von Verbunden, usw. Das Typsystem von XQuery basiert auf XML-Schema und ist beliebig erweiterbar.

In dieser Arbeit werden die wesentlichen Grundlagen von XQuery erläutert, sowie die für XQuery-Anfragen benötigten Sprachkonstrukte vorgestellt. Darüber hinaus werden einige erweiterte Konzepte der Sprache betrachtet, die insbesondere das Verarbeitungskonzept und die Unterschiede von statischer Analyse und dynamischer Verarbeitung betreffen.

1 Einleitung

Für die Speicherung und den elektronischen Austausch von semi-strukturierten Daten spielt die *Extensible Markup Language (XML)* seit einigen Jahren eine wichtige Rolle. Aufgrund der selbst beschreibenden Natur von XML kann die Sprache flexibel eingesetzt und beliebig an individuelle Bedürfnisse angepasst werden. Die Möglichkeit, XML-Dokumente an ein Schema zu binden und gegen dieses zu validieren, qualifiziert XML als vollwertiges Datenmodell. Für den Zugriff auf XML-Datenbestände hat sich in den letzten Jahren XQuery als zentrale Anfragesprache herausgebildet. Die Sprache ist das Ergebnis einer mehrjährigen Entwicklung, die durch die Arbeit der *XML Query Working Group* des W3C vorangetrieben wird, und bei der von Beginn an ein wichtiges Entwurfsziel eine möglichst große Kompatibilität zu anderen XML-basierten Standards war. Dies zeigt sich insbesondere durch die enge Verwandtschaft zum *XPath-2.0*-Standard, der als Teilmenge von XQuery betrachtet werden kann.

Zu Beginn der Entwicklung von XQuery stand die Frage, ob für Anfragen im semi-strukturierten Datenmodell eine existierende Anfragesprache angepasst werden kann, oder eine völlig neue Anfragesprache entworfen werden sollte. Insbesondere die relationale Anfragesprache *SQL* hat sich in der Vergangenheit bewährt und eine Erweiterung dieses Standards für semi-strukturierte Daten wäre

in vieler Hinsicht vorteilhaft gewesen. Es wurde jedoch schnell offensichtlich, dass die Unterschiede zwischen relationalem und semi-strukturiertem Datenmodell so groß sind, dass eine Erweiterung nicht sinnvoll erscheint. Beispielsweise verfügen semi-strukturierte Daten im Gegensatz zu Daten im relationalen Modell über eine heterogene Struktur und beinhalten – als wesentliches Merkmal – ihre eigenen Metadaten. Gerade im semi-strukturierten Modell beziehen sich Anfragen oftmals auf eben solche Metadaten und eine entsprechende Anfrage im relationalen Modell würde Verbundoperationen über mehrere Relationen und Metadatenkataloge erforderlich machen. Ein weiterer Punkt ist, dass das Ergebnis einer Anfrage im relationalen Modell stets flach ist, während semi-strukturierte Daten häufig mehrere Ebenen von ineinander geschachtelten Datenelementen enthalten. Eine Anfragesprache für semi-strukturierte Daten muss jedoch in der Lage sein, ein solches hierarchisches Inhaltsmodell zu erzeugen. Darüber hinaus sind relationale Daten von Natur aus ungeordnet, während semi-strukturierte Daten über eine innere Ordnung verfügen, die zur Semantik beiträgt und nicht aus den Daten selbst abgeleitet werden kann. Diese und weitere Unterschiede waren letztlich ausschlaggebend dafür, dass mit XQuery eine völlig neue Anfragesprache mit eigenständigen Entwurfszielen vorgestellt wurde. Einige dieser Entwurfsziele sind im folgenden aufgeführt.

- **Zusammensetzbarkeit.** Überall dort, wo XQuery einen Ausdruck erwartet, soll jeder beliebige Ausdruck übergeben werden können. Es sollten keine syntaktischen Einschränkungen bei der Zusammensetzung von Ausdrücken existieren und keinerlei Nebeneffekte entstehen.
- **Abgeschlossenheit.** Jede XQuery-Anfrage soll als Eingabe ein XML-Dokument oder ein XML-Dokumentfragment erwarten, und als Ausgabe ein XML-Dokument oder -Fragment liefern.
- **Vollständigkeit.** XQuery soll *relational vollständig* sein, also über die Ausdrucksmächtigkeit der relationalen Algebra verfügen.
- **Schemakonsistenz.** XQuery übernimmt das Typsystem von XML Schema 1.0 und soll sich so genau wie möglich an diesen Standard anlehnen.

In Abschnitt 2 werden zunächst das Datenmodell und Typsystem von XQuery genauer betrachtet. Abschnitt 3 gibt einen Überblick über einfache XQuery-Ausdrücke, die auf der Basis des Typsystems weitgehend die Ausdrucksmächtigkeit der Sprache bestimmen. Dazu gehörten als wichtiger Baustein neben einigen weiterführenden Konzepten Pfadausdrücke und die so genannten FLWR-Ausdrücke. Ein weiterer Bestandteil von XQuery bilden die eingebauten und benutzerdefinierten Funktionen, auf die im Abschnitt 4 eingegangen wird. Abschnitt 5 behandelt erweiterte Konzepte von XQuery, insbesondere das interne Verarbeitungsmodell der Sprache und die Subsprache *XQuery-Core*. Abschließend wird eine Zusammenfassung sowie ein Ausblick auf die zukünftige Entwicklung von XQuery gegeben.

2 Datenmodell und Typsystem

Um weiterführende Sprachkonzepte von XQuery verstehen und anwenden zu können, ist zunächst ein Verständnis des zugrunde liegenden Datenmodells erforderlich. Die Aufgabe des Datenmodells besteht darin, die innere Struktur und die Beziehungen von Datenobjekten untereinander zu beschreiben und in einer geeigneten Form darzustellen. In XQuery repräsentiert eine Instanz des Datenmodells ein oder mehrere XML-Dokumente oder auch nur Fragmente eines solchen. Eine Anfrage bildet eine Instanz dieses Datenmodells auf eine andere ab. Insbesondere muss das Datenmodell also die Frage beantworten, welche Informationen innerhalb eines XML-Dokuments für die Anwendungsdomäne von XQuery relevant sind, und welche ignoriert werden können. In der Vergangenheit wurden bereits für viele andere XML-Anwendungen Datenmodelle vorgestellt, so zum Beispiel das *Document Object Model (DOM)*. Dieses besitzt als eine wesentliche Eigenschaft die Fähigkeit, die tatsächliche Struktur eines XML-Dokuments zu bewahren. Während eine solche Eigenschaft für einige Anwendungsdomänen sinnvoll ist, erscheint das damit verbundene hohe Maß an Differenziertheit für eine Anfragesprache wie XQuery nicht erforderlich. Auf der anderen Seite stellt XQuery ganz neue Anforderungen an ein Datenmodell. Beispielsweise können bei XQuery Zwischenergebnisse entstehen, für die keine offensichtliche Entsprechung in XML existiert, etwa Attribut- oder Dokumentlisten.

2.1 Struktur des XQuery-Datenmodells

Um solchen und ähnlichen Anforderungen gerecht zu werden, werden Instanzen des XQuery-Datenmodells als so genannte *Sequenzen* oder *Folgen* modelliert. Eine Folge ist das grundlegende Konstrukt des Datenmodells von XQuery und alle Ausdrücke operieren auf einer oder mehreren solcher Folgen und liefern wiederum eine Folge. Folgen bestehen aus null oder mehreren Einträgen, die *Items* genannt werden, und entweder *atomare Werte* oder *Knoten*, also komplexere Strukturen, sein können. Ein Item kann als einelementige Menge aufgefasst werden – XQuery unterscheidet nicht zwischen einer solchen Menge und dem Item selbst. Eine Folge von Items kann dadurch konstruiert werden, dass einzelne Items durch Kommata getrennt aneinander gereiht werden. Derartige Folgen können heterogen sein, d.h., sie können sowohl atomare Werte als auch Knoten enthalten, dürfen aber selbst nicht aus weiteren Folgen bestehen. Daher werden geschachtelte Folgen auf flache abgebildet – zum Beispiel ist die Folge $(0, (), (1, 2))$ äquivalent zu $(0, 1, 2)$. Folgen sind geordnet und besitzen keine Mengeneigenschaften, können also Duplikate enthalten.

