Standards II :
XQuery

Technische Universität München

Lehr- und Forschungseinheit III
Prof. R. Bayer, Prof. D. Kossmann

Hauptseminar Informatik im Sommersemester 2003

Web-Services

Bearbeiterin: Anja A. Rein

Betreuer: Prof. Kossmann

Vortragsdatum: 08.05.2003

Gliederung

0. Einführung

Diese Ausarbeitung behandelt die Anfragesprache XQuery, die eine Empfehlung des W3C ist. Die W3C wurde als Organisation wurde 1994 gegründet. Sie sieht ihr Ziel in der Entwicklung von Protokollen für das Internet, die den Nachrichtenfluss zwischen verschiedenen Anwendern vereinheitlichen soll. Dazu werden sogenannte Empfehlungen herausgegeben, die für Tools und kommerzielle Systeme einen Standard-Charakter haben. Ungefähr 400 Organisationen arbeiten in dem Konsortium mit. Es gingen aus dieser Organisation z.B. HTML, XHTML hervor.

1. Wiederholung von XML-Grundlagen

1.1 XML-Dokumente

Die Sprache XML ist eine so genannte „Markup-Sprache“. Das heißt neben den Daten sind zusätzliche, das Dokument strukturierende Informationen, die „Markups“ im Dokument enthalten. Markups haben die Form „<MarkupName>“, besitzen ein schliessendes Gegenstück der Form „</MarkupName>“ und bilden das charakteristische Kennzeichen von XML-Dokumenten. Die öffnenden und schliessenden Formen der Markups können geschachtelt werden und bilden so eine Klammerstruktur, die durch einen Baum dargestellt werden kann.

Zur Erinnerung hier zunächst ein einfaches XML-Dokument, das nachfolgend wiederholt als Beispiel aufgegriffen wird. Die Datei sei in Alle_Mitglieder.xml abgelegt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Alle_Mitglieder SYSTEM Alle_Mitglieder.dtd>

<Alle_Mitglieder>

<Mitglied Mitgliedsnummer="001">
        <Name>
                <Vorname>Anja</Vorname>
                <Nachname>Rein</Nachname>
        </Name>
        <Alter>23</Alter>
        <Adresse>
                <Strasse>Willi-Graf-Strasse</Strasse>
                <Hausnr>5</Hausnr>
                <PLZ>80805</PLZ>
                <Ort>Muenchen</Ort >
        </Adresse>
        <Telefon>
                <Vorwahl>089</Vorwahl>
                <Rufnummer>7896785</Rufnummer>
        </Telefon>
</Mitglied >
<Mitglied Mitgliedsnummer="002">
        .
        .
        .

</Alle_Mitglieder>
```

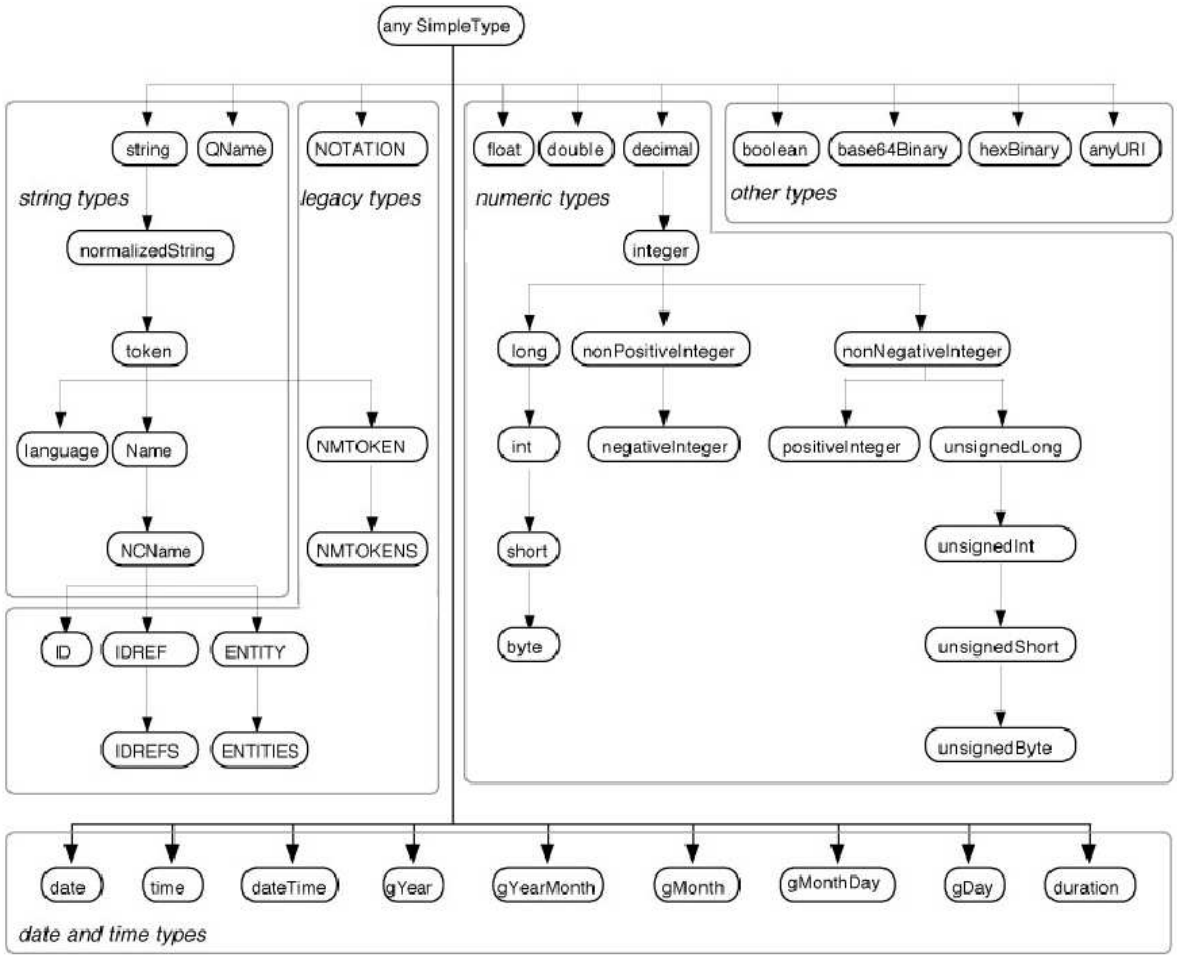
Wie am Beispiel zu erkennen ist, haben Daten folgende generelle Struktur:

