Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. Theo Härder

Integriertes Seminar
Thema: Data Streams
Sommersemester 2005

Verarbeitung von XML-Strömen

Katharina Bellon

k_bellon@informatik.uni-kl.de

Betreuer: Dipl.-Inf. Christian Mathis

1. Einleitung

Mit einem Datenstrom wird eine kontinuierliche Abfolge von Datensätzen bezeichnet. Im Gegensatz zu Datenbanken und anderen Datenquellen, können nicht alle Daten aus einem Strom zwischengespeichert, und dann später bearbeitet werden. Eine Zwischenspeicherung würde die Kapazität der meisten Datenverwaltungssysteme (*Data Stream Management System*, kurz DSMS) übersteigen, außerdem werden die Ergebnisse der Bearbeitung in vielen Fällen sofort gebraucht. Also sind die Daten fortlaufend zu bearbeiten.

Es gibt drei Arten von Datenströmen. Das sind die Tupel-, Punkt- und XML-Datenströme. In dieser Ausarbeitung wird nur die Verarbeitung der XML-Datenströme betrachtet. Diese Datenströme werden gegenüber den anderen beiden Typen immer öfters bevorzugt. Die Vorteile der XML-Daten liegen in ihrer Semistrukturierung, einer standardisierten Syntax und einer frei wählbaren Semantik. Die Struktur von XML-Daten erlaubt eine flexible Darstellung. Damit können also solche Daten, die eine unregelmäßige Struktur besitzen, in XML leicht beschrieben werden. XML ist eine plattformunabhängige Sprache, das heißt sie ist weder an eine bestimmte Hardware, noch an ein Betriebssystem oder eine Programmiersprache gebunden. Außerdem gibt es schon eine Vielzahl implementierter und bereitgestellter Anwendungen, die für die weitere Bearbeitung der XML-Daten benutzt werden können. All diese Vorteile werden zum Beispiel von Überwachungs- und Steuerungssystemen für eine schnelle und speichereffiziente Verarbeitung verwendet. Insbesondere bei den Messdatenströmen haben die zugehörigen XML-Dokumente einen sehr geringen Textanteil und können eine möglicherweise unbegrenzte Schachtelungstiefe und Größe besitzen. Diese Merkmale stellen für mehrere Anwendungen eine besondere Herausforderung dar.

Es gibt eine Reihe von Anforderungen, die an DSMS gestellt werden. Die Echtzeitanforderung ist eine der wichtigsten. Das heißt, dass in diesem Fall von einem System eine möglichst kurze Antwortzeit erwartet wird. Da die Verarbeitung von XML-Daten eventuell große Speicherplatzbereiche in Anspruch nehmen kann, ist der Bedarf an schnell arbeitenden XML-Datenbearbeitungsverfahren, die auch platzsparend sind, groß.

In dieser Ausarbeitung werden die meist verbreiteten XML-Anfragesprachen, XPath und XQuery, dargestellt und besprochen. Sehr viele Anwendungen basieren auf diesen Sprachen und verwenden sie. Zum Beispiel bildet XPath eine Basis für XQuery, welches vom W3C (*World Wide Web Consortium*) entwickelt wurde. Im Zusammenhang mit XQuery wird hier die Arbeitsweise des *XQuery-Streaming-Prozessors,* der als Hauptkomponente für das BEA-Projekt konstruiert wurde, beschrieben. Das Ziel bei der Entwicklung dieses Prozessors war eine schnelle Bearbeitung der XML-Nachrichten. Die hier beschriebene Vorgehensweise, mit der die Anfragen ausgewertet werden, ist bei fast allen Query-

Prozessoren vorzufinden und ist als Beispiel zu verstehen. Zum Abschluss des dritten Kapitels werden einige Möglichkeiten der Optimierung von Anfragen dargestellt.
Im vierten Kapitel werden die Systeme SPEX und XMLTK, sowie X- und Y-Filter-Systeme behandelt. Die Arbeitsweise des SPEX-Anfrage-Prozessors ist in vier Schritten aufgebaut. Der wichtigste Schritt ist der Aufbau eines FSXP-Anfrage-Plans. Anhand diesem, aus Transduktoren bestehenden Netzwerks, wird die XPath-Anfrage abgearbeitet. Das System XMLTK ist im Grunde genommen ein Werkzeugsystem, das aus vielen Dienstprogrammen und Anwendungen besteht, die bei der Bearbeitung von XML-Datenströmen nützlich sind. Die beiden wichtigsten Systemkomponenten im XMLTK sind, der Strom-Index und der XPath-Prozessor, die in Abschnitten 4.2.1 und 4.2.2 aufgeführt und erklärt werden. Ein XML-Filter-Systeme bietet eine schnelle Überprüfung der XML-Daten gegen eine große Anzahl von Anfragen. Die Dokumente eines XML-Stromes werden durch einen Filter, die so genannte *Filtering Engine*, durchgelassen und mit einer Menge von Anfragen abgeglichen. Die Daten, die diesen Anfragen genügen, werden ausgewählt und an eine möglicherweise große Anzahl von Benutzern, die an denen interessiert sind, weiterverschickt.
Alle hier beschriebene Anwendungen und Werkzeuge sind für die Verarbeitung der XML-Datenströme von großer Bedeutung. Viele von diesen Systemen sind immer noch im Stadium der Entwicklung. Je mehr das XML und die XML-Daten an Bedeutung gewinnen, desto mehr Anwendungen werden für ihre Bearbeitung ausgearbeitet und implementiert.

2. XML-Datenströme und ihre Anwendungen und Anforderungen

Viele Anwendungen verwenden für den Datenaustausch, statt Punkt- oder Tupelströme, immer mehr XML-Datenströme. Für manche Applikationen sind sie besser geeignet als flache Daten. Der Vorteil liegt darin, dass diese Daten semistrukturiert sind. Dies bedeutet das diese Daten eine unregelmäßige Struktur besitzen und damit in ihrer Darstellung sehr flexibel sind. Außerdem besitzt XML eine standardisierte Syntax und eine frei wählbare Semantik, was für viele Anwendungen geradezu optimal ist. Zum Beispiel sind diese Anwendungen in solchen Gebieten wie der Nachrichtenüberwachung, genauer gesagt bei den Börsen-, Presse- und Meteorologienachrichten zu finden. In der Systemüberwachung und Systemsteuerung bei Verkehrs- und Produktionssteuerung, Logistik, Netzwerkverwaltung und bei der Analyse von wissenschaftlichen Messdaten in der Medizin oder der Astronomie, wie auch bei der Früherkennung von Tornados, finden die XML-Datenstöme eine verbreitete Anwendung.

Allerdings haben auch XML-Dokumente einige Nachteile. Diese Dokumente haben gegebenenfalls eine unbegrenzte Länge und eine große Schachtelungstiefe, was zu einer mangelnder Skalierbarkeit führen kann. Zum Beispiel wird ein XML-Dokument bei dem XML-Verarbeitungsmodells DOM (*Document Object Model*) von W3C, komplett als Baum im Hauptspeicher dargestellt. Dies kann, je nach DOM-Implementierung, zu einem hohen Speicherverbrauch und einer niedrigeren Verarbeitungsgeschwindigkeit führen. Zum Beispiel belegt ein 20 MB großes XML-Dokument im Hauptspeicher als DOM repräsentiert zwischen 200 MB und 400 MB [TeFW].

Aus Anwendersicht existieren zwei Typen von Datenströmen. Es wird oft zwischen Mess- und Transaktionsdatenströmen unterschieden. Dabei ist zu bemerken, dass der Transaktionsbegriff hier nicht im traditionellem Sinne, wie er in der Datenbankterminologie verwendet wird, zu verstehen ist, sondern im Sinne einer Benutzertransaktion. Damit sind die Transaktionen wie Telefonanrufe, Nutzung der Kreditkarte oder Zugriff auf Webressourcen gemeint. Die von diesen Benutzertransaktionen ausgesandte kontinuierlichen Aufzeichnungen, auch Log-Daten genannt, zählen zu den Transaktionsdatenströmen. Messdaten werden von wissenschaftlichen, zum Beispiel medizinischen, meteorologischen oder astronomischen Messstationen geliefert. Auch Datenströme, die von Sensoren oder Rechnernetzwerke geliefert werden, lassen sich zu den Messdatenströmen zählen. Also liegt der Hauptunterschied darin, dass Transaktionsdatenströme Informationen über den Transaktionsablauf enthalten, während und die Messdatenströme Daten mit bestimmten Messwerten umfassen.

Einen Datenstrom zu analysieren bedeutet zum Beispiel, zusammenhängende Werte aufzufinden und zu untersuchen. Bei der Analyse solcher Datenströme braucht man platzsparende Verfahren, die am besten keine Zeitverzögerung aufweisen.

Als eine weitere Anforderung an die XML-DSMS wäre die Entwicklung von genaueren Approximationsmechanismen zu nennen. Diese Mechanismen werden für die Annäherung der Anfrageergebnisse gebraucht. Bei einem Datenstrom ist es unmöglich zu erkennen, welche Daten für eine Bearbeitung zu einem späteren Zeitpunkt noch benötigt werden. In dem Fall, in dem man aber auf die Anfragen, die auf früheren Daten verweisen, nicht verzichten kann oder will, braucht man die oben genannten Approximationsmechanismen. Das heißt, die Auswertungen solcher Anfragen können nicht ein exaktes Ergebnis, sondern nur ein angenähertes Resultat liefern.

