

Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplomarbeit Nr. 3390

XPath Processing Optimization for SWoM Optimizer

Xi Tu

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Phys. Dieter H. Roller
begonnen am:	01.09.2012
beendet am:	05.03.2013
CR-Klassifikation:	D.2.11, D.1.3, H.2.8

Danksagung

Ich möchte mich ganz herzlich bei Herrn Roller für die Unterstützung und seine wertvollen Vorschläge bedanken, die mich beim Schreiben dieser Arbeit erleuchtet haben.

Mein Dank gilt auch Herr Prof. Dr. Leymann, der mir die Gelegenheit gab, diese Arbeit anzufertigen.

Inhalt

Danksagung	2
1. Einleitung	5
2. Hintergrund	7
2.1 XML	7
2.2 XPath.....	8
2.3 DOM	11
2.4 SAX.....	15
2.5 Variable in BPEL.....	18
2.6 Java String Operation (Pattern und Matcher).....	19
2.6.1 Regulärer Ausdruck.....	19
2.6.2 Java Pattern und Matcher Class.....	22
2.7 XML Schema.....	23
2.8 XML Beans.....	25
3. Algorithmus	26
3.1 Fundamentale Idee	26
3.2 Transformierung eines XML-Dokuments	28
3.3Entwicklung von Regulären Ausdrücken für das Verfahren.....	29
3.3.1 Ausdruck für Knoten ohne Werte und Attribute	30
3.3.2 Ausdruck für Knoten ohne Werte aber mit Attributen	31
3.3.3 Ausdruck für Knoten mit Text-Werten aber ohne Attribute	31
3.3.4 Ausdruck für Knoten mit Text-Werten und Attributen	32
3.4 String-Operationen für Erhalt der endgültigen Inhalte	34
3.5 Das vollständige Verfahren.....	35
3.6 Das verkürzte Verfahren	38
4. Implementation	39
4.1 Hintergrund und Technologie der Implementation.....	39
4.1.1 Umgebung für den Test	39
4.1.2 Der Code für Analyse eines XML-Dokuments mit DOM+XPath.....	40
4.2 Implementation.....	44
4.2.1 Implementation für Knoten ohne Werte und Attribute.....	44
4.2.2 Implementation für Knoten mit Attributen aber ohne Werte.....	46

4.2.3 Implementation für Knoten mit Wert aber ohne Attribute	48
4.2.4 Implementation für Knoten mit Werten und Attributen.....	50
4.2.6 Implementation für Suche mit Vergleichszeichen.....	52
5. Zusammenfassung und Ausblick.....	54
Anhang A cd.xml [W3C-XML].....	55
Anhang B simple.xml [XMLSpy]	63
Anhang C books.xml [W3C-XML]	65
Literaturverzeichnis	67
Erklärung.....	69

1. Einleitung

Web services, die auf Service-orientiertem Architektur-Framework basieren, dienen als Grundlage für modern verteilte, heterogene Anwendungen. Sie sind perfekt für die Schichtfunktion des Zwei-Ebene-Programmiermodells, das charakteristisch für Workflow-basierte Anwendungen ist.

Workflow-basierte Anwendungen [LR00] setzen sich aus zwei verschiedenen Teilen zusammen: Einem Prozessmodell, das die Reihenfolge beschreibt in der die verschiedenen Aktivitäten, welche das Prozessmodell vorgibt, ausgeführt werden (programming in the large), und den einzelnen Komponenten, die die verschiedenen Aktivitäten implementieren (programming in the small). In der Umgebung von Web-Diensten werden Prozessmodelle unter Verwendung des Web Services Business Process Execution Language (WS-BPEL) beschrieben.

Das Ziel eines Workflow-Management-Systems (WFMS), in das die WS-BPEL-Spezifikation implementiert wird, ist es den gesamten Lebenszyklus von Geschäftsprozessen zu verwalten, durch die damit verbundenen Prozessmodelle zu navigieren und die entsprechenden Web-Dienste aufzurufen. Die Stuttgarter Workflowmaschine (SWoM) implementiert teilweise die entsprechenden WS-BPEL-Standard.

SWoM realisiert einen Flow-Optimierer, der für ein Prozess-Modell einen Flow Ausführungsplan erzeugt, welchen der Navigator für eine optimale Verarbeitung der Instanzen von Modell nutzen kann. Einer der Optimierungstechniken ist die Optimierung von XPath-Abfragen, die in Übergangsbedingungen und Verteilung der Aktivitäten genutzt werden.

Das Ziel dieser Diplomarbeit ist es, eine Reihe von Optimierungs-Algorithmen für die XPath-Verarbeitung zu entwerfen/zu gestalten/ zu implementieren, die die Standard DOM-basierte Verarbeitung mit einem effizienteren, String-basierten Ansatz ersetzen. Es umfasst insbesondere die folgenden Aktivitäten:

- Gestaltung/Implementierung einer Reihe von Optimierungsverfahren
- Ausführung der entsprechenden Performancetests

Die populären und klassischen XML Datei Abfragetechniken sind DOM(Document Object Model) + XPath sowie SAX.