```
<Elementname  [Attributname = wert]*>    Daten    </Elementname>
```

Dabei können die „Daten" weitere Markups enthalten.

1.2 XML-Schema

Mit dem XML-Schema können dem Markup-Bestandteilen **Typen** zugewiesen und eine **Struktur** für das gesamte XML-Dokument festgelegt werden. Das Schema ist besonders einfach zu lernen, da es in XML geschrieben wird. Es kann wohlgeformt(well-formed) oder gültig(valid) bei der Prüfung von einem XML-Dokument sein. Bei der Wohlgeformtheit werden zu der Überprüfung auf korrekte Reihenfolge und Struktur der Markups auch noch die Einhaltung der Wertebereiche und die Verwendung von korrekten Datentypen überprüft. Die Datentypen umfassen **einfache Typen** wie string, number, date. Diese können durch eine explizite Wertebereichsangabe eingeschränkt und dadurch vielseitiger verwendet werden. Die Typenvielfalt wird durch **komplexeTypen** ergänzt, die sich aus mehreren Elementen und ggf. Attributen zusammensetzten. Einmal definierte Typen können im Schema mehrmals verwendet werden. Ein XML-Dokument kann auch auf verschiedene Schema zugreifen. Selbst wenn die Schemata gleiche Namen verwenden werden diese durch die unterschiedlichen Namensräume getrennt.



einfache Typen im XML-Schema [XML_Seminar]

Durch explizites Gruppieren von Attributen (z.B. HTML: width + height) können im XML-Schema Attributgruppen definiert werden. Dadurch müssen immer

wiederverwendete Kombinationen nur einmal definiert, können aber gleichzeitig an mehreren Stellen verwendet werden.

Beispiel:

Dises Beispiel erstellt ein Schema für die einzelnen Mitglieder die oben in der Mitglieder Liste beschrieben werden. Hierbei treten komplexe Typen auf, die aus mehreren Elementen, Attributen und weiteren Typen zusammengesetzt werden. Das Mitglied hat z.B. ein Attribut „Mitgliedsnummer“ und besteht ansonsten aus einer Vereinigung(union) von komplexen und einfachen Typen.