Bei den Systemen die XML-Datenströme verarbeiten, wird auch die Erfüllung der Echtzeitanforderungen erwartet. Das heißt, es wird erwartet, das die Antwort eines Systems in einem bestimmten Zeitpunkt geliefert wird. Zum Beispiel bei der Überwachung eines Intensivpatienten, muss das System das Eintreten eines kritischen Zustandes sofort melden. Da die Datenströme oft ein unregelmäßiges Eintreffen der Daten aufweisen, kann es manchmal zur Datenüberflutung kommen. Dies führt in meisten Fällen zur Überlastung des Systems. Um die Echtzeitanforderung trotz der Überlastung erfüllen zu können, muss auf einzelne Datensätze verzichtet werden. Dies kann auch manchmal zu einer Ungenauigkeit der

Anfrageergebnisse führen. Es gibt Mechanismen, die bei der Verwerfung der Datensätze in solchen Situationen eingesetzt werden. Solches Verfahren wird *Load Shedding* genannt. Die Verwendung von Load Shedding ist für die maximale Senkung der Ungenauigkeit der Resultate von großer Bedeutung [Jörg05].

3. Verarbeitung von XML-Strömen und Anfragesprachen

Für die Anfragen gegen XML-Datenströme bieten sich die Anfragesprachen XPath oder XQuery und ein Paar andere weniger verbreitete Anfragesprachen an. Zu diesen Sprachen gehören *XSAGs* (*XML Stream Attribute Grammars*), *XML-QL*, *Lorel*, die speziell für semistrukturierte Daten entwickelt wurde, die *XML Query Language*, kurz *XQL*, *Quilt* und viele andere [Koeh01]. Für besondere Gebiete werden auch spezielle Anfragesprachen definiert. Zum Beispiel stellt die Sprache *XML-GL* eine geeignete Anfragesprache für benutzerfreundliche Schnittstellen dar.

3.1 XPath

Viele Systeme und Transformations- und Anfragesprachen, wie zum Beispiel SPEX, XMLTK, XQuery, usw. benutzen oder basieren auf XPath. Diese Sprache ist ein wichtiger Kern der Sprache XQuery und daher für die Anfrage gegen XML-Ströme von großer Bedeutung. In diesem Abschnitt werden die grundsächlichen Elemente von XPath angesprochen, um später die Arbeitweise der dargestellten Systeme besser zu verstehen. Außerdem ist hier die Sprache SXP kurz beschrieben, da sie einen Kern von FSXP bildet. Diese wiederum wird als Anfragesprache vor allem bei dem System SPEX verwendet.

3.1.1 XPath 1.0

Wie schon oben erwähnt, ist XPath eine Anfragesprache, die von vielen XML-Datenstromsystemen benutzt wird und daher von großer Bedeutung ist. Mit XPath kann man Teile eines XML-Dokumentes adressieren. Ein XML-Dokument wird als Baum repräsentiert und die Lokalisierungspfade werden zur Navigation durch diesen Baum benutzt. Die Knoten eines solchen Baumes sind die XML-Elemente. Der

Ausgangspunkt des Lokalisierungspfades ist der Kontextknoten. Falls ein Knoten nach einem Lokalisierungspfad erreicht wird, dann wird er als Kontextknoten bezeichnet.

Ein Lokalisierungspfad ist eine Folge von Lokalisierungsschritten, die durch „/“ abgetrennt werden. Ein Achsenname ist ein Teil eines Lokalisierungsschrittes. Dieser Name gibt die Richtung an, in welche der Pfad navigiert werden soll. Außerdem besteht jeder Lokalisierungsschritt noch aus einer Knotenabfrage (*node test*) und einem Prädikat.

Eine Knotenabfrage besteht wiederum aus einem Namen- oder einem Typvergleich. Mit Hilfe eines Knotentests beschränkt man die Elementauswahl einer Achse. Die Schreibweise für so eine Knotenabfrage sieht folgendermaßen aus: *Achse::Knotentest*, zum Beispiel `//child::book`.

Das Prädikat ist auch ein XPath-Ausdruck, welcher bei der Auswertung einen booleschen Wert als Ergebnis liefert. Damit wird das Ergebnis weiter eingeschränkt. Sobald das Prädikat zu „`true`“ ausgewertet wird, selektiert man den Knoten, in dem die entsprechende Knotenmenge vorhanden ist. Der Ausdruck in dem folgenden Beispiel liefert alle Knoten vom Typ „Buch“, die Kinderelemente vom Typ „Seite“ besitzen und deren Anzahl zwischen 10 und 100 liegt:

```
//Buch[count(Seite)<=100 and count(Seite)>=10].
```

In der folgenden Tabelle werden die Achsennamen und die dazugehörigen Beschreibungen und Abkürzungen dargestellt.

Achse	adressierte Knoten	Abkürzung
`child`	direkt untergeordnete Knoten	*weglassen*
`parent`	direkt übergeordnete	`..`
`self`	der Referenzknoten selbst (nützlich für zusätzliche Bedingungen)	`.`
`ancestor`	übergeordnete	
`ancestor-or-self`	übergeordnete und der Knoten selbst	
`descendant`	untergeordnete	
`descendant-or-self`	untergeordnete und der Knoten selbst	`//`
`following`	alle Konten nach dem Kontextknoten, außer seinen Nachfahren	
`following-sibling`	alle Geschwisterkonten, die sich nach dem Kontextknoten (rechts) im Baum befinden. Achse ist leer wenn Kontextknoten Attribut- oder Namenspace-Knote ist.	

`preceding`	alle Knoten vor dem Kontextknoten	
`preceding-sibling`	wie `following-sibling` nur das die betrachteten Knoten von dem Kontextknoten links im Baum stehen	

Ein weiteres Beispiel zeigt mögliche XPath-Ausdrücke und deren Auswertungsergebnisse, die auf ein gegebenes XML-Dokument angewendet werden.

Beispiel 1 für ein XML-Dokument:

```
<?xml version="1.0">
<dok>
      <!-- ein XML-Dokument -->
      <kap title="Kapitel Eins">
            <pa>Erster Satz</pa>
            <pa>Zweiter Satz</pa>
      </kap>
      <kap title="Kapitel Zwei">
            <pa>Erster Satz</pa>
      </kap>
</dok>
```

Beispiele für XPath-Ausdrücke:

`/*`	selektiert den Wurzel-Element
`//kap[@title="Kapitel Eins"]/pa`	selektiert alle Absätze des Kapitels "Kapitel Eins".
`/dok/kap`	selektiert alle kap-Elemente innerhalb des dok-Elements

Als Ergebnis der Auswertung eines Pfadausdrucks, wird ein Objekt geliefert. Es gibt vier Grundtypen zu denen ein Ergebnisobjekt gehören kann. Diese sind die duplikatfreie, ungeordnete Menge (*node-set*), der Boolescher Wert (*boolean*), die Zeichenkette (*string*) und die Gleitkommazahl (*number*) oder eine Menge von atomaren Werten.

3.1.2 SXP

Das SXP (*Simple XPath*) ist, wie der Name schon sagt, es eine vereinfachte Sprache, die einige Elemente der XPath-Sprache nicht enthält. SXP wird hier, wegen in dem Abschnitt 4.1 über SPEX-System erwähnte FSXP-Anfragesprache kurz erläutert. FSXP stellt eine Teilmenge von SXP dar. In dieser Version werden nur die Vorwärtsachsen verwendet, da die Rückwärtsachsen auf Datenströme nicht angewendet werden können. Im Abschnitt 4.1 werden zu dieser Anfragesprache mehr Erläuterungen geben.
In SXP, wie auch in XPath, sind die Vorwärts- und Rückwärtsachsen, mit Ausnahme von *ancestor-or-self* und *descendant-or-self* enthalten. Die Knotentests und die Operationen, wie *union*, *intersect* und *except* sind auch in dieser Sprache enthalten. Wertebasierte Vergleiche der Ergebnisse von Prädikaten, zum Beispiel [`child::a = desc::a`] und von positionierten Prädikaten, wie [`position() = 3`] gehören nicht zu SXP.

3.1.3 Unterschiede zwischen der alten und der neuen Version von XPath

Im Rahmen der Entwicklung der Sprache XQuery wurde XPath 2.0 vom W3C (*World Wide Web Consortium*) standardisiert. Eine der wichtigsten Änderungen ist die Verwendung des Typsystems von *XML Schema*. XPath 1.0 benutzte nur die Knotenmenge, boolesche und numerische Werte und Zeichenkettenwerte. Außerdem basiert XPath 2.0 im Vergleich zu der älteren Version auf Sequenzen und nicht auf Mengen und unterstützt zusätzlich Referenzen. Diese Sprache kann mit Dokumentkollektionen arbeiten und vor den Pfadausdrücken können Variablen oder Funktionen vorangestellt werden. Im Vergleich zu XPath 1.0 wird in XPath 2.0 erlaubt, zwischen Wertegleichheit und Knotenidentität, sowie Bereichsausdrücke in Prädikaten zu unterscheiden. Die Funktionsbibliothek in dieser Sprache ist um einiges vergrößert worden.
Die Betrachtung der neuen Version der XPath-Sprache ist deswegen wichtig, weil auf dieser die Sprache XQuery basiert, die in dieser Ausarbeitung später aufgeführt wird. Es wird auch auf die Bedeutung dieser Sprachen bei der Bearbeitung der XML-Datenströme hingewiesen.