Das Bild 1.1 zeigt einen Überblick über das Verfahren, mit dem die DOM+XPath Abfrage durch String Operationen ersetzt werden kann. Durch das Ersetzen kann der Aufwand der Zeit und des Arbeitsspeichers bei der Abfrage verringert werden. Das Verfahren liest XML-Dokument und XPath Ausdruck in den Arbeitsspeicher zuerst ein, transformiert das XML-Dokument nach String, verwendet dann eine grundsätzliche String-Operation um das Ergebnis herauszufinden.



Bild 1.1 Verfahren der Ersetzungsoperation

Der Kern der String-Operation ist der Abgleich. Der Abgleich basiert auf einer String-Operation, die Operation wie der XPath-Ausdruck „//“ funktioniert. Durch die einmalige oder mehrfache Verwendung der String-Operation kann eine XPath-Expression simuliert und die DOM+XPath ersetzt werden. Damit erhält man eine effiziente XML-Daten Suchmechanik.

2. Hintergrund

Dieser Kapitel leitet in die gebräuchlichen assoziativen Grundlagen und Technologien ein, die die Grundlage bilden, um den Inhalt dieser Diplomarbeit zu verstehen.

Dieses Kapitel wird die folgenden Themen behandeln:

- 2.1 XML
- 2.2 XPath
- 2.3 DOM
- 2.4 SAX
- 2.5 Variable in BPEL
- 2.6 Java String Operation (Pattern und Matcher)
- 2.7 XML Schema
- 2.8 XMLBeans

2.1 XML

XML (Engl. Extensible Markup Language) ist eine vom World Wide Web Consortium (W3C) akzeptierte Auszeichnungssprache. Das XML Model wurde im Jahr 1998 vom W3C aufgenommen.

XML wird entwickelt, um hierarchisch strukturierte Daten in Textform zu speichern und zu transportieren, sowie um zu erklären, was die Daten für eine Bedeutung haben [W3C-XML]. Ein XML Dokument besteht aus Textzeichen und wird als Baum dargestellt und betrachtet. Der Einsatz von XML hängt nicht von der Plattform und Implementation ab. Durch die Hilfe von XML können Daten zwischen verschiedenen Computersystemen und auch im Internet leichter ausgetauscht werden.

Das untere XML Beispiel zeigt: das XML definiert die Struktur und speichert die Informationen von Jobs, dem Mitbegründer von Apple.

```
<field>
  <name>Jobs</name>
  <Company>Apple</Company>
</field>
```

Liste 2.1 XML Fragment

Ein XML-Dokument fängt mit einer XML-Deklaration an, um XML-Version, Zeichenkodierung und Verarbeitbarkeit ohne Dokumenttypdefinition zu spezifizieren. Folgend kann eine Dokumenttypdefinition optional verwendet werden, danach kommen