Um den Datentyp von Folgen zu beschreiben, verwendet XQuery den **sequence type** Ausdruck. Mit Ausnahme des speziellen Typs `empty()`, der für leere Folgen reserviert ist, besteht ein solcher Ausdruck aus zwei Teilen, einem Typname und einem optionalen Häufigkeitsindikator. Als Typname kann jeder qualifizierende Name verwendet werden, aber auch jeder der in XQuery vordefinierten Datentypen, die im Folgenden erläutert werden. Der Häufigkeitsindikator gibt an, wie

oft ein Item des vorgegebenen Typs vorkommen darf und verwendet dazu in Analogie zu regulären Ausdrücken die Zeichen `+`, `*` und `?`.

2.2 Atomare Werte

Atomare Werte sind Instanzen eines einfachen Typs, d.h., sie besitzen keine innere Struktur und sind nicht weiter zerlegbar. XQuery besitzt eine Reihe vordefinierter atomarer Typen, die zu einem großen Teil dem Namensraum von *XML Schema 1.0* angehören. Diese Typen sind an ihrem Namensraumpräfix `xs` erkennbar, etwa `xs:integer` oder `xs:string`. Daneben führt XQuery aber auch eigene atomare Typen ein, die einem separaten Namensraum mit dem Präfix `xdt` angehören. Ein Beispiel für einen solchen atomaren Typ ist `xdt:untypedAtomic`, der atomaren Werten aus untypisierten XML-Dokumenten zugeordnet wird und in XQuery in etwa wie ein schwach typisierter String-Typ behandelt wird. Atomare Werte können auch einen benutzerdefinierten Typ haben, wenn dieser XQuery bekannt gemacht wurde. In Abbildung 1 wird das Zusammenspiel aller Datentypen in der XQuery-Typhierarchie illustriert. Die Typhierarchie besitzt keinen eindeutigen Obertyp, da ein großer Teil der XQuery-Datentypen direkt von XML Schema geerbt wird.

2.3 Knoten

Neben atomaren Werten sind Knoten die zweite Form der Ausprägung, die ein Item annehmen kann. XQuery kennt sieben unterschiedliche Knotentypen: `element`, `attribute`, `text`, `document-node`, `comment`, `processing-instruction` und `namespace`. Knoten besitzen im Gegensatz zu atomaren Werten eine Identität, weshalb sie eindeutig von gleichnamigen Knoten unterschieden werden können. Knoten vom Typ `element` oder `attribute` besitzen darüber hinaus einen Namen, der sich aus einem Namensraum-Präfix und dem lokalen Bezeichner des Knotens zusammensetzt. Jedes XML-Dokument oder Dokumentfragment wird im XQuery-Datenmodell als Baum dargestellt, der sich aus diesen Knotentypen zusammensetzt. Ein Dokument ist ein Baum, der als Wurzelknoten ein Element des Typs `document` besitzt. Im Gegensatz dazu besitzt ein Dokumentfragment als Wurzelknoten ein Element des Typs `element`. Alle Knoten (bis auf Dokument- und Namespace-Knoten) können Vorgängerknoten besitzen. Dokument- und Elementknoten können darüber hinaus auch über Nachfolgeknoten verfügen und somit eine Knotenhierarchie bilden. Alle Knoten einer Hierarchie unterliegen einer Ordnung, die *Dokumentreihenfolge* genannt wird. Diese ist konform zu der Reihenfolge, in der die Knoten in einem entsprechenden XML-Dokument auftreten würden. Abbildung 2 zeigt die Baumstruktur eines Beispieldokuments und die darin vorkommenden Knotentypen im XQuery-Datenmodell.

Jeder Knoten besitzt in XQuery einen textuellen Wert und ist konform zu dem generische Knotentyp `node()`. Einige Element- und Attributknoten besitzen darüber hinaus auch einen typisierten atomaren Wert. Als Beispiel sei ein Knoten vom Typ `element` gegeben. Der textuelle Wert dieses Knotens ist die Verkettung aller Textknoten, die dessen Nachfolger sind. Der typisierte Wert ergibt sich

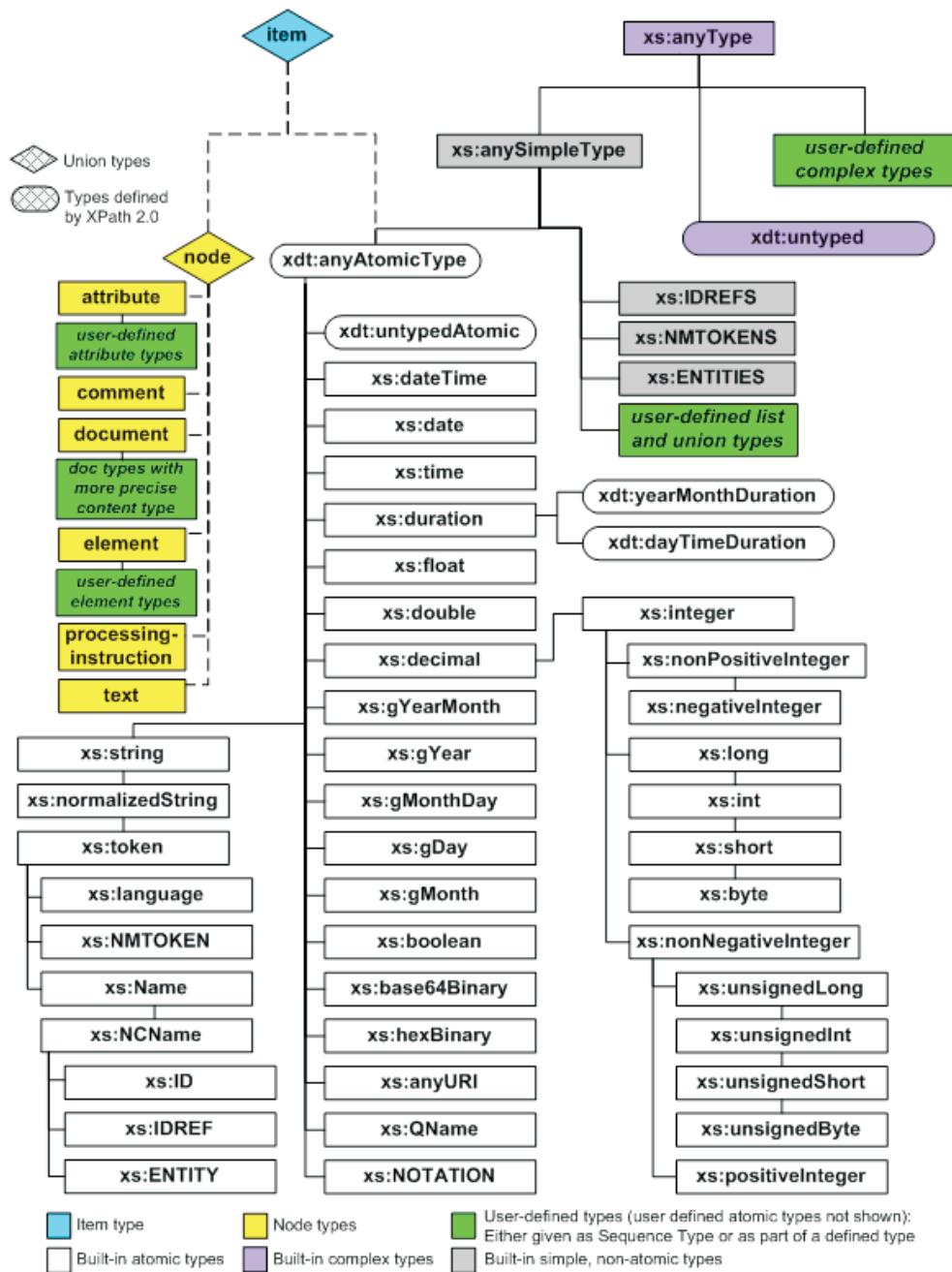


Abbildung 1. Typhierarchie in XQuery

aus der Validierung des textuellen Werts gegen die Typdefinition gemäß eines zugehörigen Schemas.