3.2. XQuery und XQuery-Prozessor

XML-Datenströme stellen eine große Herausforderung für die Anfragesprachen da. XQuery ist eine komplexe Anfragesprache. Die Semantik dieser Sprache ist ähnlich der Semantik der Sprachen SQL (*Structured Query Language*) und OQL *(Object Query Language*), außerdem basiert diese Sprache auf der Anfragesprache XPath 2.0. [Lezi03]

XQuery ist eine deklarative und funktionale Sprache. Die Berechnung der Anfragen ist als eine Transformation der XML-Dokumente zu betrachten. Der XQuery-Prozessor wertet eine von einer XQuery-Anwendung abgesetzte XQuery-Anfrage aus und liefert sie dann als eine Sequenz von einfachen Werten oder XML-Fragementen zurück. Die Funktionsbibliothek, das Datenmodell sowie die Operationsbibliothek gehören auch zu den Gemeinsamkeiten von XQuery. Und natürlich nicht zu vergessen ist die Tatsache, dass diese beiden Sprachen auf der Anfragesprache XPath 2.0 aufgebaut sind.

Obwohl eines der Ziele, die bei der Entwicklung von XQuery verfolgt wurden, die Optimierbarkeit war, lässt sich diese Sprache nicht so einfach optimieren. Doch bestimmte Techniken, die im Zusammenhang mit der Optimierung von funktionalen Sprachen eine Rolle spielen, lassen sich auch bei XQuery anwenden.

Im Zusammenhang mit der Entwicklung der BEA's WebLogic Integration (WLI) Plattform 8.1 wurde der XQuery-Streaming-Prozessor konstruiert [FHK+03]. Das Ziel war bei diesem Projekt eine bessere Leistung der Anwendungen, die XML-Datenströme bearbeiten, zu erlangen. Seine Bestandteile sind Pakete, die als eine Java-Bibliothek implementiert sind, um den Prozessor ohne große Schwierigkeiten in verschiedene Java-Anwendungen, einbetten zu können. Die Arbeitsweise des Strom-Prozessors wird in der Abbildung 1 skizziert und im Folgenden beschrieben.

Die Ein- und Ausgabedaten werden als Token-Strom dargestellt. Die Java-Anwendungen bearbeiten die XQuery-Anfragen und leiten die Resultate durch die XDBC-I*nteface* weiter. Dies ist eine Java-Schnittstelle, deren Abkürzung von JDBC (*Java Database Connectivity*) abgeleitet wurde. Sie hat auch die gleichen Funktionen wie JDBC-Schnittstelle [FHK+03]. Der Anfragekompilierer führt die syntaktische Analyse durch, überprüft den Typ der Anfrage und optimiert diese. Es wird ein Anfragenetzwerk generiert, der die Form eines Baumes besitzt und als Knoten Operatoren enthält. Diese Operatoren werden aus der XQuery-Anfragealgebra übernommen. Sie bekommen den Token-Strom als Eingabe, bearbeiten diesen und produzieren eine Ausgabe als Ergebnis, die dann der Wurzel des Operatorbaumes an das System zurückliefert. Bei der Erstellung des Anfragebaumes wird in der Regel eine Optimierung durchgeführt. Dabei wird der Baum in einen anderen, optimierten Baum transformiert. Die Semantik muss auf jeden Fall gleich bleiben. Das heißt der erzeugte Baum, soll die gleichen Resultate wie vor der Optimierung liefern. Der Optimierungsablauf ist auf der unteren Abbildung nicht dargestellt, um diese nicht zu kompliziert zu machen. Die Laufzeitumgebung wertet den Operatorbaum gegen die

konkreten Daten aus. Dieses System besteht aus Implementierungen aller Funktionen und Operatoren der XQuery-Bibliothek. Die Komponenten „XML-Parser“ und „Schema-Validator“ dienen der Bearbeitung von externen XML-Daten, wenn diese ein Teil der Anfrage sind. Die einkommenden XML-Nachrichten werden geparst, schematisch validiert und in einem speziellen Format abgespeichert. So können Diese Nachrichten in weiteren Anfragen benutzt werden, ohne bei jedem Anfrageaufruf diese kostspieligen Operationen noch einmal durchführen zu müssen. Sie werden als freie Variablen zu den Anfragen gebunden. Die Variablenbindungen werden mit Hilfe des XDBC-Interfaces durchgeführt [Beck04].

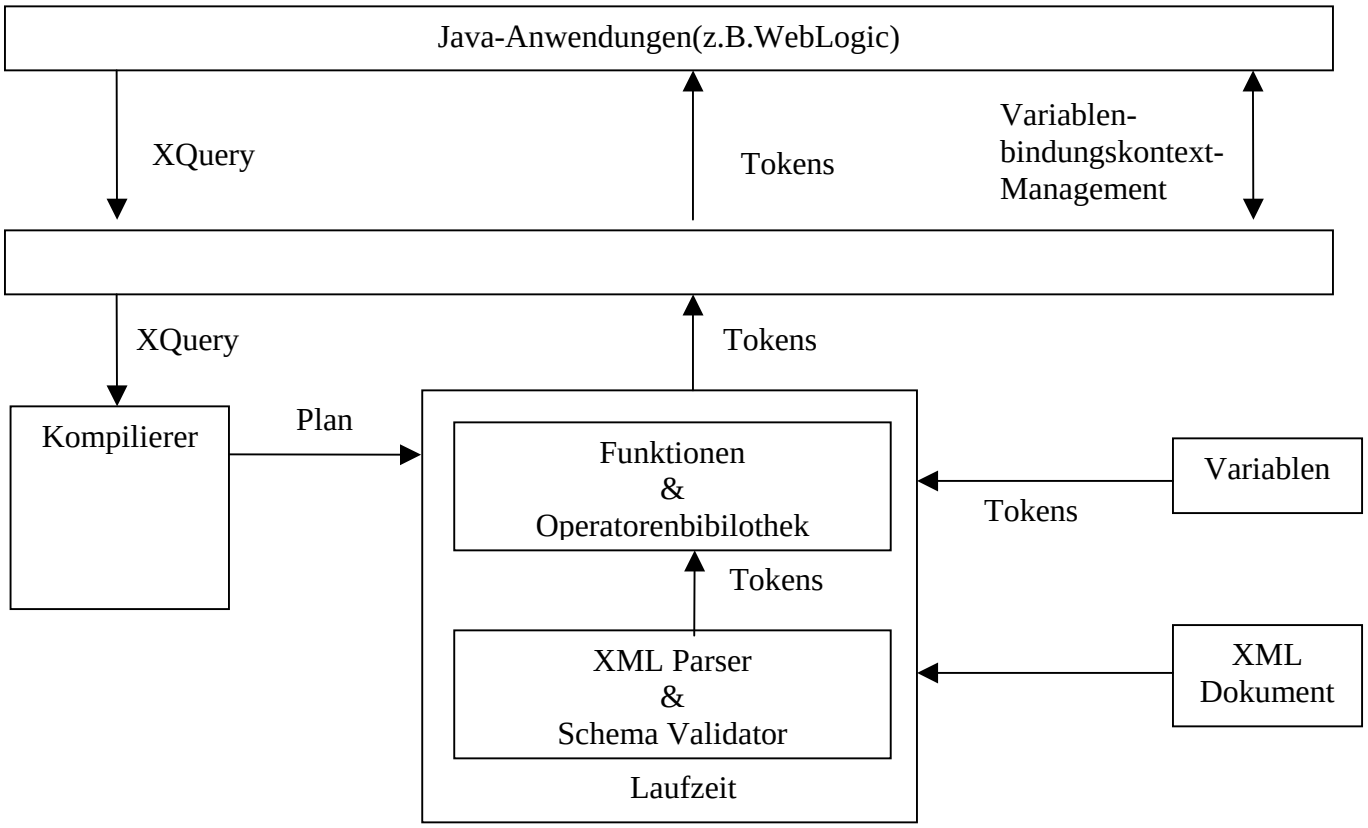


Abbildung 1: Auswertung durch den XQuery Prozessor

Wie schon oben erwähnt sind die XML-Daten im Streaming-XQuery-Prozessor als Token-Ströme vertreten oder es wird ein Parser benutzt, der XML-Daten, die in anderen Formaten repräsentiert sind, in das Token-Strom-Format umwandelt. Die Bearbeitung solcher Token-Ströme beansprucht, im Vergleich zu XML-Bäumen wie sie im oben beschriebenem Modell DOM erstellt werden, den Speicherplatz des Prozessors minimal. Während der Laufzeit verarbeitet jeder Laufzeitoperator seine Eingabe, ein Token nach dem anderen. Die Eingangsdaten, die nicht angefordert werden, werden einfach verworfen. Seriell auszuführende Anfragen sind auf beliebig großen Datenmengen möglich.