```
<xds:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema>
   <xsd:complexType name="Mitglied">
   <xsd:attribute name="Mitgliedsnummer" type="xsd:string">
   <xsd:union>
      <xsd:complexType name="Name">
      <xsd:all>
         <xsd:element name="Vorname" type="xsd:string"/>
          <xsd:element name="Nachname" type="xsd:string"/>
      </xsd:all>
      </xsd:complexType name="Name">
      <xsd:simpleType>
          <xsd:restiktion base="xsd:integer">
          <xsd:minInclusive value="5">
          <xsd:maxInclusive value="100">
      < /xsd:simpleType>
      <xsd:complexType name="Adresse">
      <xsd:all>
         <xsd:element name="Strasse" type="xsd:string"/>
         <xsd:element name="Hausnr" type="xsd:string"/>
          <xsd:element name="PLZ" type="xsd:string"/>
         <xsd:element name="Ort" type="xsd:string"/>
      </xsd:all>
      </xsd:complexType name="Adresse">
      <xsd:complexType name="Telefon">
      <xsd:all>
         <xsd:element name="Vorwahl" type="xsd:string"/>
         <xsd:element name="Rufnummer " type="xsd:string"/>
      </xsd:all>
      </xsd:complexType name="Name">
   </xsd:union>
   </xsd:complexType name="Mitglied">
</xsd:schema>
```

Wie man aus dem Beispiel erkennen kann haben wir nur Mitglieder zwischen 5 und 100 Jahren.

2 Die Anfragesprache XQuery

2.1 Einführung in XQuery

XQuery ist ein Produkt der W3C in Form eines auf Papier formulierten Standards und dient der Abfrage von XML-Dokumenten. Die aktuelle Version ist XQuery 1.0. Unter XQuery versteht man damit eigentlich die Kombination von **XQuery** und **XPath.**
XPath ist ein selbstständiger Standard und wird für die Adressierung von Teildokumenten eingesetzt. Dafür stellt XPath Ausdrücke zur Selektion von Teilen eines XML-Dokumentes sowie Operatoren und Funktionen zur Verfügung, die mit extrahierenden Werten arbeiten. Diese Basis an Funktionalität wird außer in XQuery auch bei XSLT und XPointer verwendet.
XQuery ergänzt das Ganze zu einer vollständigen Anfragesprache. Zur Vereinfachung werden in dieser Ausarbeitung die beiden Bestandteile als ein Standard aufgefasst.
XQuery ist vom Leistungsumfang eine funktionale **Programmiersprache**. Die funktionale Programmierung ist ein Programmierstil, bei dem Berechnungen ausschließlich durch den Aufruf von Funktionen durchgeführt werden. Beispiele für diese Sprachen sind z.B. OCaml und Gofer. Durch dieses Charakteristika werden Seiteneffekte vermieden.
Eine Besonderheit stellt auch der strikte Umgang mit den verschiedenen **Typen** dar. Schon bevor ein Ausdruck ausgewertet wird, wird der zu erwartende Typ bestimmt und das Ergebnis auf die Zugehörigkeit zu diesem Typ überprüft. Entspricht er ihm nicht wird ein Cast versucht und ggf. eine Fehlermeldung ausgegeben. Bei Vergleichen gibt es z.B. strikte Regeln, wie sich ein Vergleichselement zum anderen typmässig verhalten muss. Auf diese Besonderheiten wird im Verlauf noch eingegangen.

2.2 Verwendung von XQuery

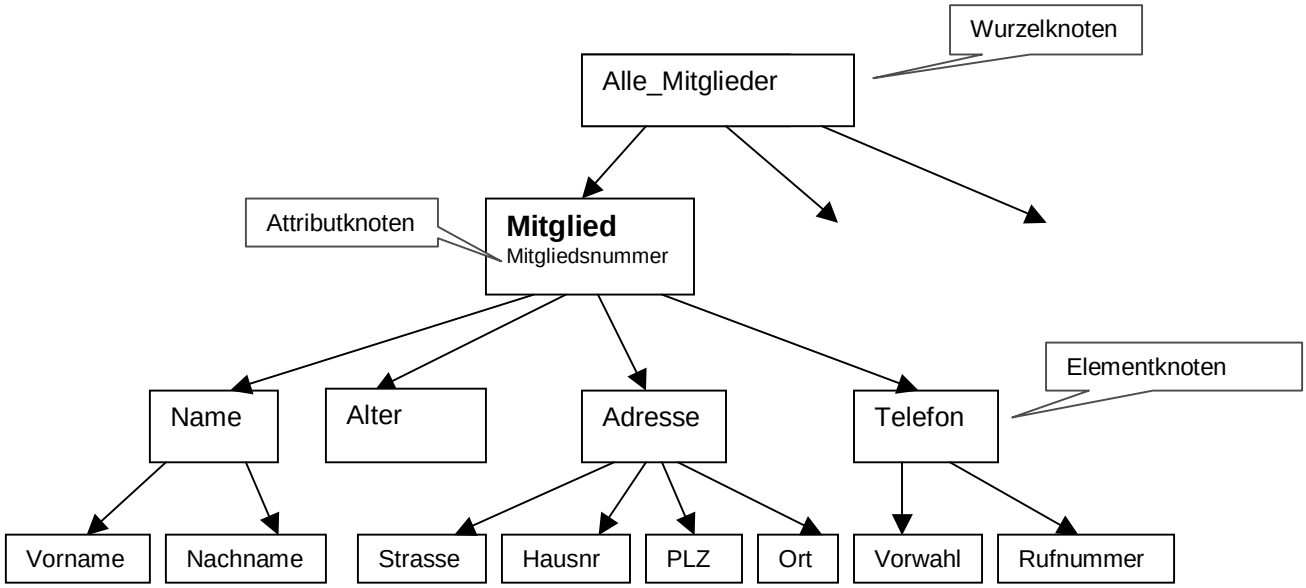
Die Anfragesprache XQuery hat das Erstellen oder Ableiten neuer XML-Dokumente als Anwendungsbereich. Dazu können aus einem oder mehreren Dokumenten unter vorher definierten Bedingungen Informationen extrahiert werden. Damit hat die Sprache alles an Funktionalität von der Abfragesprache SQL für Datenbanken. Trotzdem sind die beiden nicht vollständig gleich. Während XQuery sowohl strukturierte Daten (Inhalte aus relationalen oder objektorientierten Datenbanken) als auch semistrukturierte Daten in einem einheitlichen Format darstellen kann, beschränkt sich SQL auf Datenbanken. Diese Flexibiltität hat den Nachteil, dass sie XQuery komplexer und schwerer zu erlernen macht. Auch das Datenmodell auf dem beide basieren ist unterschiedlich. Während SQL in Tabellen abgepeichert wird, arbeitet XQuery auf dem Baummodell. Die gespeicherten **Daten** werden nach demselben Prinzip manipuliert. Das später erklärte FLWR(FOR/LET WHERE RETURN) ist stark an das SELECT-FROM-WHERE der Datenbank angelehnt. Es werden zunächst Werte ausgewählt, dann werden Bedingungen angegeben und schliesslich eine Ausgabe erzeugt. Abweichungen treten bei der Realisierung der

Sortierung auf, die bei XQuery nicht in die Datenmanipulation eingebunden, sondern mit einem extra Ausdruck(sortby) ermöglicht wird.
Von wirtschaftlicher Seite kann man noch die **Kosten** anführen. Datenbanken wie Oracle sind teuer und können somit nur bei grösseren Firmen, die neben dem erforderlichen Kapital auch die entsprechende Hardware besitzen, eingesetzt werden. Für XML-Dokumente hingegen benötigt man Dateien zum Abspeichern der Daten und einen normalen Browser zur Darstellung. Ausserdem kann man die Daten anschauen indem man eine Datei öffnet. Bei einer Datenbank müsste man sich extra durch Select-Ausdrücke Daten herauslesen.
Sind die Daten im XML-Format lassen sie sich durch XQuery sehr gut **Wieder-** und **Weiterverwenden**. Beispielsweise kann aus einem XML-Adress-Dokument mittels einer geeigneten Query die Liste aller Namen extrahiert werden. Damit kann ein Dokument zu einem anderen transformiert werden. Ausserdem kann literales XML zur Laufzeit durch Einfügen von XQuery neue Werte bekommen. Damit kann ein Dokument wie z.B. eine Rechnung für jeden Kunden zur Laufzeit erstellt werden. „Alte“ XML-Werte werden so in neue Dokumente integriert. Auch können erhaltene XML-Dokumente auf ihre Werte hin analysiert werden. Eine Buchhaltung kann z.B. anhand von den XML-Kontoauszügen automatisch die Richtigkeit ihrer Buchungen auf den Posten Bank überprüfen lassen. Ein weiterer Anwendungsbereich ist die Konstruktion von neuen XML-Dokumenten, wobei die XML-Fragmente in das XQuery eingebettet werden. Bei der Administration können z.B. Konfigurationsskripte oder Anwenderprofile benutzt werden, um Informationen zu erhalten. Zudem sollen Datenströme analysiert und von einem DOM(Data Object Model) duch Queries Knoten mit bestimmten Bedingungen erkannt werden. Auch soll man Schemata, Dokumenttypen, usw. nach nützlichen Vorgaben für das eigene zu erstellende Dokument durchsuchen können d.h. wenn es schon ein Mitglied-Schema gibt, kann man es finden und verwenden.
In allen Fällen bekommt man entweder benötigte Informationen oder ein neues XML-Dokument bei dem der Vorteil von Zusatzinformation durch Markups erhalten bleibt.
Da XQuery in Verbindung mit XML soviel Funktionalität zur Verfügung stellt, soll es an möglichst vielen Stellen einsetzbar sein. Zum Beispiel in einem normalen XML-Dokument, der URL oder der typischen Web-Sprache JSP. Eine weitere Möglichkeiten wäre als String in jeder gewöhnlichen Programmiersprache oder Mithilfe eines Protokolls.

2.3 Das zugrunde liegende Baum- Modell

Das XQuery zu grunde liegende Datenmodell basiert auf einem abstrakten Baum. Die Elemente des XML-Dokumentes sind die Knoten. Die Kanten realisieren die Beziehung zu den Subelementen.