Dokument-, Element und Attributknoten können neben einem typisierten atomaren Wert auch eine zusätzliche strukturelle Typkomponente besitzen, falls sie einen oder mehrere Kindknoten enthalten. Sie besitzen dann entweder einen *einfachen* oder einen *komplexen Typ*, wie sie aus XML Schema bekannt sind. Einfache Typen bilden eine Obermenge der atomaren Typen und können im Allgemeinen nicht als Item in einer Folge verwendet werden. Komplexe Typen sind mit Ausnahme von `xs:anyType` und `xdt:untyped` benutzerdefinierte Typen, die in XML Schema definiert werden müssen.

Knoten sind im Allgemeinen stark typisiert – so würde beispielsweise ein Laufzeitfehler auftreten, falls eine Operation ein Attribut erwartete aber ein Element vorfände. In Bezug auf atomare Werte sind Knoten jedoch teilweise schwach typisiert. Es ist möglich, dass eine Operation mit einem Knoten umgehen kann, obwohl sie eigentlich einen atomaren Typ erwarten würde. Dies ist möglich, da XQuery in einem Vorgang, der Atomarisierung genannt wird, in bestimmten Fällen die atomaren Werte aus Knoten extrahieren kann. Dies ist deshalb erforderlich, da in XML keine atomaren Werte vorgesehen sind, sondern Daten immer als Knoten eines Baums dargestellt werden.

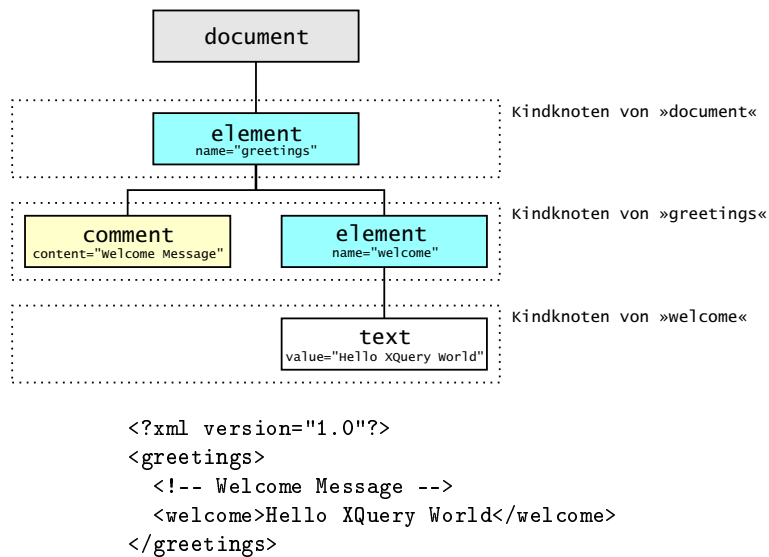


Abbildung 2. Einige Knotentypen im XQuery-Datenmodell.

3 XQuery-Ausdrücke

Eine Anfrage in XQuery besteht aus zwei Teilen, dem *Anfrageprolog* und dem *Anfragekörper*. Der Anfrageprolog enthält eine Reihe von Deklarationen, welche die Umgebung für die Anfrageverarbeitung bestimmen. Er wird nur benötigt, falls in der Anfrage externe Namensräume, Schemata oder Funktionen verwendet werden. Der Anfragekörper ist ein Ausdruck, der das Ergebnis der Anfrage spezifiziert. Die nachfolgenden Erläuterungen beziehen sich vor allem auf diesen Teil einer XQuery Anfrage.

3.1 Grundlagen

XQuery ist eine funktionale Sprache, d.h., jede Anfrage setzt sich zusammen aus einer Menge von Ausdrücken, die vollständig unabhängig voneinander sind und getrennt vom Rest der Anfrage ausgewertet werden können. Ausdrücke können mit Hilfe von Operatoren oder Schlüsselwörtern beliebig miteinander kombiniert werden. Somit kann überall dort, wo ein Ausdruck erwartet wird, jeder beliebige Ausdruck übergeben werden. Die einfachste Form eines Ausdrucks, ein so genanntes *Literal*, repräsentiert einen atomaren Wert. Literale sind also zum Beispiel Werte vom Typ `xs:integer`, `xs:double` oder `xs:string`. Andere atomare Werte können mit Hilfe von *Konstruktoren* erzeugt werden. So kann zum Beispiel ein atomarer Wert des Typs `xs:date` durch den Konstruktor `date('yyyy-mm-dd')` erzeugt werden.

Ausdrücke können *Operatoren*, *Variablen* und *Funktionsaufrufe* enthalten. Ein einfacher Operator in XQuery ist der *Komma-Operator*. Mit seiner Hilfe können zwei Werte zu einer Folge zusammengefasst werden. Darüber hinaus können Folgen auch durch den *to-Operator* erzeugt werden, der eine Folge numerischer Werte zurück gibt, die mit dem Wert des linken Operanden beginnt und von dort aus sukzessive alle Ganzzahlen bis einschließlich des Werts des rechten Operanden beinhaltet. Eine Variable ist in XQuery ein Name, der an einen Wert gebunden ist und in Ausdrücken verwendet werden kann, um dort diesen Wert zu repräsentieren. Eine weitere einfache Form von Ausdrücken sind Funktionsaufrufe. XQuery stellt eine Bibliothek von Kern-Funktionen bereit, auf die im Abschnitt 4.1 noch ausführlicher eingegangen wird. Außerdem können in XQuery benutzerdefinierte Funktionen verwendet werden, wie in Abschnitt 4.2 verdeutlicht. Funktionsaufrufe folgen in XQuery der in Programmiersprachen üblichen Notation, d.h., die Argumente einer Funktion werden dieser in einer Argumentliste übergeben.

3.2 Pfadausdrücke

Pfadausdrücke in XQuery basieren auf der Syntax von *XPath*. In XPath besteht ein Pfadausdruck aus einer Aneinanderreihung von *Lokalisierungsschritten*, den so genannten *Lokalisierungspfaden*. Ein Lokalisierungsschritt selektiert ausgehend von einer Menge von Referenzknoten eine Knotenmenge, die als Ausgangspunkt für den darauf folgenden Lokalisierungsschritt dient. Alle im letzten

Lokalisierungsschritt so selektierten Knotenmengen werden vereinigt und bilden das Ergebnis des Lokalisierungspfads.

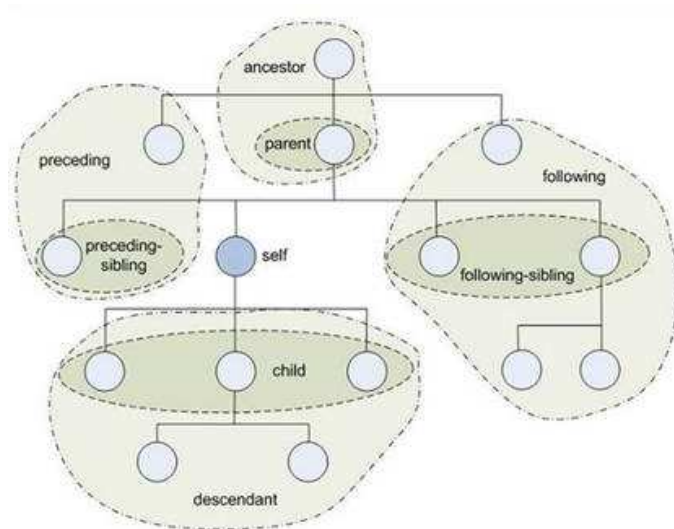


Abbildung 3. Visualisierung der XPath-Pfadausdrücke.

Jeder Lokalisierungsschritt besteht aus einer *Achse*, einem *Knotentest* und einem *Prädikat*. Die Achsen in einem Lokalisierungsschritt orientieren sich an der Baumstruktur eines XML-Dokuments und geben innerhalb des Baums eine Richtung an. Eine Achse kann einen, mehrere, oder überhaupt keinen Knoten als Ergebnis haben. Abbildung 3 zeigt die Ergebnisse der XPath-Achsen relativ zu einem ausgezeichneten Ausgangsknoten. Der Knotentest schränkt die Achse ein, so dass sie nur dann navigiert wird, falls ein Zielknoten diesen Test erfüllt – es werden also nur Knoten des angegebenen Namens berücksichtigt. Komplexere Bedingungen können als Prädikat formuliert werden, das zusätzlich zum Knotentest erfüllt sein muss, damit sich der über eine XPath-Achse erreichbare Zielknoten qualifiziert.