Es gibt eine Reihe von Werkzeugen, die eine Redefinition von Parser, Serialisierer und Adapter erlauben, die dann einen XQuery-Prozessor mit den meisten XML-Verarbeitungsanwendungen koppeln können. Um unterschiedliche Implementierungen zu erlauben, ist der Token-Strom als eine Java-Schnittstelle definiert. Eine Default-Implementierung benutzt einfache Java-Objekte.
Der beschriebene Prozessor kann auch dazu benutzt werden, um den Zeichen-Strom zu einem XML-Text umzuwandeln oder aus ihm eine DOM-Instanz zu konstruieren [FHK+03].

3.3 Optimierung von Anfragen gegen XML-Ströme

Es wurde schon gesagt, dass es bei der Bearbeitung von XML-Datenströmen wünschenswert ist, die Daten nicht speichern zu müssen, sondern sie sofort zu bearbeiten. Bei einem beschränkten Speicherplatz und riesigen Mengen an Daten ist es sogar unmöglich alle Daten zwischenzuspeichern. Die Reihenfolge der Daten in einem Strom legt auch die Bearbeitungsreihenfolge innerhalb der Operatoren, die die Elemente bearbeiten, fest. Die Operatoren können bei der Durchführung der Berechnungen möglicherweise andere Elemente des XML-Stroms brauchen, die noch nicht angekommen sind. Doch man kann nicht alle Daten, die eventuell zu einem späteren Zeitpunkt gebraucht werden, speichern. Bei der Bearbeitung der Datenströme ist es wichtig, so wenige Daten zu speichern und so schnell ein Resultat zu liefern wie nur möglich.
Der Zugriff auf die Daten wird auch von der Reihenfolge des Eingangs dieser Daten festgelegt. Deswegen sind die wichtigsten Ziele bei der Optimierung der Anfragen gegen die XML-Ströme folgendermaßen festgelegt. Als erstes ist es wichtig, ein optimales Anfragenetzwerk zu generieren, das die, von dem Strom gegebene Datenzugriffsreihenfolge, beachtet. Dabei soll der Zugriff auf ältere Daten soweit wie möglich vermieden werden.
Um ein Anfragenetzwerk zu optimieren, kann man nicht einfach die Operatorpositionen umordnen, denn die Positionen dieser Operatoren sind, wie schon oben erwähnt, festgelegt. Wurde aber ein Netzwerk gefunden, das optimal und äquivalent dem gegebenem ist, dann können die Operatoren entsprechend umgeordnet werden. Eine der Möglichkeiten, das Query-Netzwerk zu optimieren, ist die Ersetzung der Operatoren durch andere passende Operatoren. Ein Beispiel für eine solche Optimierung wird in Abbildung 2 demonstriert. Angenommen, zwei Zweige eines solchen Operatorbaumes an der Verzweigungsstelle den gleichen Operator haben und nach der nächsten Operation auch das gleiche Ergebnis benutzen, dann kann man diese Zweige miteinander verknüpfen [Furc03].

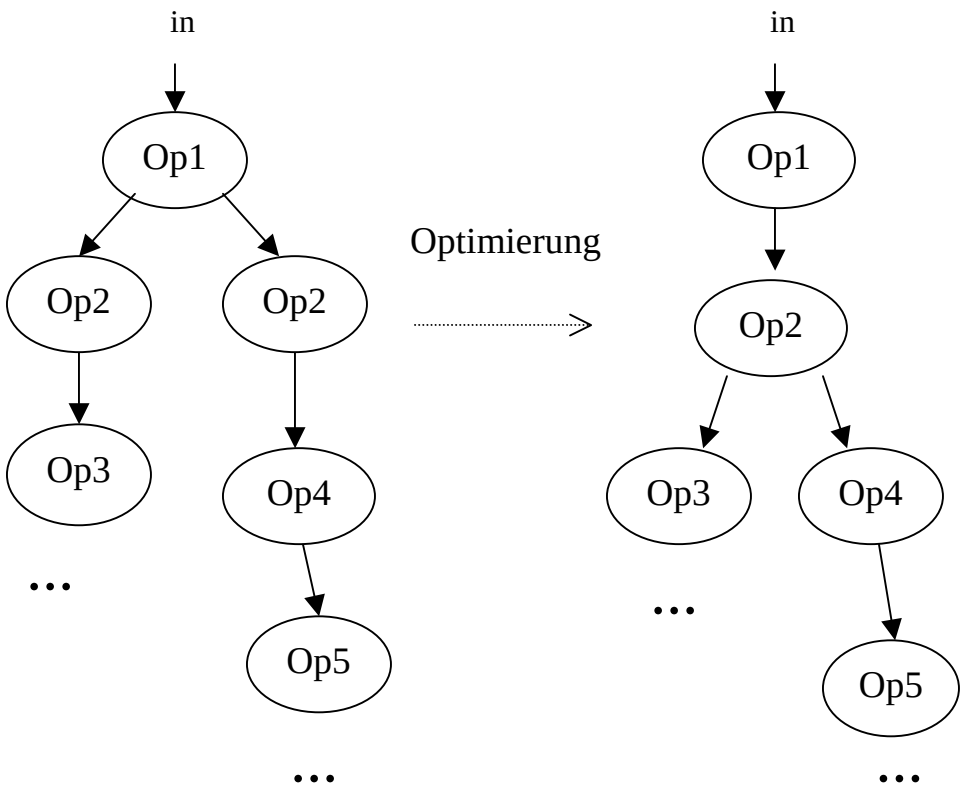


Abbildung 2: Beispiele für Optimierung der Anfragen

4 Systeme zur Bearbeitung der XML-Ströme

Die Lösungskonzepte, mit denen man eine Anfrage gegen einen XML-Strom auswerten kann ohne die Daten speichern zu müssen, werden immer öfter benötigt. Außerdem sollten diese Lösungen den höchstmöglichen Datendurchsatz und möglichst kurze Antwortzeiten aufweisen. SPEX ist ein System, das die Auswertung der XPath-Anfragen gegen XML-Datenströme ermöglicht.

XMLTK ist ein System bestehend aus mehreren Werkzeugen mit deren Hilfe die XML-Datenströme unterschiedlich bearbeitet werden können. Diese Tools werden in zwei Bereichen unterteilt. Ein Bereich besteht aus einfachen Operationen, die XML-Ströme bearbeiten. Die zweite XMLTK-Komponente ist der *XML-Stream-Processor* (XML-Strom-Prozessor). Um die XML-Datenströme zu verarbeiten benutzt XML-Strom-Prozessor den so genannten *Stream Index* (SIX).

Im Folgenden werden diese beiden Systeme ausführlicher dargestellt.

4.1. SPEX

SPEX bedeutet *Streamed and Progressive Evaluator for XPath.* Dieses System verwendet ein Netzwerk von deterministischen Kellerautomaten, um Anfragen gegen einen XML-Strom auszuwerten. Solche Anfragen werden in einem einzigen sequentiellen Durchlauf ausgewertet. Das ist ein Vorteil von SPEX, denn es kann für Systeme mit kleinem Speicher und schwacher Rechenleistung verwendet werden [BCD+04]. Auch hier wird XPath als Anfragesprache benutzt, wenn auch in einer etwas eingeschränkter Version. Diese Version wird FSXP (*Forward Simple XPath*) genannt. Da der Zugriff auf die Daten bei der Anfrage gegen ein XML-Strom nicht wahlfrei erfolgen kann, sind die Rückwärtsachsen in diesem Fall unerwünscht. Solche Achsen sind zum Beispiel *parent* oder *ancestor*. Wenn eine SXP-Anfrage in die FSXP-Anfrage überführt wird, ändert sich nicht die Semantik, sondern nur syntaktische Aufbau der Anfrage. Für die Umschreibung wird die Symmetrie der Achsen verwendet. Diese Symmetrie ist in der folgenden Tabelle abgebildet:

Rückwärtsachsen	**Vorwärtsachsen**
parent	child
ancestor	descendant
ancestor-or-self	descendant-or-self
preceding	following
preceding-sibling	following-sibling

Die Gesamtauswertung sieht in SPEX wie folgt aus. Als erstes wird die eingegebene XPath-Anfrage in äquivalente FSXP umgeschrieben und optimiert. Im folgenden Beispiel kann man so eine Umformung erkennen.

Von einem Betriebssystem werden mehrere Prozesse bearbeitet. Diese Prozesse können mehrere Subprozesse enthalten. Angenommen, dass in diesem Beispiel eine solche Prozesshierarchie als ein XML-Baum dargestellt ist. In diesem Fall sind die Prozesse die XML-Elemente, also Konten, deren Kinderelemente „`time`“ (Laufzeit), „`memory`“ (Speicher), „`priority`“ (Priorität), „`state`“ (Zustand) und Subprozesse sind. Die Laufzeit wird in Stunden, Speicher in MB und Priorität in ganzen Zahlen ausgedrückt. Jeder Prozess befindet sich in einem Zustand, wie „`stopped`“ für gestoppt oder einem anderen Zustand.