Es lassen sich sieben verschiedene **Knotenarten** identifizieren:
- Wurzelknoten: oberster Knoten in einem Baum
- Elementknoten: Knoten, der ein Element darstellt
- Attributknoten: beinhalten die Attribute eines Elementes
- Namensraumangaben
- Textknoten: stellen den Inhalt eines Elementes dar
- Kommentarknoten
- Verarbeitungshinweise (Processing Instructions)



Alle Vereinsmitglieder besitzen den Wurzelknoten „Alle_Mitglieder“, da jedes XML-Dokument eine Wurzel benötigt um gültig zu sein. An dieser Wurzel hängen die Mitglieder mit ihrem Attribute „Beitrittsjahr“. Die Knoten „Name“, „Adresse“ und „Telefon“ strukturieren den Baum. Sie besitzen die Elemente „Vorname“, „Nachname“, usw..
Dieser Baum dient auch als Basis für spätere Beispiele.
Manche Operationen und Ausdrücke beziehen sich auf die **Dokumentordnung**. Sie ist durch die Abfolge der Elemente, gefolgt von der Namensraumangabe vor den Attributen, gegeben.
Ein weiterer wichtiger Begriff ist die **Sequenz**. Sie ist eine Instanz oder ein Objekt entsprechend dem Datenmodell und kann aus keinem oder beliebig vielen Elementen bestehen. Eine Sequenz enthält keine Untersequenzen, ist geordnet (Dokumentordnung) und kann auch Duplikate enthalten.
Als Grunddatentypen stehen bei XQuery boolean, string, float, decimal, sowie einige weitere zur Verfügung. Diese wurden aus dem XML Schema übernommen. Der Nutzer kann allerdings auch eigene Typen definieren.

2.3 Einfache Ausdrücke

2.3.1 Konstanten, Variablen und Kommentare

Konstanten können für elementare Werte wie String, Integer, Double oder Float angegeben werden. Strings bestehen aus einer Zeichenkette, die in einfache oder doppelte Hochkommata eingeschlossen wird.

Beispiele: “Fahrrad“, ’Fahrrad’, “2000.20“ {--Typ: String--}
134, -123, 0 {--Typ: Integer--}
56.01, -12,7 {--Typ: Float--}
12e5, -234.09e7 {--Typ: Double, e ist Exponent--}

Variablen sind durch ein vorangestelltes $-Zeichen gekennzeichnet. Man kann ihnen keine Werte zuweisen, sondern sie nur an das Ergebnis von Ausdrücken binden, das dadurch auch den Typ bestimmt. Somit bleiben sie von eventuellen Seiteneffekten verschont. Ihre Gültigkeit beschränkt sich auf den aktuellen und aller eingeschlossener Anfrageausdrücke. Wird die Variable außerhalb aufgerufen, ist sie ungebunden und erzeugt eine Ausnahme.

Beispiel: LET $neustesMitglied := Ausdruck
RETURN $neustesMitglied

Kommentare können zur besseren Lesbarkeit des Codes eingefügt werden. Sie werden von einer geschweiften Klammer und zwei Gedankenstrichen eingerahmt.

Beispiel: {--Das hier ist ein Kommentar--}

2.3.2 Arithmetik, Vergleiche und logische Ausdrücke

Bei den **arithmetischen Ausdrücken** werden die Grundrechenarten Addition „+“, Subtraktion „-“, Multiplikation „*“, Division „DIV“ sowie die Modulo-Rechnung „MOD“ verwendet. Die Klammernregeln der Algebra werden auch hier verwendet. Ausserdem stehen analog zu Programmiersprachen wie Java abkürzende Ausdrücke zur Verfügung:

Im Beispiel wird das aktuelle Alter um ein Jahr hochgesetzt:

//alter+1

In XQuery gibt es 4 verschiedene Typen von **Vergleichsausdrücken**, die den Vergleich zwischen unterschiedlichen Operanten erlauben:

-allgemeine Vergleichsausdrücke mit Sequenzen: „=“, „!=“, „<“, „<=“, „>“, „>=“

- Vergleich elementarer Werte: „EQ“(equal), „NE“(not equal), LT(lower then), „LE“(lower equal), „LT“(lower then), „GT“(greater then), „GE“(greater equal)

-Knotengleichheit überprüfen: “==”, “!==”

- Vergleich der Knotenabfolge bzgl. der Dokumentordnung: „<<“, „>>“

Beispiele:

In diesem Beispiel soll herausgefunden werden, ob das Mitglied mit dem Namen „Anja Rein“ die Mitgliedsnummer 001 hat. Dabei werden die später noch diskutierten Operatoren [...] zur Selektion von Mitgliedern und @variable fuer den Attributzugriff verwendet.

$mitglied/ name [vorname=“Anja“]==$mitglied[@id=“001“]

Interessant wird es, wenn zwei nicht zusammenpassende Typen aufeinander treffen. Wie oben schon erwähnt gibt es eine strenge Typauswertung. Für einfache Werte gilt folgendes: Ist einer der beiden Vergleichselemente boolean oder nummerisch wird der zweite versucht anzupassen. Andernfalls werden beide als string verglichen.

$mitglied/ name [vorname=“Anja Rein“]==$mitglied[@id=“001“] == count($mitglied)

Der erste Ausdruck ergibt „true“ und der zweite einen nummerischen Ausdruck. Dieser ist nicht null und wird somit ebenfalls in „true“ umgewandelt. Somit ist der Ausdruck insgesamt ebenfalls wahr.

Ein weiteres Beispiel, das das Zusammenspiel ziwschen der nachfolg diskutierten Logik und Sequenzen zeigt, stellen die beiden nachfolgenden Ausdrücke dar:

$mitglied/name/vorname != „Donald “

NOT($mitglied/name/vorname = „Donald “)

Auf den ersten Blick sehen diese beiden Ausdrücke ähnlich aus. Sie haben aber eine ganz unterschiedliche Bedeutung. Im ersten Fall ist der Ausdruck wahr, wenn es ausser „Donald“ noch andere Mitglieder des Vereins gibt. Im zweiten darf er kein Mitglied sein, um „true“ zu erhalten.

Logische Ausdrücke
Als logische Ausdrücke stehen Und- sowie Oder-Verknüpfungen mit AND bzw. OR bereit. Die Negation wird durch „NOT“ ausgedrückt.

Beispiel:

Hier soll herausgefunden werden, ob das Mitglied, das den Namen „Anja Rein“ besitzt nicht das gleiche ist, wie das Mitglied mit der Nummer 002.

NOT($mitglied/name [vorname=“Anja“] == $mitglied[@id=“002“])

2.3.3 Navigation mit Pfadausdrücken

Pfadausdrücke dienen zur Selektion und Navigation im Baum. Sie sind relativ stark an die Unix-Konvention zur Definition von Pfadausdruecken angelehnt und können aus mehreren Teilausdrücken (Schritten,Steps) bestehen. Jeder Schritt liefert entweder eine Knotenmenge oder einen einzelnen Wert, da man durch sich einen Baum bewegt. Einzelne Schritte werden durch einen Schrägstrich getrennt.
Eine Pfadangabe ist entweder ein relativer Pfad, der im Bezug zu einem Kontextknoten ausgewertet wird, oder ein absoluter Pfad, der von der Wurzel aus geht. Ein absoluter Pfad ist wie in Unix-Pfadangaben durch einen vorangestellten Schrägstrich gekennzeichnet.

Beispiel:

Hier werden die Vornamen aller Mitglieder gefunden.

/mitglied/name/vorname

Eine (nicht vollständige) Liste der häufig verwendeten Pfadausdrücke:

- „.“ bezeichnet den Kontextknoten. Der Kontextknoten eines Knoten ist der im aktuellen Zusammenhang verwendete Knoten.
- „..“ bezeichnet den Vaterknoten
- „/“ bezeichnet den Wurzelknoten oder trennt Pfadausdrücke
- „//“ bezeichnet alle direkten Nachkommen des Kontextknotens
- „@attribut“ bezeichnet ein Attribut des Kontextknotens
- „@*“ bezeichnet alle Attribute des Kontextknotens
- „[n]“ n ist das n-te Element aus einer Sequenz von Knoten
- „[expr1 TO expr2]“ ist eine Teilsequenz, die durch den Bereich vom ersten bis zum zweiten Ausdruck beschrieben wird

Beispiel:

In diesem Beispiel wird die Bibliotheksfunktion last() verwendet. Sie gibt die Position des letzten Elementes als Integer wieder. Achtung: Hier wird von eins aus gezählt!