Elemente vom XML-Dokument zum Einsatz. Die Elemente eines XML-Dokuments werden als Baumstruktur organisiert. Als Beispiel zeigt die Liste 2.2 ein XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
  <CD category="COOKING">
    <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1989</YEAR>
  </CD>
```

Liste 2.2 XML-Dokument

Die erste Zeile des oben genannten XML-Dokuments ist eine XML-Deklaration. Sie zeigt, dass das XML-Dokument die XML Version 1.0 ist und die UTF-8 Zeichenkodierung verwendet.

2.2 XPath

XPath (Engl. XML Path Language) ist eine Abfragesprache, um Teile eines XML-Dokuments zu adressieren [W3C-XPath]. XPath ist vom World Wide Web Consortium (W3C) entwickelt worden.

XPath liegt auf einem Baummodell von XML-Daten. Es gibt folgende sieben Knoten Typen [W3C-XPath]:

1. Wurzel Knoten
2. Element Knoten
3. Attribute Knoten
4. Nameräume Knoten
5. Verarbeitungsanweisungen Knoten
6. Kommentare Knoten
7. Textknoten

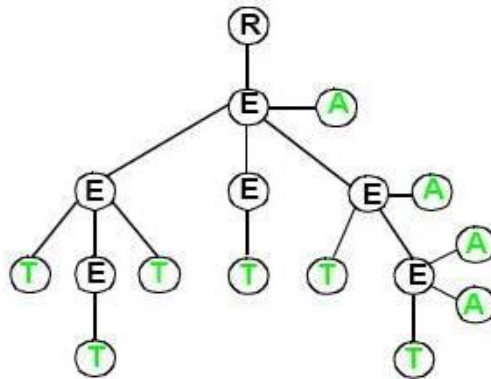


Bild 2.1 Baummodell von XML-Daten [AIM]

Ein XPath-Ausdruck ist ein Ausdruck, der eine Schrittfolge von einem Knoten zu einem anderen Knoten bedeutet. Die Schritte werden von „/“ getrennt. Jeder Schritt besteht aus drei Teilen:

- Achse (axis)
- Knotentest (node-test)
- null oder mehrere Prädikaten

Der unten gezeigte XPath-Ausdruck bedeutet, Cooking muss das Kind des Knotens Book sein, Book muss das Kind des Knotens Catalog sein, wobei der Knoten Catalog die Wurzel ist.

/child::Catalog/child::Book/child::Cooking

Liste 2.3 Original XPath-Ausdruck

Bei jedem Prozessschritt vom XPath-Ausdruck werden Achsen benutzt, um die Einschränkung für jede Suche klar zu definieren. Eine Achse besteht aus einer Menge von Knoten, die in Beziehung mit dem Zielknoten stehen. Durch Angabe von Achsen wird vom aktuellen Knoten im XML-Dokument navigiert. Dem Knotentest folgt :: Zeichen nach. X::Y bedeutet, dass man durch die Richtlinie X Achse in Y wählt. Beispielsweise das oben genannte child :: Catalog bedeutet die Achse Child, dass das Ergebnis Kinder vom getesten Knoten ist. Der Knoten ist Catalog.

Für gewöhnlich wird ein vereinfachter XPath-Ausdruck verwendet, weil ein XPath Ausdruck mit Achsen zu kompliziert ist. Deshalb wird generell ein vereinfachter XPath-Ausdruck benutzt.

Der unten gezeigte XPath-Ausdruck ist der vereinfachte XPath-Ausdruck aus Liste 2.3. Die beiden Ausdrücke sind semantisch äquivalent.

Liste 2.4 Vereinfachte XPath-Ausdruck

Die untere Liste zeigt alle Achsen von XPath sowie seine Abkürzungen sowie die Bedeutung der Achsen.

Achse	Adressierte Knoten	Abkürzung
ancestor	übergeordnete Knoten	
ancestor-or-self	übergeordnete Knoten inklusive des Kontextknotens	
attribute	Attributknoten	@
child	direkt untergeordnete Knoten	Einfach weg
descendant	untergeordnete Knoten	
descendant-or-self	untergeordnete Knoten inklusive des Kontextknotens	
following	im XML-Dokument nachfolgend (ohne untergeordnete Knoten)	
following-sibling	wie following, aber zugleich vom selben parent stammend	
parent	Direkt übergeordnete Elternknoten	../
preceding	im XML-Dokument vorangehend (ohne übergeordnete Knoten)	
preceding-sibling	wie preceding, aber zugleich vom selben parent stammend	
namespace	Namensraumknoten, die aus dem Attribut xmlns stammen	
self	der Kontextknoten selbst (nützlich für zusätzliche Bedingungen)	.

Liste 2.5 Achsen vom XPath-Ausdruck [Wiki]

2.3 DOM

DOM (Engl. Document Object Model) ist vom World Wide Web Consortium (W3C) als eine Spezifikation einer Schnittstelle definiert, die verwendet wird, um auf den Inhalt von XML einfach zugreifen zu können. Durch die Richtlinie vom W3C ist DOM eine Schnittstelle für Plattformen, Programmierungssprache und Internetbrowser, damit die anderen Standard-Komponenten von Webseiten manipuliert werden können [W3C-DOM]. DOM ist Plattform- und Sprachenunabhängig.

DOM ist eine Menge von Knoten, die hierarchisch strukturiert organisiert werden. Diese hierarchische Struktur wird als Baummodell dargestellt. Der Entwickler darf in der Struktur navigieren, um Informationen zu suchen.

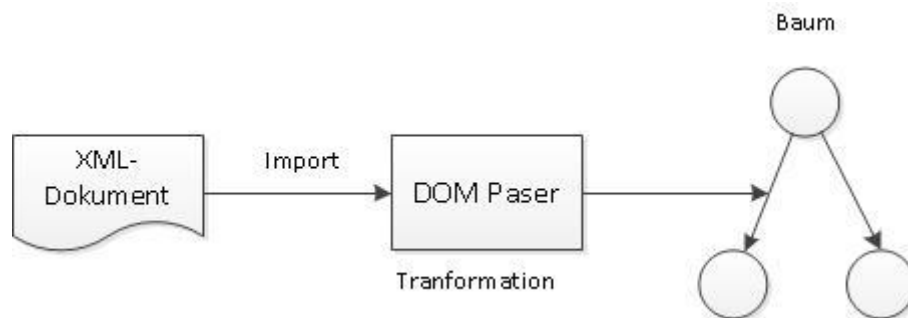


Bild 2.2 DOM Analysierung

Das Bild 2.2 zeigt das Arbeitsprinzip von DOM.

Folgendes XML-Fragment definiert einen Katalog mit dem Element Catalog und verschiedenen Firmen.