Beispiel 4:

Eingabe (XPath-Ausdruck):

```
/desc::process[child::time > 24 or child::memory > 500]/anc::process[child::priority <
10 and child::state = ”stopped”]
```

Ausgabe (FSXP-Ausdruck):

```
/desc::process[child::priority < 10 and child::state = ”stopped” and
desc::process[child::time > 24 or child::memory > 500] ]
```

Als erstes werden in der angegebenen XPath-Anfrage alle Prozesse gesucht, die länger als 24 Stunden ausgeführt werden oder mehr als 500MB des Speicherplatzes ausnutzen. Aus der resultierten Menge kommen dann Prozesse, deren Priorität kleiner als 10 ist und den aktuellen Zustand „`stopped`“ haben, in die Anfrageergebnismenge. Nach der Umformung bekommt man als Ausgabe folgende Anfrage: Suche alle Nachfolger deren Prozesse, die oben beschriebe Prädikate erfüllen. Diese Anfrage wird dann zu einem entsprechenden logischen Netzwerk, das aus den Transduktoren besteht, kompiliert. Abbildung 3 enthält das logische Netzwerk zur Ausgabe von Beispiel 4. Jeder Knoten, der sich im Pfad befindet, wie zum Beispiel ein Prädikat oder eine Achse, entspricht einem Knoten aus dem Netzwerk. Die Komponente, die in einem rechteckigen Kasten dargestellt wird, ist das gesuchte Ergebnis. Die, in den Kreisen dargestellte Komponente sind Prädikate oder Teile davon. Hell gefärbte Komponente entsprechen den Achsennamen und grauen den Operatoren.

Im dritten Schritt wird aus diesem Netzwerk einen passenden physischen Anfragegraphen, das so genannten Transduktor-Netzwerk (oder auch Kellerautomatenetzwerk) generiert. Jeder Operator wird als Kellerautomat realisiert. In der Abbildung 4 ist das zum Beispiel 4 entsprechende Netzwerk skizziert.

Solch ein Netzwerk berechnet die Antworten auf die Anfragen gegen den XML-Strom. Am Anfang jedes Kellerautomatennetzwerks befindet sich ein *in-Transducer*, der den Beginn eines XML-Stroms markiert. Am Ende gibt es ein Subnetzwerk genannt *funnel*. Die Transduktoren in diesem Subnetzwerk sind dafür zuständig, die berechneten Antworten auf die Anfragen zusammenzufassen. Der letzte Automat in dem Netzwerk heißt *out*. Dieser Automat puffert die Antworten auf die Anfragen und leitet sie dann weiter.

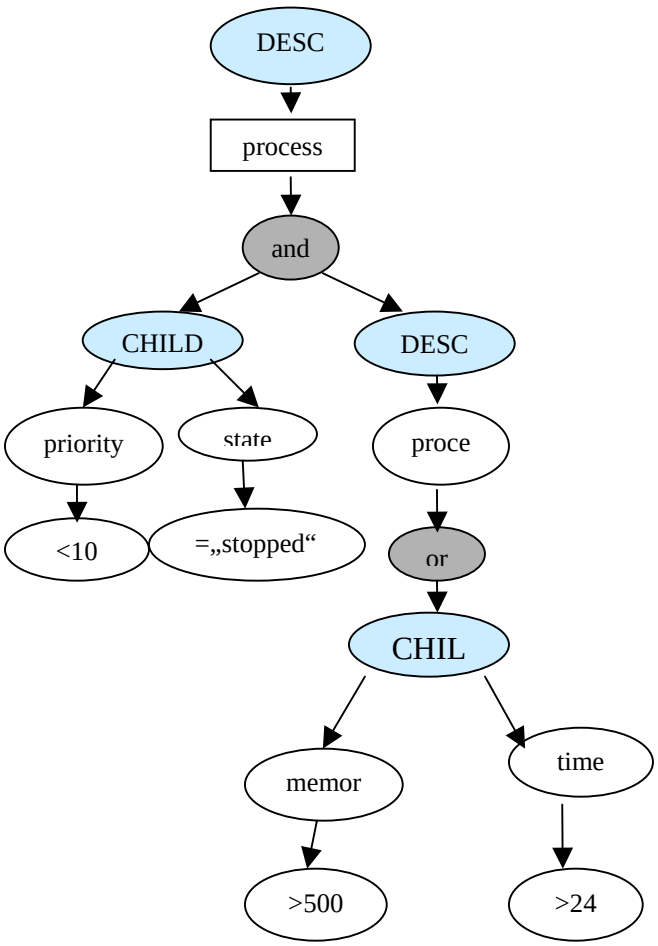


Abbildung 3: logisches Netzwerk

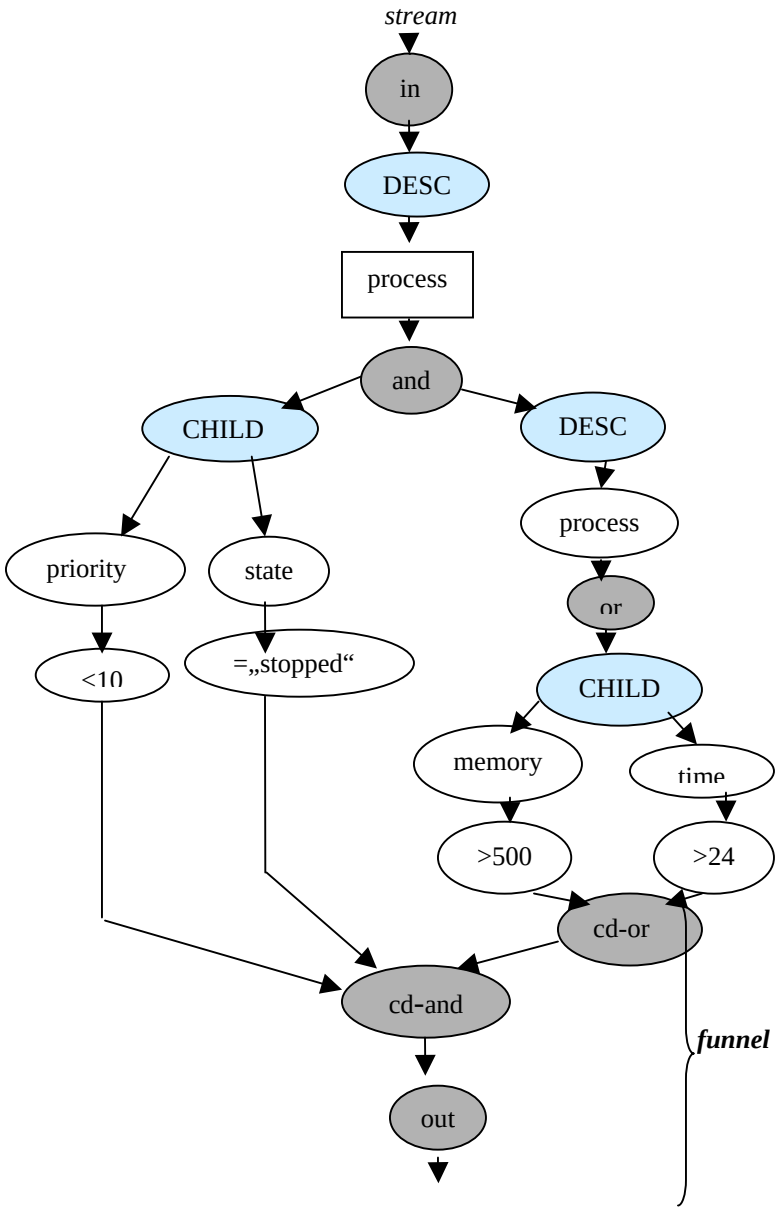


Abbildung 4: Kellerautomatennetzwerk

Jedes Prädikat in der Anfrage hat ein entsprechendes Paar von Transduktoren. Zum Beispiel, *or*- hat als Paar ein *cd-or*-Automat und *and*-Automat ein *cd-and*-Automat. *And*- und *or*-Transduktor werten die Prädikate aus. Sie entsprechen den Prädikaten in der FSXP-Notation. Der *cd*-Transduktor verknüpft die Ergebnisse der Auswertungen und leitet die weiter. Dabei steht *cd* für *condition determinant*.

In der vierten Phase leitet der Ausgabe-Strom die Antworten auf die Anfragen weiter. Die Keller der Automaten im Netzwerk werden dazu benutzt, die Tiefe eines XML-Baumes des zugehörigen Stromes zu ermitteln und zu annotieren. Außerdem kann mit Hilfe von *Stacks* auch die relative Position eines Elementes im Strom berechnet werden. Zum Beispiel werden so die Geschwisterbeziehungen erkannt und weitergegeben. Der XML-Strom wird von einem Automaten zum anderen geleitet. Dies geschieht

entweder unverändert oder entsprechend annotiert. Die Ergebnis-Transduktoren sind, wie oben schon beschrieben, in den eckigen Boxen dargestellt. Sie werden auch als *Head* bezeichnet und kommen in einem Netzwerk nur einmal vor. Solche Elemente sind nur als potentielle Antworten zu sehen, da sie noch von Prädikaten abhängig sein können. Solange werden diese Ergebnisse im *out-Transducer* gepuffert. Die Prädikatabhängigkeiten werden auch als Informationen bei jedem Automaten annotiert. Es kann aber unter Umständen erst später entschieden werden, ob das Ergebnis korrekt ist oder nicht, denn die notwendige Information kann erst im später ankommenden Strom enthalten sein.
Mit SPEX können die XPath-Anfragen gegen XML-Ströme von unbegrenzter Größe und Tiefe ausgewertet werden. Um beispielsweise Sichtkonzepte zu realisieren, kann man auch die angenäherten Antworten mit SPEX auswerten.