```
/mitglied[3]/name/nachname              {--Name des dritten Mitglieds --}
 /mitglied[last()]/name/nachname        {--Name des letzten Mitglieds --}
 /mitglied[1 TO 100]/name/nachname      {--Name der ersten hundert Mitglieder --}
```

2.4 Kontrollbeschreibung durch FLWR und bedingte Ausdrücke

FLWR-Ausdrücke bilden eine weitere Basis der XQuery-Anfragen. Das Kürzel steht für **F**OR, **L**ET, **W**HERE und **R**ETURN und erlaubt eine der konventionellen prozeduralen Programmierung ähnliche Programmstruktur.
Allgemein können die Ausdrücke wie folgt kombiniert werden:

(FOR-Ausdruck | LET-Ausdruck)+
WHERE-Ausdruck? RETURN-Ausdruck

FOR und LET binden mindestens eine Variable an die Auswertungsergebnisse. Die FOR-Klausel bindet jedes einzelne Element in einer Schleife an die Variable und führt dann den Schleifenrumpf aus. Die Variable hat daher den Typ einzelner Elemente der Sequenz. Im Gegensatz dazu wird bei LET der Variable genau ein Wert zugewiesen. Ist der berechnete Ausdruck selbst eine Sequenz, so wird diese Seqeunz als Ganzes an die Variable gebunden.

Mit der WHERE-Klausel kann die Ergebnismenge eingeschränkt werden. Alle gefundenen Variabeln können so auf bestimmte Eigenschaften überprüft werden.

Die Ergebnismenge wird schliesslich mit RETURN zurückgeben.

Die einzelnen Ausdrücke setzen sich wie folgt zusammen:

FOR-Ausdruck ::=
FOR $Variablenname IN Ausdruck (, $Variablenname IN Ausdruck)*

LET-Ausdruck ::=
LET $Variabelnname IN Ausdruck (, $Variablenname IN Ausdruck)*

WHERE-Ausdruck ::=
WHERE Ausdruck

RETURN-Ausdruck ::=
RETURN Ausdruck

Aufgrund des syntaktischen Aufbaus der Sprache XQuery, kann an jeder Stelle, an der „Ausdruck“ steht , ein beliebiger XQuery-Ausdruck eingesetzt werden und so können verschachtelte Ausdrücke gebildet werden. Ausserdem ist die Syntax von

XQuery orthogonal zu XML und kann deshalb an Stellen eingesetzt werden, an denen ein XML-Ausdruck gefordert wird.

Beispiel:

Die Nachnamen aller Mitglieder sollen herausgesucht und zurückgegeben werden. Das Ergebnis liefert eine Sequenz.