```
<Catalog>
  <Company>
    <NAME>Apple</NAME>
    <CEO>Jobs</CEO>
  </Company>
  <Company>
    <NAME>Microsoft</NAME>
    <CEO>Gates</CEO>
  </Company>
</Catalog>
```

Liste 2.6 XML Fragment

DOM ist in drei Teile aufgeteilt. Die drei Kerne sind DOM, HTML DOM und XML DOM. In der Ausarbeitung sprechen wir hauptsächlich über XML DOM. XML DOM definiert die Objekte und Attribute aller XML-Elemente sowie die Methoden (Schnittstellen) [W3C-DOM], die verwendet werden, um die XML-Elemente zu manipulieren bzw. die XML Elemente hinzuzufügen, zu erfassen, zu ändern oder zu löschen.

Durch folgende Richtlinien liest DOM die XML-Dokumente in Arbeitsspeicher eines Computers ein. Die Richtlinien sind [W3C-DOM]:

- Ein XML-Dokument ist ein Dokumentknoten
- Jedes XML-Kennzeichen ist ein Elementknoten
- Der Text eines Elementknotens, ist ein Textknoten
- Jedes XML-Attribut ist ein Attributknoten
- Annotation ist Annotationsknoten

Beispielsweise im in Liste 2.6 genannten XML-Dokument ist der Knoten <Catalog> die Wurzel und alle anderen Knoten des XML-Dokuments sind vom <Catalog> enthalten. Der Knoten <Catalog> hat zwei <Company> Knoten und jeder <Company> Knoten hat zwei Knoten. Diese sind <NAME> und <CEO>. Jeder Knoten hat einen Textknoten, wie „Apple“, „Jobs“.

Das Bild 2.3 zeigt die hierarchische Organisation vom XML-Dokument, welches von der Liste 2.6 präsentiert wird. Wir müssen aufpassen, dass ein Elementknoten keinen Text enthält. Der Text eines Elementknotens wird von Textknoten enthalten. Das bedeutet, der Elementknoten <NAME>Apple</NAME> zeigt, dass der Elementknoten einen Textknoten hat. Dieser Textknoten hat den Wert „Apple“.

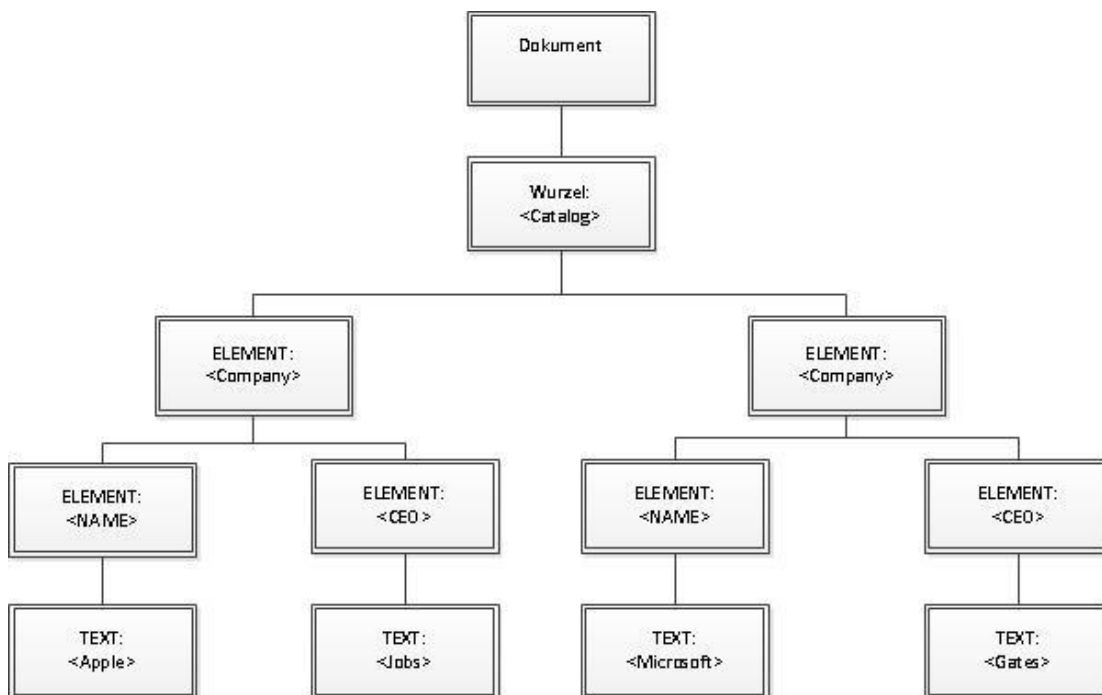


Bild 2.3 Ein hierarchisches Diagramm der Liste 2.6

DOM organisiert Daten als ein Baum-Modell. Soll ein XML-Dokument mit Hilfe von DOM geladen oder analysiert werden, muss das XML-Dokument zuerst das ganze Dokument mit seiner hierarchischen Struktur als ein Baum in den Arbeitsspeicher eines Rechners eingelesen werden. Danach kann damit angefangen werden, das XML-Dokument zu bearbeiten. Aus diesem Grund hat DOM folgende Vorteile:

- XML-Dokument kann in Arbeitsspeichern durch Hilfe von DOM lange Zeit und stabil vorliegen oder modifiziert werden, weil das Dokument von DOM als ein Baum-Modell organisiert wird. Dadurch können normale Anwendungsprogramme Daten und Struktur des XML-Dokuments leicht modifizieren
- Mit Hilfe von DOM kann jederzeit im XML-Dokument navigiert werden.
- Die Benutzung von DOM ist für Entwickler freundlich. DOM liefert API, durch diese den Entwicklern ermöglicht wird, im Baum Knoten leicht umzuwandeln, zu verschieben, zu löschen und einzufügen
- DOM kann leicht mit XPath kombiniert werden, um Informationen im XML-Dokument zu suchen

Das untere Code-Fragment zeigt, wie mit Hilfe von DOM für ein XML-Dokument ein Baum-Modell erstellt wird, und mit Hilfe von XPath-Expression „//Company“ in importiertem XML-Dokument alle Knoten durchgesucht werden, die am Anfang mit <Company> ausgezeichnet sind:

```
import java.io.InputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class DomParse {

    //Analyisierung durch DOM+XPath.

    void eval(InputStream in, String XP, Collection<String> result) throws Exception{

        //ein DocumentBuilderFactory-Instanz zu erzeugen.

        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();

        docFactory.setNamespaceAware(true);

        //ein Parser durch Hilfe von DocumentBuilderFactory zu erzeugen

        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        //eine Inputstream nach Document Object zu transformieren.

        Document doc = docBuilder.parse(in);
```

```

//ein XPath-Instanz zu erzeugen.
XPath xpath = XPathFactory.newInstance().newXPath();

//kompilier die importierte XPath-Expression nach eine XPathExpression Instanz.
XPathExpression expr = xpath.compile(XP);

//Das Suchergebnis wird in eine NodeList-Object eingespeichert.
NodeList nodeList = (NodeList) expr.evaluate(doc, XPathConstants.NODESET);

//Durch eine For-Schleif wird alle Suchergebnisse in List-result gespeichert.
for (int i = 0; i < nodeList.getLength(); ++i) {
    Node node = nodeList.item(i);
    result.add(node.getNodeValue());
}
}

```

Liste 2.6 DOM Anwendung

Durch den in Liste 2.6 gezeigten Code kann man wissen, welche Behandlungsschritte von DOM+XPath sind:

- erzeuge eine DocumentBuilderFactory-Instanz
- erzeuge eine DocumentBuilder-Instanz durch die Methode docFactory.newDocumentBuilder()
- importiere XML-Dokument und transformiere nach Dokument-Object
- erzeuge eine XPath-Instanz
- Kompiliere den importierten XPath-Ausdruck durch die Methode compile(XPath-Ausdruck) von XPath-Instanz
- erhalte das Suchergebnis durch evaluate(doc, XPathConstants.NODESET) von XPath-Ausdruck Instanz

Liste 2.7 Behandlungsschritte von DOM+XPath

Im Arbeitsspeicher wird ein Baum-Modell eingerichtet. Davon kann DOM viele Vorteile wie z.B leichte Navigierung sowie Programmierung usw. bekommen. Aber es gibt auch Nachteile mit der Einrichtung des Baum-Modells im Arbeitsspeicher. Wenn ein XML-Dokument zu groß ist, dann kann das Verfahren von Laden und Analysieren lange dauern und hohe Anforderungen an die Ressourcen vom Rechner stellen, weil eine Baum-Struktur viele Plätze im Arbeitsspeicher benötigt und in Arbeitsspeichern stabil bleibt.

In der vorliegenden Ausarbeitung wird über die Findung einer neuen Methode diskutiert. Die Methode soll gefunden werden, um bei der Suche in XML-Dokumenten als DOM+XPath

weniger Rechenzeitaufwand sowie weniger Anforderungen an die Ressourcen vom Rechner zu stellen.

2.4 SAX

JAXP (Engl. Java API for XML Processing) liefert zwei verschiedene Mechanismen für die Behandlung eines XML-Dokuments. Eine ist die im letzten Kapitel genannte DOM, die andere ist SAX (Engl. Simple API for XML). Im Vergleich zu DOM besteht das Arbeitsprinzip von SAX nicht durch Erzeugung eines Baum-Modells im Arbeitsspeicher, sondern durch das sequentielle Scannen eines XML-Dokuments.

SAX bietet ein Modell, um XML-Dokumente schnell zu analysieren. SAX wird benutzt, um ein XML-Dokument zu analysieren. SAX-Parser wird zwischen Scannen viele Ereignisse treffen, z.B. den Start und Ende eines Dokuments oder eines Elements, dann wird der geeignete Ereignis-Handler informiert. Der Handler behandelt jedes Ereignis durch vorher definiertes Verhalten. Anschliessend scannt SAX wie vorher weiter bis zum Ende des Dokuments. Im Allgemeinen entstehen folgende Ereignis-Typen bei Realisierung eines SAX-Parsers:

- Vor und nach der Analysierung jedes Elements im XML-Dokument werden die Elements-Ereignisse ausgelöst
- Bei dem Start und Ende von Text werden die Charakter-Ereignisse ausgelöst
- Bei der Behandlung vom Dokument DTD und Schema entsteht das DTD- oder Schema-Ereignis
- Error-Ereignis wird benutzt, das Anwendungsprogramm zu informieren

Das Bild 2.3 zeigt das Arbeitsprinzip von SAX:

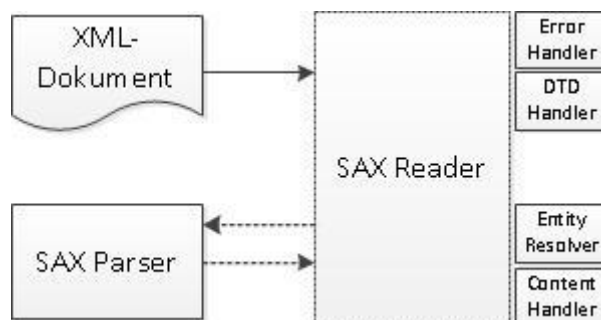


Bild 2.4 Das Arbeitsprinzip von SAX

Es gibt bei SAX vier oft verwendete Handler. Die sind ContentHandler, DTDHandler, ErrorHandler und EntityResolver.

ContentHandler bleibt in org.xml.sax von Java Class Package. ContentHandler ist der wichtigste Handler von SAX. Bei der Analyse eines XML-Dokuments werden viele Ereignisse wie z.B. der Start und Ende des XML-Dokuments, Start und Ende des Elements usw. auftauchen. Wenn solche Ereignisse entstehen, dann ruft SAX-Parser die geeignete

Methode von ContentHandler ab, um die Ereignisse zu behandeln. ContentHandler hat folgende Methoden:

```
void startDocument()  
void characters(char[ ] ch, int start, int length)  
void endDocument()  
void startElement(String uri, String localName, String qName, Attributes atts)  
void endElement(String uri, String localName, String qName)
```

Liste 2.8 Methode von ContentHandler

DTDHandler wird benutzt, um die Mitteilung von DTD-Ereignissen anzunehmen. SAX-Parser muss nach startDocument() und vor erstem startElement() alle DTD-Ereignisse melden. DTDHandler hat folgende zwei Methoden:

```
void notationDecl(String name, String publicId, String systemId)  
  
void unparsedEntityDecl(String name, String publicId, String systemId, String  
notationName)
```

Liste 2.9 Methode von DTDHandler

ErrorHandler ist eine SAX-Schnittstelle, um Fehler zu behandeln. Wenn SAX-Anwendungsprogramm eigene Behandlung für Fehler haben möchte, dann muss das Programm die Schnittstelle verwirklichen, durch die der Parser alle Fehler und Alarme melden kann. Liste 2.10 zeigt zwei Methoden von ErrorHandler:

```
void error(SAXParseException exception)  
void fatalError(SAXParseException exception)
```

Liste 2.10 Methoden von ErrorHandler

EntityResolver wird benutzt, um Entität zu analysieren und hat nur eine Methode wie Liste 2.11 zeigt:

```
public InputSource resolveEntity(String publicId, String systemId)
```

Liste 2.11 Methode von EntityResolver

Für folgendes XML-Dokument:

```
<Doc>  
  
  <field>Hallo</field>  
  
</Doc>
```

Liste 2.12 ein XML Beispiel

Bei der Analysierung der oben genannten XML-Dokumente werden durch SAX folgende Ereignisse erzeugt:

- start Dokument
- start Element: Doc
- start Element: field
- characters: Hallo
- end Element: field
- end Element: Doc
- end Dokument

Liste 2.13 Ereignisse bei Analysierung der Liste 2.7 durch SAX

Bei einem vollständigen Verfahren von SAX-Behandlung sollten folgende Schritte enthalten sein:

- erzeuge Programm für Ereignisbehandlung
- erzeuge SAX-Parser
- analysiere Dokument und schick ereignis nach Programm;
- verteile Programm zu Parser

Liste 2.14 Das Verfahren von SAX-Behandlung

Die XML-Dokument Analysierung durch SAX, ergibt folgende Schritte:

- erstelle eine SAX Analysierungsklass, das aus DefaultHandler vererbt
- rewrite Methode des Class
- erzeuge eine Factory-Instanz (SAXParserFactory)
- erzeuge SAX-Parser durch Methode von SAXParserFactory
- erzeuge XML-Dokumentenfluss
- erzeuge Instanz des Analysierungsklass;
- verwende Methode für Parse

Liste 2.15 Analysierungsschritte von SAX

Im Vergleich zu DOM hat SAX die folgenden Vorteile:

- Analysierungsgeschwindigkeit ist schneller als DOM
- ContentHandler kann mehrfach sein, das heißt, dass der Analysierungsprozess parallel implementiert werden kann
- SAX braucht im Analysierungsprozess weniger Arbeitsspeicher als DOM, weil bei der Analysierung vom SAX Dokument nicht einmalig total in Arbeitsspeicher geladen wird, sondern eine Seite den Inhalt vom Dokument einzulesen, gleichzeitig zu analysieren

Liste 2.16 Vorteile von SAX im Vergleich zu DOM

Im Vergleich zu DOM hat SAX die folgenden Nachteile:

- Bei Analysierung von SAX muss das Programm für Ereignisbehandlung verwirklicht werden
- SAX-Analysierung darf XML-Dokument nicht korrigieren
- SAX-Analysierung kann nur Daten sequentiell besuchen

Liste 2.17 Nachteile von SAX im Vergleich zu DOM

2.5 Variable in BPEL

Variable wird in BPEL verwendet, um Business Status zu behalten. Jede Variable hat einen eigenen Bereich, bei dem die Variable manipuliert werden kann. Es gibt in BPEL zwei Methoden, die Variable erzeugen können. Die beiden Methoden sind in Liste 2.18 beschrieben.

- Durch Definition von xsd
- Durch Umtausch von WSDL Message

Liste 2.18 Methoden für die Definition der Variablen in BPEL

Durch xsd können einfache und komplexe Variablen erzeugt werden. Liste 2.19 zeigt ein Beispiel für Definition einer einfachen Variablen.

- `<variable name=„Name“ type=„xsd:string“/>`

Liste 2.19 Definition der Variablen durch XML Einfache Type

Liste 2.20 beschreibt die Definition durch XML Komplex Variablen. Der erste Teil der Liste ist die Definition der komplexe Variable SuchResultes. Der zweite Teil ist die Instanz der Definition.