4.2 XMLTK

XMLTK (*XML Toolkit*) ist ein System aus Werkzeugen, das sich in zwei Komponenten unterteilen lässt. Die erste Komponente ist eine Reihe von Werkzeugen, die einfache Operationen auf den XML-Daten ausführen können. Es sind Funktionen, wie zum Beispiel Sortieren und Aggregation, Schachtelung, sowie viele andere einfache XML-Transformationsfunktionen. Eines der wichtigsten dieser Tools ist der XML *Stream Index* (SIX). Für eine komplexere Verarbeitung von XML-Daten können die Werkzeuge zu Pipelines kombiniert werden.
XMLTK definiert ein binäres XML-Zwischenformat. Dieses Zwischenformat ist kompakter als ein XML-Format und kann zu Zwecken der Kommunikation der Werkzeuge in der Pipeline benutzt werden. Es benutzt eindeutige Nummern für jeden auftretenden Elementnamen und vereinfacht damit Vergleiche zwischen XML-Bezeichnern. Die zweite wichtige Komponente, die das XML Toolkit vervollständigt, ist ein höchst skalierbarer *XML-Stream-Prozessor*.
Durch den Austausch von XML-Text ist eine Integration mit anderen XML-Tools auf der Betriebssystemebene möglich.

4.2.1 XPath-Prozessor für XML-Ströme

Das zweite Teil, der das XML Toolkit vervollständigt, ist der *XML-Stream-Prozessor*. Die API (*Application Programmer Interface*) des XPath-Prozessor basiert auf Programmiersprache C. In der Abbildung 5 sieht man die Architektur dieser API. Alle Werkzeuge, die oben als das erste Bestandteil des

XMLTK erwähnt wurden, benutzen dieses Interface um die XPath Ausdrücke auszuwerten. Die API definiert ein einfaches *XML-Prozessmodell*, das ereignisbasiert ist. Die zentrale Aufgabe des XPath-Stromprozessors ist, Identifizieren der Übereinstimmung der *Variablen* mit Eingabe-XML-Strom. Die Variablen sind Markierungen (*labels*) für die Knoten eines Anfargebaumes.

Der XPath-Prozessor nimmt den Anfragebaum und von den TSAX(*Token Simple API for XML*)-Parser generierten Strom aus TSAX-Ereignissen und definiert neue Variableereignisse.

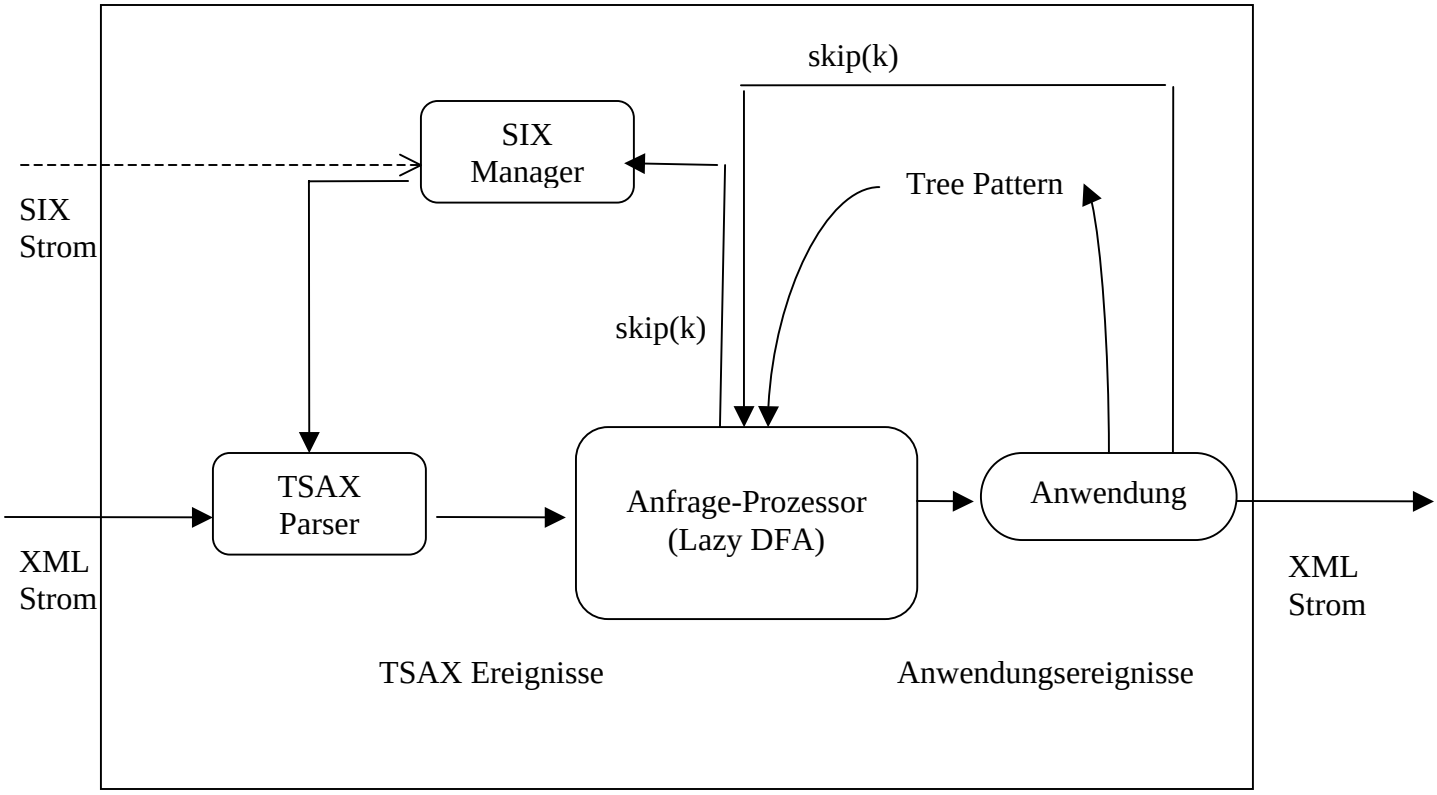


Abbildung 5: Systemarchitektur

Der XPath-Prozessor konvertiert einen gegebenen Anfragebaum zuerst in einen nichtdeterministischen endlichen Automat, dann aber überführt ihn in einen entsprechenden deterministischen endlichen Automat (DEA). Bei der Abarbeitung eines DEA wird ein Stack verwendet. Mit dem Eintretten eines `startElement`-Ereignisses geht der Prozessor in den nächsten aktuellen Zustand des DEA und legt den alten Zustand auf den Stack. Wenn ein `endElement`-Ereignis auftritt, nimmt der Prozessor den Zustand vom Stack und betrachtet ihn als den aktuellen Zustand. Die Endzustände des DEA haben eine Menge von zugehörigen Variablen. Sobald ein Endzustand erreicht wird ist ein passendes Variablenereigniss für jede Variable in der Menge generiert.

Der Stack dieser Automaten ist maximal so groß, wie die maximale Tiefe des entsprechenden XML-Dokumentes. Am Anfang wird auf dem Stack ein bestimmter Bereich belegt, der sich dann, bei Bedarf, verdoppeln lässt. Als Folge erzielt der XPath-Prozessor einen konstanten Durchsatz, der von der Zahl der XPath-Ausdrücken unabhängig ist.
SIX-Manager reguliert die Funktion, Überspringen (*skip(k)*) der XML-Elemente. Das *k* bedeutet die Anzahl der Elemente, die übersprungen werden. Im nächsten Abschnitt ist diese Funktion im Zusammenhang mit SIX genauer beschrieben.
Das Ziel bei der Entwicklung solcher Prozessoren ist eine effiziente Verarbeitung sehr großer Mengen von XPath-Ausdrücken auf XML-Strömen [IGG+03].

4.2.2 Stream IndeX (SIX)

Für das XML-Strom ist der SIX ein binärer Strom, der für jedes gegebene Element eines XML-Stroms aus Paaren der Form *beginOffset* und *endOffset,* besteht. Ein Byte kennzeichnet jeweils der Beginn und das Ende eines Elements. SIX kann mit dem XML-Strom synchronisiert werden. Der XPath Prozessor vergleicht SIX Einträge mit den Tags im XML-Strom. Mittels *endOffsets* werden einige Zeichen im XML-Strom übersprungen, wenn die aktuellen Elemente von dem Prozessor als nicht notwendig befunden werden. Dabei wird der Inhalt von diesen übersprungenen Teilen nicht geparst. Im Beispiel 5 ist ein XML-Fragment dargestellt, der die Tags `<dok>`, `<kap>, <pa>` und `<book>` enthält. Angenommen es wird in diesem Fragment nach einem Element `dok/book` gesucht, dann kann also das Element `kapitel` übersprungen werden.