```
LET $mitgliedernamen := //name/nachname
RETURN $mitgliedername
```

Bei einem **bedingten Ausdruck** ist ein bestimmtes Ergebnis von einem Ausdruck abhängig. Ergibt dieser den Wahrheitswert true wird die THEN-Bedingung ausgewertet, andernfalls wirddie optional vorhandene ELSE-Bedingung ausgewertet. Das Grundgerüst der Ausdrücke sieht wie folgt aus:

IF (expr) THEN expr_1 ELSE expr_2

Beispiel:

Alle Mitglieder über 65 Jahre bekommen im Verein den Status „Senior“, alle über 21 sind Erwachsene und der Rest zählt zu den Jugendlichen.

```
LET $alter := /mitglied/alter
IF ($alter < 21)
        THEN $status := „Jugendlicher“
ELSE  $status := „Erwachsener“
```

2.5 Verbunde

Um Informationen, die sich aus verschiedenen Elementen und Attributen ergeben zu kombinieren, verwendet man den Verbund.
Der „natürliche Verbund“ führt die Daten über gleiche Werte für gemeinsame Elemente zusammen. Sind diese nicht vorhanden, entsteht ein (möglicherweise sehr großes) karthesisches Produkt über den Variablenbindungen.
Die Verbundbedingung wird direkt nach den Variablenbindungen der Daten angegeben. Falls die benötigten Elemente hier noch nicht zur Verfügung stehen, schreibt man sie in die WHERE-Klausel.

Beispiel:

Wir nehmen an, es bestehe zusätzlich zu dem XML-Dokument „Alle_Mitglieder“ ein Dokument mit dem Namen „Vorstandsmitglieder.xml“ in dem alle Vorstandsmitglieder namentlich genannt sind. Die Struktur ist gleich der ersten, nur hat ein Vorstandsmitglied als einziges Subelement einen Namen. Jetzt soll aber eine Liste der Vorstandsmitglieder mit sämtlichen Daten erzeugt werden.