```
<xsd:complexType name=„SuchResults“>
  <xsd:sequence>
    <xsd:element name=„SuchResult“
      maxOccurs=„unbounded“ form=„qualified“>
      <xsd:complexType>
        <xsd:attribute name=„SuchID“ type=„int“/>
        <xsd:attribute name=„Result“ type=„xsd:String“/>
      </xsd:complextype>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<variable name=„Results“ type=„e:SuchResults“/>
```

Liste 2.20 Definition der Variablen durch XML Komplex Type

Liste 2.21 beschreibt die Definition durch WSDL Message. Der erste Teil der Liste ist die Definition der Variable SuchInfo. Der zweite Teil ist die Instanz der Definition.

```

<message name=„SuchInfo“ >
  <part name=„Info1“ element=„e:InfoContainer“ />
  <part name=„Info2“ element=„e:InfoContainer“ />
</message>

<variable name=„InfoVariable“ messageType=„e:SuchInfo“ / >

```

Liste 2.21 Definition der Variablen durch WSDL Message

2.6 Java String Operation (Pattern und Matcher)

Java String Operation enthält umfangreiche Inhalte. Hier werde ich nur auf die Inhalte eingehen, die wir in dieser Diplomarbeit benötigen. Das bedeutet, dass ich im folgenden Abschnitt nur über Regulären Ausdruck, sowie Java Pattern und Matcher Class schreiben werde. Im Abschnitt diskutieren wir die Anpassung aus einer Zeichenkette.

Ein normales Verfahren für die Anpassung zeigt das untere Bild 2.5.

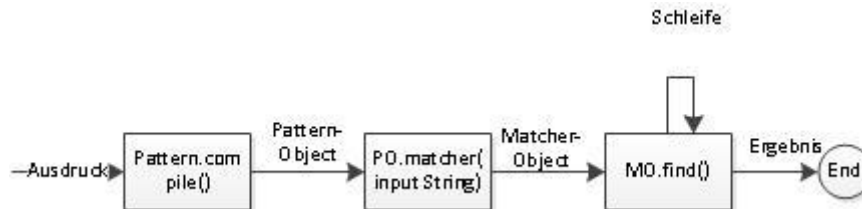


Bild 2.5 Verfahren für Anpassung aus Zeichenkette

Man kann dem Bild 2.5 entnehmen, dass das Verfahren zuerst den Regulären Ausdruck einliest und die Methode `Pattern.compile()` verwendet, um durch den Ausdruck ein Pattern zu erzeugen. Beim zweiten Schritt benutzt die Methode `PO.matcher()`, um das Matcher-Object zu erhalten. Danach wird durch die Methode `find()` von Matcher-Object die gewünschte Zeichenkette aus importierter Zeichenkette erhalten.

2.6.1 Regulärer Ausdruck

Regulärer Ausdruck wird benutzt, um String-Pattern zu bestimmen. Wenn man irgendwann eine Zeichenkette, die an ein Pattern anpasst, positionieren möchte. Dann kann Regulärer Ausdruck verwendet werden [Core Java 2]. Ich werde durch die folgenden Beispiele die Grundlage des Ausdrucks erklären, um das gewünschte Ergebnis zu erhalten.

Zuerst fange ich mit einem einfachen Beispiel an. Wir suchen eine Zeichenkette, die „book“ enthält. Dies wird in der Liste 2.22 gezeigt:

- Regulärer Ausdruck: book
- Geeignete Zeichenkette: bookstore, handbook, newbooks

Liste 2.22 Einfaches Beispiel für Regulären Ausdruck

Die Liste 2.23 zeigt die Benutzung der Notation „|“. Das Symbol bedeutet „oder“.

- Regulärer Ausdruck: d(am|u|umm)e
- Geeignete Zeichenkette: dame, due, dumme

Liste 2.23 Beispiel für Zeichen „|“

Wie Liste 2.23 zeigt, alle Strings sind angepasst, dass seine erstes Charakter „d“, letztes Charakter „e“ sind, und zwischen Charakteren „d“ und „e“ kann aus „am“, „u“ sowie „umm“ irgendeine sein.

Die Liste 2.24 zeigt die Verwendung der Notation „[]“. Durch Hilfe der Notation können alle Wörter angepasst werden, die ersten Charakter „d“, letzten Charater „x“ und dazwischen die Symbole haben, die in der Notation enthalten werden.