In XQuery sind Prädikate allgemein Ausdrücke, die aus einer Folge von Werten diejenigen heraus filtern, die das Prädikat wahr machen. Anfrage 1 illustriert einen Pfadausdruck, der über vier Lokalisierungsschritte verfügt. Im ersten Schritt wird die vordefinierte Funktion `document` aufgerufen, die den Dokumentknoten des Dokuments `items.xml` zurück gibt. Im zweiten Schritt werden über die `child`-Achse alle Kindknoten des Dokumentknotens ermittelt. Danach werden von diesen Knoten alle jene Kindknoten ausgewählt, die `item` heißen und wiederum selbst über Kindknoten verfügen, deren Name „seller“ lautet und die den Wert „Smith“ besitzen. Der vierte Schritt folgt der `child`-Achse erneut, um diejenigen Kindknoten der bisher gefundenen Referenzknoten zu finden, deren Name `description` lautet.

Anfrage 1. Beschreibungen aller Artikel, die von „Smith“ verkauft werden.

```
document('items.xml')/child:*/
child:item[child:seller='Smith']/
child:description
```

Neben dieser ausführlichen Notation existiert für XPath auch noch eine verkürzte Syntax, die kompakter und oft auch besser lesbar ist. So kann beispielsweise die Angabe der `child`-Achse vollständig entfallen, wenn man diese traversiert. Darüber hinaus existieren Abkürzungen für weitere Achsen, so zum Beispiel `.` und `..` für die `self`- bzw. `parent`-Achsen oder `//` für die `descendant-or-self`-Achse. Eine vollständige Übersicht über die verkürzte XPath-Syntax gibt Tabelle 1. Eine zu Anfrage 1 äquivalente Anfrage in verkürzter Syntax ist die folgende

Anfrage 2. Verkürzte Syntax für Anfrage 1.

```
document('items.xml')/*/item[seller='Smith']/description
```

Tabelle 1. Verkürzte XPath-Syntax.

Abk.	Bedeutung
<code>.</code>	Selektiert den aktuellen Referenzknoten (self).
<code>..</code>	Selektiert den unmittelbaren Vorgängerknoten des aktuellen Referenzknotens (parent).
<code>/</code>	Selektiert den Wurzelknoten eines Dokuments. Wenn das Zeichen innerhalb eines Pfadausdrucks vorkommt, dient es als Trennzeichen.
<code>@</code>	Selektiert die attribute des aktuellen Knotens (attribute).
<code>//</code>	Selektiert alle Nachfahren des aktuellen Knotens einschließlich des Referenzknotens (descendant-or-self).
<code>[n]</code>	Wenn das Prädikat nur aus einem Zahlenwert <i>n</i> besteht, dient es dazu, das <i>n</i> -te Element auszuwählen.

3.3 Elementkonstruktoren

Pfadausdrücke sind einerseits sehr mächtig, unterliegen andererseits jedoch der Einschränkung, dass mit ihrer Hilfe nur auf existierende Knoten zugegriffen werden kann. Für die Konstruktion neuer Elemente und Attribute sind sie allerdings nicht geeignet. Diese Aufgabe übernehmen in XQuery so genannte *Elementkonstruktoren*. Die einfachste Variante eines Elementkonstruktors ist das zu erzeugende XML-Fragment selbst. In diesem Fall sind die Werte und Namen der erzeugten Elemente und Attribute konstant. Oftmals ist es jedoch erforderlich, diese dynamisch durch eine Rechenvorschrift zu erzeugen. Um Werte dynamisch berechnen können, müssen die entsprechenden Ausdrücke in geschweifte Klammern gesetzt werden, wie das nachfolgende Beispiel verdeutlicht.

Anfrage 3. Elementkonstruktor mit konstanten Element- und Attributnamen.

```
<highbid status="{s}">
  <itemno>{i}</itemno>
  <bid-amount>
    {max($bids[itemno = $i]/bid-amount)}
  </bid-amount>
</highbid>
```

Diese Methode ist jedoch nicht dazu geeignet, die Namen von Elementen und Attributen dynamisch zu erzeugen. Aus diesem Grund stellt XQuery *berechenbare Elementkonstruktoren* bereit. Diese bestehen aus den Schlüsselwörtern **element** bzw. **attribute**, denen jeweils ein Ausdruck zur Berechnung des Element- bzw. Attributnamens und zur Berechnung des Element- bzw. Attributwerts nachfolgen. Das folgende Beispiel erzeugt ein neues Element, das den selben Namen hat und über die selben Attribute wie das Element verfügt, an das die Variable $\$e$ gebunden ist, jedoch einen doppelt so großen (numerischen) Wert besitzt.

Anfrage 4. Elementkonstruktor mit berechneten Element- und Attributnamen.

```
element {name($e)} {$e/@*, data($e * 2)}
```

3.4 FLWR-Ausdrücke

Die bisher vorgestellten Konzepte bilden lediglich die Grundlage für XQuery-Anfragen. Pfadausdrücke dienen zum Beispiel im Wesentlichen dazu, XML-Dokumente hinsichtlich eines bestimmten Kriteriums zu filtern. XQuery ist jedoch auch in der Lage, Dokumente beliebig zu transformieren und dadurch neue Dokumente zu erzeugen, die strukturell keine Verwandtschaft mit den Originaldokumenten aufweisen müssen. Das zentrale Grundkonstrukt zur Formulierung neuer Dokumente mit Hilfe von XQuery bildet das Konzept der **FLWR**-Ausdrücke. **FLWR** ist eine Abkürzung für **for**, **let**, **where** und **return** und ist unter dem Ausspruch *Flower* bekannt geworden. In einem **FLWR**-Ausdruck werden Folgen von Knoten durch die Bindung von Variablen in **for**- und **let**-Klauseln erzeugt. Die einzelnen Items dieser Folgen werden in der nachfolgenden **where**-Klausel

hinsichtlich einem vorgegebenen Prädikat gefiltert und gemäß der angegebenen Elementkonstruktoren in der `return`-Klausel ausgegeben, wodurch das Ergebnis im Sinne einer gültigen Instanz des XQuery-Datenmodells generiert wird.

Die Grammatik für gültige FLWR-Ausdrücke, wie sie nachfolgend auszugsweise gezeigt ist, verdeutlicht, dass diese an vielen Stellen XQuery-Ausdrücke (`ExprSingle`) erwarten:

```
FLWRExpr ::= (ForClause|LetClause)+ WhereClause? return ExprSingle
ForClause ::= for $VarName TypeDecl? PosVar? in ExprSingle
            (, $VarName TypeDecl? PosVar? in ExprSingle)*
LetClause ::= let $VarName TypeDecl? := ExprSingle
            (, $VarName TypeDecl? := ExprSingle)*
TypeDecl ::= as SequenceType
PosVar ::= as $VarName
WhereClause ::= where Expr
```

Da FLWR-Ausdrücke selbst XQuery-Ausdrücke sind, können Anfragen aus einer Vielzahl von ineinander geschichteten FLWR-Ausdrücken bestehen. In den folgenden Abschnitten werden nun die einzelnen Bestandteile eines FLWR-Ausdrucks genauer betrachtet.

for-Klausel Die einfachste Form der Bindung wird mit Hilfe der `for`-Klausel hergestellt. Diese ordnet einer Variable sukzessive die Elemente einer Item-Folge zu und wertet dann für jedes Item einen Ausdruck aus. In Anfrage 5 wird die Variable n zuerst an den Wert 2 und dann an den Wert 3 gebunden. Für jede Bindung wird der Ausdruck $n - 1$ ausgewertet. Das Ergebnis dieser Anfrage ist somit die Folge (1, 2). In einer `for`-Klausel können auch mehrere Variablen an die Elemente einer Folge gebunden werden. In diesem Fall werden stets Tupel von Variablenbindungen erzeugt, die dem kartesischen Produkt der Elemente der Folge entsprechen.