Beispiel 5:

```
<dok>
      <!-- ein XML-Dokument -->
      <kapitel title="Kapitel Eins">
            <pa>Erster Satz</pa>
            <pa>Zweiter Satz</pa>
      </kapitel>
      <book title="Kapitel Zwei">
            <pa>Erster Satz</pa>
      </book>
</dok>
```

Je größer solche Teile im XML-Dokument sind, desto besser ist der Durchsatz des XPath-Prozessors.

4.3 XML-Filter-Systeme

Die Systeme XFilter und die YFilter werden auch *Publish-Subscribe-Systeme* genannt. Mittels dieser Systeme, verteilen die *Publisher* (Herausgeber, Verleger) unter *Subscribern* (Abonennten) die Informationen und Ereignisse, die sie abonniert haben. Solche Informationen können zum Beispiel Nachrichten, Aktienkurse, Unterhaltung und viele andere sein. Am häufigsten finden die Publish-Subscribe-Systeme ihre Anwendung zum Beispiel im Internet.
Wie der Name schon sagt, werden diese Systeme zum Filtern der XML-Datenströme eingesetzt. Ein Filtersystem besitzt in ihrer Architektur eine *Filtering-Engine*, durch die angekommene Ströme durchgeleitet werden. In dieser Filtering-Engine werden die XML-Dokumente mit den Anfragen verglichen und die Dokumente, die auf diese Anfragen passen, werden an viele Benutzer und Systeme, die an diesen Daten interessiert sind, weitergeleitet. Die zentrale Problemstellung der Filter-Systeme ist das Finden einer Teilmenge aus einer Menge von XPath-Ausdrücken. Diese Teilmenge soll auf den eingegebenen XML-Dokument passen [Eise03].
Als Anfragesprache wird bei solchen Filtersystemen am häufigsten XPath verwendet, da XPath die absolute und relative Adressierung separater Elemente der XML-Dokumente erlaubt. Da ein XML-Dokument als ein Baum repräsentiert ist, sind die XPath-Ausdrücke somit eine Art Muster, die auf die Knoten des XML-Baumes angewendet werden. Die Resultate der Auswertungen dieser Muster sind dann Objekte von unterschiedlichen Typen, zum Beispiel *boolean*, *string* oder *number*. Man kann Texte und Attribute filtern oder auch nur Attribute (wertebasiert). Es gibt die Möglichkeit der strukturbasierter, also auf die Position bezogener Anwendung der Filter. Sogar Pfadausdrücke wie *nestdet paths* (geschachtelte Pfade) können die Filter beinhalten. Ein Beispiel für einen Ausdruck mit *nestdet paths* sieht wie folgt aus: `//product[price/msrp<300]/name`.
Ein wichtiges und umfassendes Problem bei solchen Systemen stellt vor allem die effiziente Ermittlung und Identifikation passender XPath-Ausdrücke zu einem gegebenen XML-Dokument dar.

4.3.1 XFilter

In diesem Abschnitt wird das XFilter-System für die XML-Datenströme dargestellt. Eine wichtige Komponente dieses System ist die *Filtering-Engine*. Diese Komponente verwendet die dynamische Indexstruktur und den *Query-Index* von XMLTK. Außerdem wird hier der Ansatz der FSM (*Finite State Machines*) benutzt. Unter FSM versteht man ein System, das in sich geschlossen ist und durch eine Menge

von Zuständen und Übergängen zwischen diesen Zuständen definiert wird. Der Anlass für den Übergang eines Zustandes kann entweder spontan auftreten oder von außen kommen. Dieser Ansatz ist vor allem in der Internet-Umgebung relevant, um die erforderlichen Benutzerprofile, also die XPath-Anfragen, schnell zu lokalisieren und zu überprüfen. Der event-basierte *XML-Parser* steuert den Überprüfungsprozess. Nach dem Parsen eines XML-Dokuments verschickt der Parser Ereignisse an die *Filtering-Engine*. Um ein XML-Dokument zu überprüfen, müssen in XPath spezifizierte Nutzerprofile verfügbar sein, die dann nach dem Parsen mit dem *XPath-Parser* als *Path-Nodes* in die Maschine eingegeben werden. Diese Knoten repräsentieren die Elementknoten einer Anfrage und gleichzeitig sind sie die Zustände der einzelnen FSM. Sie werden dann dem *Query-Index* übergeben. Eines dieser ist als eine Hashtabelle organisiert und basiert auf Elementnamen aus dem XPath-Asudruck. Mittels eines *Execution Algorithm* werden in der Engine die Nutzerprofile für ein XML-Dokument identifiziert und die gefilterten Daten mit dem *Data-Dissemination*-Komponente weitergeleitet [Eise03].

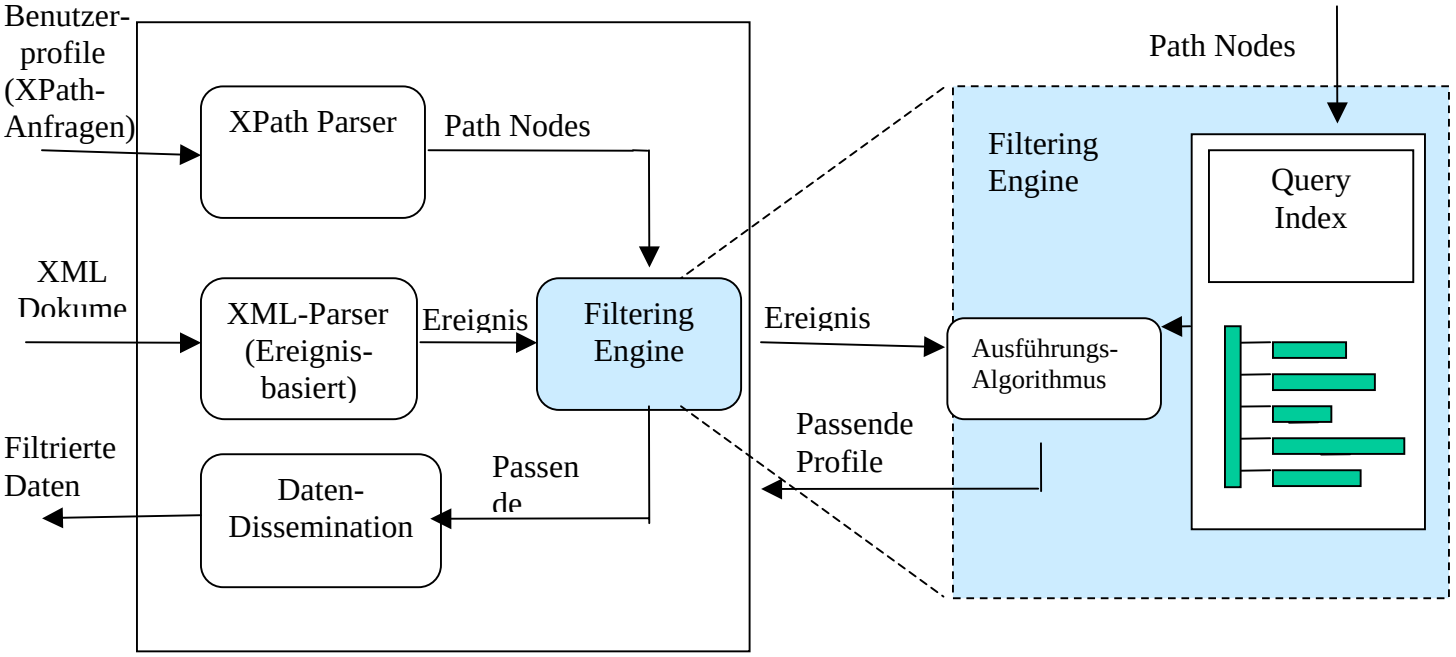


Abbildung 5: Architektur von XFilter

Im XFilter-System werden die XPath-Ausdrücke als Prädikate auf die XML-Dokumente angewendet. Sobald mindestens ein Element des Dokuments durch einen XPath-Ausdruck adressiert werden kann, wird das Dokument als passend erkannt. Ein Beispiel ist in der Abbildung 6 dargestellt. Hier genügt der XML-Fragment den Ausdrücken q1 und q2, aber nicht q3.

```
q1: /catalog/product//msrp
q2: //product/price
    [@currency = “USD”]/msrp
q3://product[price/msrp<300]/name
```