```
<vorstandsmitgliederliste>
FOR $mitglied IN /
      $mname IN dokument(Vorstandsmitglieder.xml)//name/nachname
WHERE   $mitglied/name/nachname =  $mname
RETURN
       <vorstandsmitglied>
              <name>{$mitglied/name/nachname , $mitglied/name/vorname}</name>
              <adresse>{$mitglied/adresse/strasse, ...}</adresse>
               ...
       </vorstandsmitglied>
</vorstandsmitgliederliste>
```

2.7 Funktionsaufrufe und Sichten

Zu den Standardfunktionalitäten gehören auch die **Funktionsaufrufe**, die es ermöglichen, auf die Funktionsbibliothek zuzugreifen. Hier werden vor allem Typcastings und einfache Funktionen auf Sequenzen, wie z.B. first() oder die bereits in einem Beispiel benutzten last() und count(), bereitgestellt. Aufgrund der Wichtigkeit des Typcastings für XQuery haben die Casts eine zentrale Bedeutung in der Sprache. Die Funktionssammlung kann vom Benutzer durch **selbstdefinierte Funktionen** erweitert werden.

Um bereits vorhandene Daten in unterschiedlichen Kontexten zu nutzen, kann man wie bei einer Datenbanken **Sichten** erzeugen. Diese werden über eine Funktionsdefinition realisiert, die sogar rekursive Sichten unterstützt.
Ihr Aufbau:

define function viewname (parameter) **returns** [type] expression

Beispiel:

In diesem Beispiel werden aus der Sicht „alter()“ alle Altersangaben, die grösser als 65 sind, zurückgegeben. Aus dieser Sequenz kann man dann z.B. das Durchschnittsalter der Senioren ausrechen oder die Anzahl der Senioren im Verein zählen.