Anfrage 5. Sukzessives Binden einer Variablen an die Elemente einer Item-Folge.

```
for $n in (2, 3) return $n - 1
```

let-Klausel Die **let**-Klausel bindet im Gegensatz zur **for**-Klausel eine Variable einmalig an einzelne Werte oder an eine Menge von Werten, d.h. das Ergebnis eines Ausdrucks wird vollständig als Folge von Knoten und/oder Werten übernommen. Die unterschiedliche Semantik wird dabei bereits in der Grammatik deutlich, indem **let**-Klauseln mit **:=** an ein Ergebnis gebunden werden, während bei **for**-Klauseln die Grammatik die Verwendung des Schlüsselwortes **in** vorschreibt. Die Anfragen 6 und 7 demonstrieren die unterschiedliche Semantik der beiden Schlüsselwörter.

Anfrage 6. Verwendung des Schlüsselworts **for**.

```
for $x in (<a/>, <b/>, <c/>)
return
  <result>{$x}</result>

<result><a/></result>
<result><b/></result>
<result><c/></result>
```

Anfrage 7. Verwendung des Schlüsselworts **let**.

```
let $x := (<a/>, <b/>, <c/>)
return
  <result>{$x}</result>

<result><a/><b/><c/></result>
```

where-Klausel Mit Hilfe der **where**-Klausel kann für jeden gebundenen Wert ein vorgegebener Ausdruck ausgewertet werden. Wird der Ausdruck einer **where**-Klausel mit der gerade aktuellen Variablenbelegung zu **true** ausgewertet, so wird die Ausführung der **where**-Klausel angestoßen. Andernfalls wird die gerade gültige Belegung verworfen. Das Ergebnis der Auswertung entspricht dabei dem *effektiven booleschen Wert* des jeweiligen Ausdruck, d.h. die Auswertung liefert **false**, wenn die Auswertung des Ausdrucks eine leere Folge oder **false** ergibt, ansonsten **true**.

return-Klausel Die **return**-Klausel eines FLWR-Ausdrucks gibt eine Berechnungsvorschrift an, nach der das Ergebnisdokument konstruiert werden soll. Falls die aktuelle Variablenbindung die Bedingung der **where**-Klausel erfüllt, wird das berechnete Ergebnis eines Ausdrucks zurückgegeben. Dabei muss die Dereferenzierung von Variablen durch einen Auswertungskontext explizit angeregt werden. Dies geschieht durch die Einbettung der Variablen in geschweifte Klammern.

3.5 Verbunde

Eine Verbundoperation wird in XQuery als geschachtelte **for**-Schleife realisiert, wobei in der **for**-Klausel die Laufvariablen gebunden, in der **where**-Klausel das Verbundprädikat ausgewertet und in der **return**-Klausel die durch die Variablen identifizierten Dokumentbestandteile herangezogen werden, um ein Fragment des verknüpften Gesamtdokuments zu erzeugen. Anfrage 8 zeigt einen einfachen symmetrischen Verbund, der aus jeweils einer Menge von Abteilungen und Mitarbeitern einer Firma, für jede Abteilung den Namen des Mitarbeiters ermittelt, der die Abteilung leitet. Zuerst wird dabei in der **for**-Klausel das kartesische Produkt beider Mengen gebildet. In der **where**-Klausel werden anschließend nur jene Tupel übernommen, deren Werte für die Abteilungsleiter- bzw. Mitarbeiter-Kennzahl übereinstimmen. Anschließend wird in der **where**-Klausel ein entsprechendes Ergebnis konstruiert. Das Problem der „doppelten Spalten“ das bei einem natürlichen Verbund in der relationalen Algebra auftritt, wird bei XQuery konstruktiv gelöst, indem das Ergebnis durch eine nahezu beliebige freie Struktur des Ergebnisdokuments formuliert werden kann.

Anfrage 8. Symmetrischer Verbund.

```
for $d in $departments/department, $e in $employees/employee
where $d/manager = $e/ID
return
  <department>
    {$d/name}
    <management>
      {$e/forename}
      {$e/lastname}
    </management>
  </department>
```

Alternativ hätte das Verbundprädikat, das in diesem einfachen Fall ein Test auf Gleichheit ist, in der **where**-Klausel auch entfallen können und in die innere Schleife hineingezogen werden können. Die entsprechende **for**-Klausel würde dann wie folgt lauten:

```
for $d in $c//department, for $e in $c//employee[@ID = $d/@management]
```

Diese und andere Optimierungen sind Gegenstand der Anfrageverarbeitung, die ein XQuery-Prozessor durchführt.

In Anfrage 8 wurde vorausgesetzt, dass jede Abteilung über einen Abteilungsleiter verfügt. Abteilungen, für die dies nicht zutrifft, tauchen im Ergebnisdokument allerdings nicht auf. Um dies zu erreichen, ist es erforderlich, einen äußeren Verbund zu definieren, bei dem alle Abteilungen in das Ergebnisdokument übernommen werden, unabhängig davon, ob sie einen Abteilungsleiter besitzen, oder nicht. Dazu muss in einem äußeren FLWR-Ausdruck zunächst die Abteilungsinformation erzeugt werden und dann in einem inneren FLWR-Ausdruck die (möglicherweise leere) Liste der Entsprechenden Abteilungsleiter hinzugefügt werden. Dies wird in Anfrage 9 illustriert.

Anfrage 9. Äußerer Verbund.

```
for $d in $departments/department
return
  <department>
    {$d/name}
    <management>
    {
      for $e in $employees/employee
      where $d/manager = $e/ID
      return ($e/forename, $e/lastname)
    }
  </management>
</department>
```

3.6 Typausdrücke

Innerhalb vieler XQuery-Ausdrücke ist es erforderlich, auf bestimmte Datentypen zu verweisen. Dies kann durch die Verwendung eines qualifizierenden Namens geschehen. Zum Beispiel verweist `xs:string` auf einen vordefinierten Zeichenkettentyp, `abc:address` könnte einen Verweis auf den Datentyp eines externen Schemas darstellen, usw. Weitere Typverweise stellen die Schlüsselwörter `element` und `attribute` dar, denen optional ein qualifizierender Name folgen kann, der den Datentyp eines Knotens weiter einschränkt. Das Konstrukt `element <name>` kennzeichnet beispielsweise einen Elementknoten mit dem angegebenen Namen, `element of type <data type>` deklariert ein Element des gegebenen Typs, `node` und `item` repräsentieren die generischen Typen für beliebige Knoten bzw. Items. Häufigkeitsindikatoren können verwendet werden, um Anzahl und Vorkommen von Knoten eines bestimmten Typs genauer zu spezifizieren.

Typverweise werden in XQuery häufig für Funktionsdeklarationen verwendet, treten aber auch an vielen anderen Stellen in Erscheinung. So gibt der binäre Operator `instance of` den Wert `true` zurück, wenn sein erstes Argument eine Instanz des im zweiten Argument übergebenen Datentyps ist. Gelegentlich ist es auch erforderlich, einen Ausdruck abhängig von seinem dynamischen Laufzeittyp auszuwerten. Ein Beispiel dafür sind Postadressen, die aufgrund unterschiedlicher Bestimmungsorte unterschiedliche Typen annehmen können. XQuery stellt

hierzu den `typeswitch`-Ausdruck bereit, der abhängig vom Datentyp eines übergebenen Ausdrucks verschiedene `case`-Klauseln zur Ausführung bringt. Typnamen werden auch in `cast`-Ausdrücken verwendet, mit deren Hilfe der Ergebnistyp eines Ausdrucks explizit in einen anderen Typ umgewandelt werden kann. Zum Beispiel kann das Ergebnis von `$x div 5` durch den Ausdruck `cast as xs:double($x div 5)` in den Datentyp `xs:double` umgewandelt werden. Falls eine Umwandlung nicht durchgeführt werden kann, tritt ein Laufzeitfehler auf.