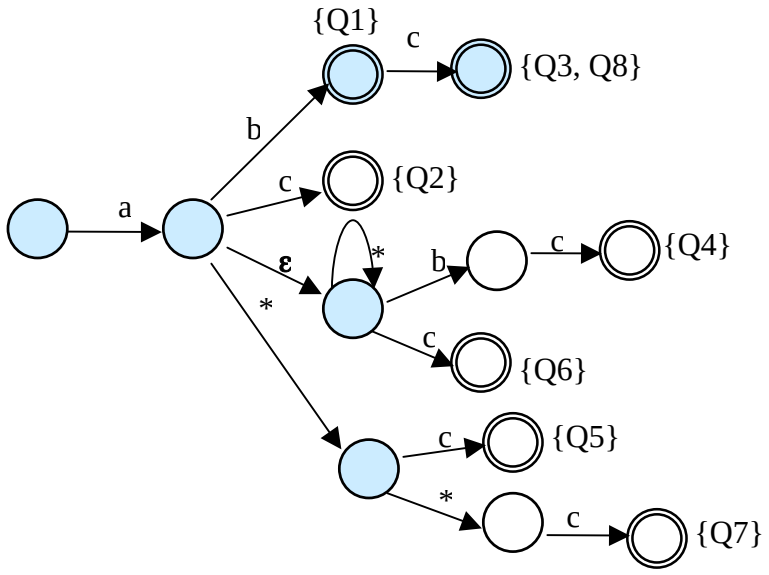
```
<?xml version="1.0"?>
<catalog>
<product id="Kd-245">
<name> Color Monitor </name>
<price currency="USD">
<msrp> 310.40 </msrp>
</price>
</product>
</catalog>
```

Abbildung 6: XPath-Ausdrücke und XML-Dokumentfragment

4.3.2 YFilter

Der YFilter stellt sozusagen eine Verbesserung des XFilter-Algorithmus dar. Die Idee ist dabei die Ausnutzung der Gemeinsamkeiten der Pfadausdrücke, um den Aufwand durch wiederholtes verarbeiten zu minimieren. In XFilter existierte für jede XPath-Anfrage eine FSM. Bei YFilter wird jede Anfrage in eine einzelne FSM kombiniert. Diese wird dann zum nichtdeterministischen endlichen Automaten (engl.: *Nondeterministic Finite Automaton*), kurz NFA. Ein Beispiel für solchen Automaten, der die acht Queries in Abbildung 7 a) repräsentiert, ist in der Abbildung 7 b) zu sehen. Ein Zustand wird als Kreis repräsentiert. Doppelt gezeichnete Kreise deuten auf einen Endzustand hin, der auch als akzeptierender Zustand bezeichnet wird. Durch einen Pfeil wird der Übergang von einem in den nächsten Zustand gekennzeichnet und die Symbole entsprechen den Eingabeelementen einer Anfrage. Das Symbol „*“ steht für die Eingabe eines beliebigen Symbols und „ε“ für leere Eingabe. Die hell gefärbten Kreise stellen die Zustände dar, die von mehreren Anfragen verwendet werden. Um für einen Pfadausdruck ein NFA zu konstruieren, wird zuerst für jeden Lokalisierungsschritt einen NFA-Fragment erzeugt. Dann werden diese Fragmente zu einem NFA zusammengebaut.

Q1=/a/b
Q2=/a/c
Q3=/a/b/c
Q4=/a//b/c
Q5=/a/*/c
Q6=/a//c
Q7=/a/*/*/c
Q8=/a/b/c



a) Path-Anfragen b) NFA (YFilter)

Abbildung 7: Path-Anfragen und zugehöriger NFA (YFilter)

In einem etwas vereinfachten Modell des YFilters werden die XPath Anfragen so beschränkt, dass beim Filtern keine Auswertung nötig ist. Zum Beispiel wären solche Ausdrücke, wie `/catalog/product[position() = 2]`, bei denen die Position oder ein Attribut, wie hier `//product/price[@currency = ”USD”]` berechnet werden sollen, nicht zulässig. Denn bei der direkten Kodierung der Prädikate, wäre die Ausnutzung gemeinsamer Zustände unter den Anfragen sehr stark eingeschränkt. Dadurch hätte der YFilter seinen Vorteil gegenüber dem XFilter-System verloren. Es gibt dennoch ein Paar Ansätze, die dieses Problem des YFilter-Systems etwas entschärfen. Beim ersten Lösungsansatz, genannt *Inline*, werden die wertebasierten Prädikate, die einem Zustand zugeordnet sind, in eine Tabelle gespeichert. Dabei werden diese Prädikate zur weiteren Erkennung mit einer Query- und einer Prädikaten-ID gespeichert und beim Eintreffen eines Start-Elements für jeden aktiven Zustand überprüft. Bei einem anderen Lösungsansatz kommen die Auswahlprädikate erst später zum Einsatz. Zuerst wird die Struktur des Dokumentes überprüft. Die mit jeder Anfrage abgespeicherte passenden Prädikate werden mit einer Reihenfolgennummer vermerkt und dann für jede strukturüberprüfte Anfrage ausgewertet. Dieser Ansatz wird *Selection Postponed*, kurz SP, genannt. Der Vorteil des Ansatzes ist, der schrittweise Zusammenbau des NFA. Dies ist deswegen nützlich, weil dann die neuen Anfragen leicht hinzugefügt werden können.

5. Zusammenfassung

Ein Datenstrom ist eine kontinuierliche Folge von potentiell unbegrenzten Datensätze. In meisten Fällen ist es unerwünscht oder sogar technisch unmöglich den kompletten Datenstrom zu speichern. Die sequentielle Repräsentation der eingehenden Daten gibt die Reihefolge ihrer Bearbeitung an. Es kann also nicht wahlfrei auf die Datensätze zugegriffen werden. Eine hohe Präzision mit der die Ergebnisse auf eine Anfrage geliefert werden ist bei den Strömen nicht vertreten. Das heißt, dass die Antworten angenähert werden müssen. Einige Gründe für diese Approximation sind die Echtzeitanforderung an die Systeme und die Unmöglichkeit der Vorhersage, welche der Daten später gebraucht werden.

Die hier vorgestellten Systeme und Anfragesprachen haben sowohl Vorteile als auch einige Mängel aufzuweisen. Außer der angenäherten Resultate können die Stromverarbeitungssysteme, wegen unvorhersagbarem Überfluss an Daten überlastet und dadurch überfordert werden.

Das Ziel der Forscher und Entwickler in diesem Gebiet ist es, einfache, schnelle, skalierbare und effiziente Werkzeuge und Anwendungen für die Strombearbeitung zu entwickeln. Sie sollen speicherplatzsparend und erweiterbar sein. Es wurde schon eine Menge von Systemen zu Verfügung gestellt, nichtsdestotrotz ist dieses Forschungsgebiet noch bei Weitem nicht erschöpft. Andere, verbesserte Versionen und Projekte werden noch in Zukunft, im Zusammenhang mit der Verarbeitung von XML-Datenströmen, parallel mit der Technikentwicklung, erbaut und konstruiert.

Literaturverzeichnis

[ATG+03] Avila-Campillo, I.; Todd J. Green.; Gupta A.; Onizuka M.: XMLTK: An XML Toolkit for Scalable XML Stream Processing. Universität Washington 2003 http://xmltk.sourceforge.net/

[BCD+04] Bry, F.; Coskun, F.; Durmaz, S.; Furche, T.; Olteanu, D.; Spannagel, M: The XML Stream Query Processor SPEX Institute for Informatics, University of Munich, Germany 2004 rewerse.net/publications/ download/REWERSE-RP-2005-15.pdf

[Beck04] Becker, O. :Dissertation: Serielle Transformationen von XML Probleme, Methoden, Lösungen. Universität Humboldt Berlin 2004. http://edoc.hu-berlin.de/dissertationen/ becker-oliver-2004-11-26/PDF/Becker.pdf

[Eise03] Eiseler, T.: Hauptseminar: Verarbeitung von Datenströmen. Technische Universität München WS 2002/03 http://wwwbayer.in.tum.de/lehre/WS2002/HSEM-bayer/

[FHK+03] Florescu, D.; Hillery, C.; Kossmann, D.; Lucas, P.; Riccardi, F.; Westmann, T.; Carey, M.; Sundararajan, A.: The BEA Streaming XQuery Processor. BEA Systems San Jose, CA, USA 2003 www.dbis.ethz.ch/research/publications/vldbj.pdf

[Furc03] Furche T.: Diplomarbeit Optimizing Multiple Queries against XML Streams Institut für Informatik Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen Oettingenstrasse 67, D-80538 München 2003 www.pms.ifi.lmu.de/forschung/ spex/resources/mqspex-thesis.pdf

[IGG+03] Iliana Avila-Campillo; Greeny, T.J.; Gupta, A.; Onizukaz M.; Raven, D.; Suciu D.: XMLTK: An XML Toolkit for Scalable XML Stream Processing Universität Waschington 2003

[Jörg05] Jörg, T.: Seminar: Datenbanken und Informationssysteme. Januar 2005 wwwdvs.informatik.uni-kl.de/courses/seminar/WS0405/

[Lezi03] Lezius, W.: Auf den Weg gebracht. Die XML-Anfragesprache XQuery im Praxiseinsatz. Aus XML & Web Services Magazin Ausgabe 02.2003 http://www.xmlmagazin.de/itr/online_artikel/psecom,id,363,nodeid,69.html

[Kies02] Kiesling, T.: Towards a Streamed XPath Evaluation. Diplomarbeit Institut für Informatik Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen Oettingenstrasse 67, D-80538 München 2002 www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Tobias.Kiesling/diplomarbeit.pdf

[Koeh01] Koehler D.: Problemseminar Datenintegration XML – Query Sprachen. Universität Leipzig WS 00/01. lips.informatik.uni-leipzig.de/pub/2001-7

[Wiki05] Wikipedia die freie Inzeklopedie. Impressium 06.2005 http://de.wikipedia.org/wiki/XQuery