```
define function alter() returns xs:integer{
        dokument(„Alle_Mitglieder.xml“}/alter
}
```

Aufruf:

```
FOR $a  IN alter() WHERE $a>65
        RETURN $a
```

2.6 Weiterführende Funktionalitäten

Ausser den bereits oben vorgestellten Funktionalitäten, gibt es noch eine Reihe weiterer Funktionalitäten, die der XQuery-Standard bietet.
Sequenzen können zum Beispiel **sortiert** oder nach verschiedenen Kriterien **gruppiert** werden.

Auch der mathematisch **All-** bzw. **Existenz-Quantor** ist in dem Standard integriert werden aber über endliche Sequenzen interpretiert und sind damit berechenbar.
Sollen Namensraueme, Schemaimporte oder Funktionsdefinitionen erfolgen, stehen diese vor der eigentlichen Anfrage im **Anfrageprolog**.

Die gezeigten und alle weiteren **Elementkonstruktoren** erzeugen ihrerseits XML-Fragmente, die entweder als XML-Rückgabewert einer Anfrage fungieren oder den Rahmen eines XML-Dokuments flexibel ergänzen;
Wie man sieht, stellt XQuery verschiedene Möglichkeiten zur Verfuegung aus einem XML-Dokument Daten herauszusuchen, sie zu verändern und zu neuen zusammenzufügen. Damit bietet die Anfragesprache dieselben funktionalen Möglichkeiten wie eine Datenbank.

3 Zusammenfassung und Bewertung

Mit XQuery steht dem Entwickler eine Sprache zur Verfügung, die es ihm erlaubt, effizient Anfragen auf ein XML-Dokument zu machen und die Anfrageergebnisse in Form neuer XML-Dokumente zu erhalten.
Dieser Standard ist für viele praktische Anwendungen sinnvoll, da infolge der Globalisierung immer mehr moderne Firmen im XML-Format plattformunahängig Daten austauschen. Zur Erstellung dieser Informationen sind Aufbereitungen der intern verwendeten Daten notwendig. XQuery stellt auch dafür die notwendigen Funktionalitäten bereit. Eines von vielen weiteren Anwendungsfeldern ist die Aufbereitung von XML-Dokumenten zur Anzeige im Web, die ähnlichen Grundmechanismen unterliegt.

Auf Basis der gesammelten Informationen halte ich es daher für wahrscheinlich, dass XQuery in Zukunft als Basis für Anfragen auf XML-Dokumente dient. Dafür spricht vor allem der Einfluss der W3C-Organisation auf diesen Bereich und die durchdachten und von bewährten Systemen übernommenen zugrundeliegenden Konzepte. Besonders erwähnenswert ist hierbei die Orthogonalität und die daraus resultierende Einbettbarkeit in XML sowie die Berechnungsmächtigkeit durch die angebotenen Kontrollstrukturen.

Vom Funktionsumfang entspricht die Sprache dem heute zum beherrschenden Standard gewordenen SQL für XML-Strukturen. Da XML-basierter Datenaustausch und die damit verbundene Datenverarbeitung immer wichtiger wird, ist zu erwarten, dass auch XQuery wesentliche Verbreitung finden wird. Ein erster Beleg dazu sind die zahlreiche Prototypen zu diesem Thema, die auf der Seite „http://www.w3.org/XML/Query“ nachzulesen sind.

Allerdings sind, wie auch die in dieser Ausarbeitung formulierten Beispiele zeigen, XQuery-Anfragen nicht besonders leicht zu lesen und neigen, wie ganz XML zu Overhead aufgrund der verwendeten Markups. Diese umschliessen die Daten und erschweren damit das Erkennen der eigentlichen Information.

4 Literatur

[XML_Seminar] Heiko Mining, „XML_Schema“
http://www.heikominning.de/seminar1/FolienOhneAnimation.pdf

[XML_Schema] W3C
http://www.w3.org/XML/Schema

[XML] W3C
http://www.w3.org/XML/

[XQuery_Requirements] W3C
http://www.w3.org/TR/xmlquery-full-text-requirements/

[XQuery_UseCases] W3C
http://www.w3.org/TR/xmlquery-use-cases/

[XQuery] W3C
http://www.w3.org/XML/Query

[XML_Vorlesung] Prof. Kossmann, WS02/03
http://www3.in.tum.de/lehre/WS2002/XML-kossmann/