Häufig ist in einem komplexen XQuery-Ausdruck nicht offensichtlich, welchen Typ ein Ausdruck hat. Besonders problematisch ist dies bei der statischen Analyse einer Anfrage, bei der keine konkreten Daten verwendet werden. In solchen Fällen ist eine strikte Prüfung erst dynamisch zur Laufzeit möglich. Um diesem Problem zu begegnen stellt XQuery den Ausdruck `treat as` bereit, mit dem eine Zusicherung über den Typ eines Ausdrucks erfolgen kann. `treat as` weist seinem Operator bei der Übersetzung der Anfrage einen spezifischen Datentyp zu, der für die statische Typüberprüfung verwendet wird. Zur Laufzeit wird jedoch ein Fehler aufgeworfen, falls der dynamische Datentyp des Ausdrucks nicht mit dem statischen Typ übereinstimmt. Neben `treat as` existiert in XQuery auch noch der Ausdruck `assert`, der dem Anfrageprozessor zugesichert, dass ein Ausdruck einen bestimmten statischen Datentyp besitzt.

3.7 Bedingte und quantifizierende Ausdrücke

Ein bedingter Ausdruck stellt zwei mögliche Ausführungen einer Anfrage bereit. XQuery verwendet für bedingte Ausdrücke die aus vielen Programmiersprachen bekannte `if...then...else`-Syntax, wobei keine dieser Klauseln optional ist. Das Ergebnis eines solchen Ausdrucks hängt vom Wert eines gegebenen Testausdrucks ab. Falls der Testausdruck zu `true` evaluiert, oder eine Folge mit mindestens einem Item ist, wird die `then`-Klausel ausgeführt. Ergibt die Auswertung des Testausdrucks `false` oder eine leere Folge, dann kommt die `else`-Klausel zur Ausführung.

Quantifizierende Ausdrücke ermöglichen die Überprüfung, ob eine Bedingung für mindestens ein Item einer Folge bzw. für alle Items einer Folge erfüllt ist. Das Ergebnis eines quantifizierenden Ausdrucks ist daher stets `true` oder `false`. Quantifizierende Ausdrücke iterieren ähnlich wie FLWR-Ausdrücken über die Items einer Folge, wobei sukzessive eine Bindung jedes Items an eine Variable erfolgt. Für jede Variablenbindung wird anschließend ein Testausdruck ausgewertet. Quantifizierende Ausdrücke verwenden die Schlüsselwörter `some`, `every` und `satisfies`. `some` gibt `true` zurück, falls der Testausdruck für mindestens eine Variablenbindung zu `true` evaluiert, wie im folgenden Beispiel verdeutlicht:

Anfrage 10. Verwendung von `some`.

```
some $n in (5, 7, 9) satisfies $n > 10
```

Ein quantifizierender Ausdruck, der mit `every` beginnt, gibt `true` zurück, falls der gegebene Testausdruck für alle Variablenbindungen zu `true` ausgewertet wird. So liefert beispielsweise der Ausdruck in Anfrage 11 `false`, da der Testausdruck für einige, jedoch nicht für alle Bindungen zu `true` evaluiert.

Anfrage 11. Verwendung von `some`.

```
every $n in (5, 7, 9) satisfies $n > 10
```

4 Funktionen

Häufig wiederkehrende Operationen werden in Programmiersprachen in der Regel nicht mehrfach implementiert, sondern in Form von Prozeduren und Funktionen zusammengefasst. Auch XQuery bietet die Möglichkeit, Funktionen zu verwenden. Neben einer Reihe vordefinierter Funktionen können in XQuery auch benutzerdefinierte Funktionen in Anfragen verwendet werden. Der erste Teil dieses Abschnitts beschäftigt sich mit den vordefinierten Funktionen von XQuery, von denen einige bereits implizit verwendet wurden. Im zweiten Teil wird mit Hilfe einiger Beispiele illustriert, wie benutzerdefinierte Funktionen in XQuery verwendet werden können.

4.1 Vordefinierte Funktionen

XQuery verfügt über eine Vielzahl an vordefinierten Funktionen, die ein breites Anforderungsspektrum abdecken. Insbesondere stellt die Sprache Funktionen für numerische und boolesche Werte, Zeichenkettenfunktionen, Zeit- und Kalenderfunktionen, Funktionen für Knoten und Folgen, Aggregatfunktionen und viele andere mehr zur Verfügung. Alle vordefinierten Funktionen gehören dem Namensraum <http://www.w3.org/2004/07/xpath-functions> an und werden in W3C-Spezifikationen stets mit dem Namensraum-Präfix `fn` versehen. Einige dieser Funktionen wurden bereits im Verlauf dieser Arbeit verwendet – eine Übersicht über diese und einige weitere ausgewählte XQuery-Funktionen gibt Tabelle 2.

4.2 Benutzerdefinierte Funktionen

Benutzerdefinierte Funktionen in XQuery bestehen aus zwei Teilen, einem *Funktionskopf* und einem *Funktionsrumpf*. Der Funktionskopf repräsentiert die Signatur der Funktion und besteht aus einem Funktionsbezeichner, einer Liste von Parametern (optional mit korrespondierendem Datentyp) und der optionalen Angabe eines Datentyps für den Rückgabewert der Funktion. Der Rumpf

Tabelle 2. Einige vordefinierte XQuery-Funktionen.

Name	Beschreibung
<code>fn:number(arg)</code>	Gibt den numerischen Wert des Arguments zurück.
<code>fn:abs(num)</code>	Gibt den absoluten Wert des Arguments zurück.
<code>fn:string(arg)</code>	Wandelt das Argument in eine Zeichenkette um.
<code>fn:concat(string,string,...)</code>	Konkateniert die übergebenen Zeichenketten.
<code>fn:contains(string1,string2)</code>	Gibt <code>true</code> zurück, falls <code>string2</code> in <code>string1</code> enthalten ist, ansonsten <code>false</code> .
<code>fn:avg(arg,arg,...)</code>	Gibt den Durchschnittswert der übergebenen numerischen Werte zurück.
<code>fn:sum(arg,arg,...)</code>	Gibt die Summe der übergebenen numerischen Werte zurück.
<code>fn:empty(item,item,...)</code>	Gibt <code>true</code> zurück, falls die übergebene Folge leer ist, ansonsten <code>false</code> .
<code>fn:data(item,item,...)</code>	Erwartet eine Folge von Items und gibt eine Folge atomarer Werte zurück.
<code>fn:node-name(node)</code>	Gibt den Namen des Argumentknoten zurück.
<code>fn:namespace-prefix()</code>	Gibt das Namensraum-Präfix zurück.

einer Funktion besteht entweder aus einem XQuery-Ausdruck oder einer Referenz auf eine externe Realisierung der Funktion. Wird eine Funktion aufgerufen, werden die Argumente des Funktionsaufrufs an die Parameter der Funktion gebunden und der Funktionsrumpf ausgeführt. Falls für die Funktionsparameter keine Datentypen spezifiziert wurden, nimmt die Funktion Parameter jedes Typs an. Umgekehrt kann eine Funktion jeden beliebigen Datentyp als Funktionswert zurückgeben, falls kein spezieller Rückgabebetyp deklariert wurde. Wenn die Datentypen der Parameter einer Funktion explizit deklariert sind, müssen die übergebenen Argumententypen mit diesen übereinstimmen. Jedoch können numerische Funktionsargumente in einen entsprechenden Typ umgewandelt werden, falls dies zur XQuery-Aufwertungshierarchie `integer` \rightarrow `decimal` \rightarrow `float` \rightarrow `double` konform ist. Darüber hinaus sind Argumente erlaubt, deren Datentypen von den Datentypen der Funktionsparameter (z.B. durch Bildung von Subtypen) abgeleitet werden können. Wird eine Funktion, die als Parameter einen atomaren Wert erwartet, mit einem Knotentyp als Argument aufgerufen, so wird der typisierte Wert des Knotens ermittelt und der Funktion übergeben.

Im Folgenden wird die Verwendung benutzerdefinierter Funktionen in XQuery anhand einiger ausgewählter Beispiele illustriert. Anfrage 12 zeigt Definition und Aufruf einer Funktion, die in einem Online-Auktions-Szenario für einen gegebenen Artikel aus einer Menge von Geboten das höchste abgegebene Gebot ermittelt.

Anfrage 12. Einie einfache Funktionsdefinition.

```
define function highbid(element $item) returns xs:decimal
{
  max(document("bids.xml")//item[itemno = "1234"]/bid-amount)
}

highbid(document("items.xml")//item[itemno = "1234"])
```

Die Funktion in Anfrage 13 erwartet zwei Parameter, ein optionaler Elementknoten und einen beliebigen Default-Wert. Falls der Funktion ein Elementknoten übergeben wird, der einen typisierten Wert enthält, wird dieser zurückgegeben. Andernfalls gibt die Funktion den Default-Wert zurück.

Anfrage 13. Einie einfache Funktionsdefinition.

```
define function defaulted(element? $e, anySimpleType $d) returns anySimpleType
{
  if (empty($e)) then $d
  else if (empty($e/*)) then $d
  else data($e)
}
```

XQuery erlaubt auch die Verwendung von rekursiven Funktionen. Rekursion ist ein sehr mächtiges Hilfsmittel, insbesondere, wenn sie zusammen mit einem hierarchischen Datenmodell verwendet wird, wie es XML darstellt. Ein Beispiel für eine rekursive Funktion in XQuery gibt die folgende Anfrage, welche die Tiefe einer Elementhierarchie ausgehend von ihrem Argument ermittelt.

Anfrage 14. Rekursive Funktion.

```
define Function depth(element $item) returns xs:integer
{
  if (empty($e/*)) then 1
  else 1 + max(for $c in $e/* return depth($c))
}
```

5 Erweiterte Konzepte

Für ein tieferes Verständnis von XQuery ist es erforderlich, das interne Verarbeitungsmodell und die Auswertungsumgebung der Sprache genauer zu untersuchen. In der Regel ist die Auswertung einer Anfrage in mehrere Teilabschnitte untergliedert. In der ersten Phase erfolgt die so genannte *statische Analyse* der

Anfrage. Im Rahmen der statischen Analyse kann eine XQuery-Implementierung bereits konstante Ausdrücke berechnen und mit Hilfe von Typableitungsregeln Typzuweisungen und -prüfungen vornehmen. Wenn die statische Analyse keine Fehler festgestellt hat, folgt die *dynamische Ausführung* der Anfrage, zu der nun auch die Eingabedaten herangezogen werden. In diesem Abschnitt werden diese und weitere Details der Anfrageverarbeitung in XQuery genauer untersucht. Dazu muss jedoch zunächst eine exakte Definition der Semantik von XQuery gegeben werden.

5.1 XQuery-Core

Neben der eigentlichen Empfehlung für den XQuery-Standard hat das W3C unter dem Titel *XQuery Formal Semantics* ein Dokument veröffentlicht, in dem die Spezifikation von XQuery durch eine eindeutige Beschreibung der Semantik der Sprache ergänzt werden soll. Das Dokument spezifiziert ein Fragment von XQuery, welches unter dem Namen *XQuery-Core* bekannt geworden ist. XQuery-Core besitzt die selbe Ausdrucksmächtigkeit, über die auch XQuery verfügt, ist jedoch gemessen an dessen Umfang relativ überschaubar. In XQuery-Core werden XQuery-Ausdrücke durch Objekte repräsentiert und die Semantik der Ausdrücke wird mit Hilfe einer systematischen Definition der Beziehungen dieser Objekte zueinander beschrieben. Die statische Semantik von Ausdrücken wird dabei durch Typableitungsregeln bestimmt, die dynamische Semantik durch Zuordnungen zwischen XQuery-Ausdrücken und den XML-Werten, zu denen diese evaluiert werden können. Darüber hinaus definiert das Dokument die Abbildung von XQuery auf XQuery-Core und somit eine exakte formale Semantik für XQuery als Ganzes.

Beim Entwurf von XQuery-Core wurde besonderen Wert auf die Vollständigkeit und Korrektheit der Sprache gelegt. XQuery-Core beinhaltet Mechanismen zur Projektion, Selektion, sowie Mengenoperationen und ist nachweisbar relational vollständig. Die statische Typüberprüfung ist in der Lage, aus einer gegebenen Anfrage und einem Eingabeschema ein entsprechendes Ausgabeschema abzuleiten. Außerdem kann der Ergebnistyp einer Anfrage statisch gegen ein gegebenes Ausgabeschema validiert werden. XQuery-Core führt keine impliziten Typumwandlungen durch, wie sie häufig in XPath aber auch in XQuery vorkommen. Zum Beispiel tritt im Gegensatz zu XQuery ein statischer Fehler auf, wenn ein Elementknoten mit einem atomaren Wert verglichen wird. Auf diese Weise bleibt die formale Spezifikation der statischen und dynamischen Semantik überschaubar, und die Korrektheit der Sprache nachweisbar.

5.2 Das Verarbeitungsmodell von XQuery

Die XQuery-Spezifikation definiert nicht nur eine Sprache und ein Datenmodell, sondern auch ein Verarbeitungsmodell für die Sprache. Dieses Modell bestimmt das Verhalten einer XQuery-Implementierung bei der Ausführung einer Anfrage. Abbildung 4 gibt eine leicht vereinfachte grafische Darstellung des Verarbeitungsmodells von XQuery. Die Anfragebearbeitung erfolgt in vier Schritten.

Im ersten Schritt prüft ein Parser die Anfrage, das Eingabedokument und – falls vorhanden – das Eingabeschema auf syntaktische Korrektheit und erzeugt aus diesen Eingaben einen Operatorbaum, eine XQuery-Datenmodellinstanz und einen Eingabe-Typbaum. Im nächsten Schritt wird die Anfrage übersetzt. Dabei wird die Datenmodellinstanz gegen das Eingabeschema validiert und jedem Element ein Typvermerk zugeordnet, der dessen Zugehörigkeit zu einem bestimmten Datentyp anzeigt. Elemente, die keinem Datentyp zugeordnet werden können, erhalten den generischen Typvermerk `anyType`. Darüber hinaus wird der Operatorbaum mit Hilfe der Übersetzungsregeln auf einen Operatorbaum von XQuery-Core abgebildet. Die statische Semantik von XQuery-Core beschreibt, wie aus diesem Operatorbaum und dem Eingabetyp der Ausgabebetyp der Anfrage abgeleitet wird. Im darauf folgenden Schritt wird die Anfrage ausgewertet und ein Anfrageergebnis als Instanz des XQuery-Datenmodells konstruiert. Dies wird durch die dynamische Semantik von XQuery-Core beschrieben. Im letzten Schritt wird das Ergebnis der Anfrage in einem Prozess, der *Serialisierung* genannt wird, wieder in eine entsprechende XML-Repräsentation umgewandelt. Zusätzlich wird aus dem Ausgabe-Typbaum eine XML-Schemainstanz erzeugt.

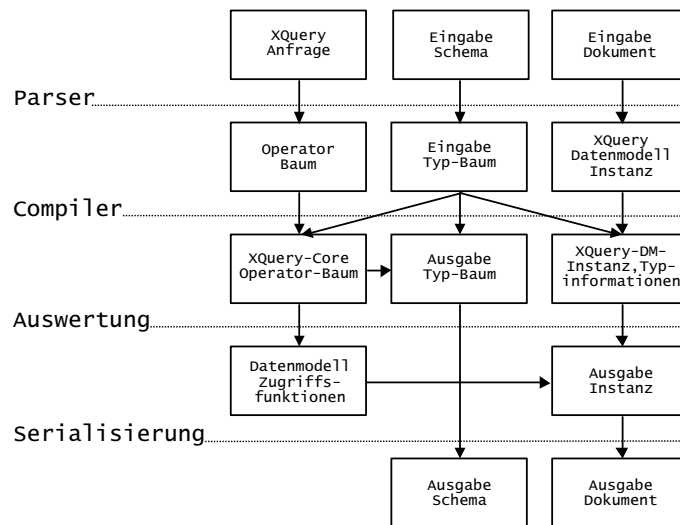


Abbildung 4. Das Verarbeitungsmodell von XQuery.

5.3 Abbildung von XQuery auf XQuery-Core

In diesem Abschnitt soll nochmals anhand der Anfrage 8 aus Abschnitt 3.5 die Abbildung einer XQuery-Anfrage auf XQuery-Core gezeigt werden. Dazu wird angenommen, dass jede Abteilung genau einen Namen und null oder mehrere

Abteilungsleiter besitzen kann. Der Datentyp, der aus der Verbundoperation aus Anfrage 8 abgeleitet werden kann, lautet dann folgendermaßen:

```

ELEMENT department {
  ELEMENT name {xs:string},
  ELEMENT management {
    ELEMENT forename {xs:string},
    ELEMENT lastname {xs:string}
  }*
}*

```

Für eine entsprechende Anfrage in XQuery-Core muss zunächst der Pfadausdruck `$departments/department` in eine geschachtelte `for`-Schleife umgewandelt werden, da XQuery-Core Pfadausdrücke nicht direkt unterstützt:

```

FOR $v1 IN $departments RETURN
  FOR $v2 IN NODES($v1) RETURN
    TYPESWITCH ($v2) AS $v3
      CASE ELEMENT department {ANYTYPE}
        RETURN $v3
      DEFAULT RETURN ()

```

Abhängig vom Vorhandensein von Schemainformationen kann eine solche Anfrage statisch optimiert werden. Falls zum Beispiel bekannt ist, dass `$departments` nur ein `<departments>`-Element enthält, kann die äußere `for`-Schleife entfallen. Enthält `$departments` darüber hinaus ausschließlich `<department>`-Elemente, wird die `DEFAULT`-Klausel niemals verwendet und kann ebenfalls entfallen. Zusammen führt dies zu der Anfrage

```

FOR $v2 IN NODES ($departments) RETURN $v2

```

und schließlich zu `NODES ($departments)`. Analog kann mit dem Pfadausdruck `$employees/employee` verfahren werden. In einem weiteren Schritt kann nun die `where`-Klausel auf XQuery-Core abgebildet werden. Aus XPath 1.0 erbt XQuery die implizite Existenzquantifizierung von Prädikaten. So wird der Ausdruck `$d/manager = $e/ID` wahr, falls `$d` ein Element `manager` besitzt, dessen Inhalt gleich einem Element ID von `$e` ist. In XQuery-Core würde ein solches Prädikat wie folgt aussehen, wobei `[[...]]` analog zu den oben gezeigten Regeln abgeleitet wird:

```

NOT ( EMPTY (
  FOR $v1 IN [[ $d/manager ]] RETURN
  FOR $v2 IN [[ $e/ID ]] RETURN
  IF EQ($v1, $v2) THEN $v1 ELSE ()
) )

```

Falls ein Schema zu den Abteilungs- und Mitarbeiterdatenbanken vorgibt, dass die Elemente `manager` und `ID` jeweils nur einmal vorkommen dürfen, kann die Anfrage statisch weiter optimiert werden:

```

NOT ( EMPTY (
  IF EQ([[ $d/manager ]], [[ $e/ID ]])
  THEN [ $d/manager ] ELSE ()
) )

```

Dies ist offensichtlich analog zu der Anfrage `EQ([[$d/manager]], [[$e/ID]])`. In einem letzten Schritt muss schließlich noch die `for`-Klausel, die die Variablen `$d` und `$e` bindet, in eine geschachtelte Schleife umgewandelt werden, bei der zuerst `$d` und anschließend `$e` gebunden werden. Zusätzlich muss die `where`-Klausel in eine `if...then...else`-Klausel umgewandelt werden:

```

FOR $d IN NODES($departments) RETURN
  FOR $e IN NODES($employees) RETURN
    IF (EQ([[ $d/manager ]], [[ $e/ID ]])) THEN
      ELEMENT department {
        [[ $d/name ]],
        ELEMENT management {
          [[ $e/forename ]],
          [[ $e/lastname ]]
        }
      }
    }

```

6 Zusammenfassung und Ausblick

Seit der ersten Veröffentlichung von XQuery wurde die Sprache ständig verbessert und ihre Weiterentwicklung vorangetrieben. Gegenwärtig besitzt XQuery den Status einer *Candidate Recommendation*, was andeutet, dass die Spezifikation von XQuery bereits für Implementierungen freigegeben ist. Deren zunehmende Anzahl und die Aufnahme von XQuery-Funktionalität in marktbeherrschende Datenbanksysteme lassen kaum einen Zweifel, dass die Sprache in absehbarer Zeit den Status einer vollwertigen W3C-Empfehlung erhalten wird. Bereits jetzt qualifiziert sich XQuery als ein wertvolles und unverzichtbares Hilfsmittel für Anfragen auf semi-strukturierten Daten. Das Datenmodell von XQuery bildet eine solide Basis für die darauf aufbauenden Sprachkonzepte. Mit einer Fülle von beliebig zusammensetzbaren Ausdrücken und vordefinierten Funktionen ist die Sprache ausdrucksmächtig und nachweisbar relational vollständig. Durch die formale Definition der Semantik einer Teilmenge der Sprache, auf die alle Ausdrücke von XQuery abgebildet werden können, ist zudem ein sicheres mathematisches Fundament für die Sprache gegeben.

Ungeachtet der Tatsache, dass sich XQuery bereits seit dem Jahr 2000 in der Entwicklung befindet, fehlen der Sprache jedoch noch einige wichtige Bausteine. Als erstes ist hier die Aktualisierungsproblematik zu nennen. Das ursprüngliche Design von XQuery sieht keine Mechanismen zur Manipulation von XML-Dokumenten vor. Informationen können lediglich aus vorhandenen Dokumenten extrahiert und in ein Ergebnisdokument transformiert werden. Zwar existieren einige proprietäre Ansätze, XQuery mit Änderungsoperationen anzureichern, jedoch wurde von offizieller Seite her noch nicht mit einer Standardisierung einer

solchen Änderungsfunktionalität begonnen. Ein weiteres Problem ist, dass die Typableitungsregeln, wie sie von XQuery-Core verwendet werden, unvollständig sind. Es kann nachgewiesen werden, dass in bestimmten – wenngleich konstruierten – Fällen eine Typüberprüfung fehlschlägt, obwohl die zu überprüfende Anfrage korrekt ist. Wenngleich diese Problematik in der Praxis kaum eine Rolle spielt, ist die Situation dennoch unbefriedigend. Eine weitere Schwäche von XQuery ist die mangelnde Unterstützung der Sprache für Volltext-Primitiven. XML-Dokumente enthalten oft erhebliche Textanteile, so dass eine Volltextsuche für solche Dokumente unabdingbar ist. Jedoch ist ein erster Schritt zur Erweiterung von XQuery in diese Richtung bereits getan, indem das W3C Dokumente herausgegeben hat, die Anwendungsfälle und Anforderungen einer Volltext-Erweiterung von XQuery beschreiben. Mit *TeXQuery* existiert darüber hinaus bereits ein Vorschlag, der diese Anforderungen weitgehend erfüllt.

XQuery versucht mit einem durchdachten und sehr mächtigen Sprachkonzept den Anforderungen einer Anfragesprache gerecht zu werden. Dies ist einerseits sicherlich in einem hohen Maß gelungen, auf der anderen Seite fehlen der Sprache jedoch auch noch einige wichtige Bestandteile. Es dürfte jedoch lediglich eine Frage der Zeit sein, bis auch diese Lücken geschlossen werden und mit XQuery eine vollwertige Anfragesprache für semi-strukturierte Daten zur Verfügung steht.

Literatur

1. M. Brundage (2004): *XQuery – The XML Query Language* Addison-Wesley
2. W. Lehner, H. Schöning (2004): *XQuery – Grundlagen und fortgeschrittene Methoden* dpunkt.verlag
3. World Wide Web Consortium (2005): *XQuery 1.0: An XML Query Language* <http://www.w3.org/TR/xquery/>
4. World Wide Web Consortium (2005): *XQuery 1.0 and XPath 2.0 Formal Semantics* <http://www.w3.org/TR/2005/CR-xquery-semantics-20051103/>
5. D. Chamberlin (2002): *XQuery: An XML query language* www.research.ibm.com/journal/sj/414/chamberlin.pdf
6. Peter Fankhauser (2001): *XQuery Formal Semantics State and Challenges* www.sigmod.org/sigmod/record/issues/0109/SPECIAL/fankhauser2.pdf
7. D. Suciu (2002): *The XML Typechecking Problem* wam.inrialpes.fr/people/roisin/mw2004/Suciu2002sigmod.pdf