- Regulärer Ausdruck: d[au+]x
- Geeignete Zeichenkette: dax, dux, d+x

Liste 2.24 Beispiel für das Zeichen „[]“

Die Liste 2.25 zeigt die Benutzung der Notation „.“. Durch Verwendung der Notation kann mit irgendeinem Kennzeichen mit „.“ ausgetauscht werden.

- Regulärer Ausdruck: d.x
- Geeignete Zeichenkette: dax, dux, d+x, ddx... usw.

Liste 2.25 Beispiel für Zeichen „.“

Die Liste 2.26 zeigt die Benutzung der Notation „-“. Durch Hilfe der Notation darf ein Umfang bestimmt werden.

Regulärer Ausdruck	Bedeutung
[0-9]	Irgendeine Ziffer aus 0-9
[A-Z]	Irgendein Charakter aus A-Z

Liste 2.26 Beispiel für Zeichen „-“

Liste 2.27 zeigt eine Zeichengruppe, mit der die Häufigkeit der Symbole definiert werden kann.

Regulärer Ausdruck	Bedeutung
{n}	Häufigkeit ist n
{n,m}	Häufigkeit ist von n bis m
*	Häufigkeit kann 0 oder irgend wieviele sein
+	Häufigkeit kann 1 oder irgend viele sein
?	Häufigkeit ist entweder 1 mal oder keine

Liste 2.27 Zeichengruppe für die Häufigkeit [Core Java 2]

Liste 2.28 zeigt eine Definition für ein Autoschild:

- Regulärer Ausdruck: `[A-Z]{2}[0-9]{4}[A-Z]{2}` Das bedeutet: Erste zwei Kennzeichen müssen aus A-Z irgend zwei wählt, mitte vier Kennzeichen aus 0-9 irgend vier und letzte zwei Kennzeichen wieder aus A-Z irgend zwei wählt werden.
- Geeignete Zeichenkette: DC8888ST

Liste 2.28 Beispiel: Regulärer Ausdruck für ein Autoschild

Wenn man Charakter und Ziffer bei dem Beispiel, das in Liste 2.28 gezeigt wird, mit Zeichen „-“ verbinden möchte, dann soll man den Regulären Ausdruck wie in Liste 2.29 gezeigt, umformulieren.

- Regulärer Ausdruck: `[A-Z]{2}\-[0-9]{4}\-[A-Z]{2}`
- Geeignete Zeichenkette: DC-8888-ST

Liste 2.29 Beispiel: Regulärer Ausdruck für Autoschild mit Zeichen „-“

Werden die Zeichen „-“ nicht unbedingt gebraucht, dann gibt es die Möglichkeit hinter das Zeichen „-“ einfach ein „?“ zu legen. z.B. `[A-Z]{2}\-?[0-9]{4}\-?[A-Z]{2}`, dann sind die beiden in Liste 2.28 und 2.29 gezeigten Zeichenketten richtig.

Zeichen „^“ bedeutet negativ, wenn das Zeichen mit anderen Zeichen von „[]“ enthält, wie die untere Liste 2.30 zeigt:

- Regulärer Ausdruck: `[^D][A-Z]{2}` Das bedeutet, dass die Zeichenkette irgendeine Zeichenkette sein darf, aber ihr erstes Zeichen nicht D sein darf.
- Geeignete Zeichenkette: AC-8888-ST

Liste 2.30 Beispiel: Regulärer Ausdruck mit Zeichen „^“

Die richtige Erzeugung von Regulärem Ausdruck ist die Grundlage, mit der wir das Ersetzungsverfahren durchführen können. Nachdem wir den Regulären Ausdruck korrekt erstellt haben, werden wir mit Hilfe des Regulären Ausdrucks ein Pattern herstellen.

2.6.2 Java Pattern und Matcher Class

Durch Hilfe von Java Pattern und Matcher Class kann für einen Regulären Ausdruck ein Pattern konstruiert werden, um geeignete Zeichenkette zu finden. Java Pattern Class befindet sich in `java.util.regex.Pattern`. Java Matcher Class ist bei `java.util.regex.Matcher`. Liste 2.31 zeigt seine typische Benutzung [JavaTM 2 PlatformStandard Ed. 5.0]:

```
Pattern pattern = Pattern.compile("t*x");  
  
Matcher matcher = pattern.matcher(importiertes Zeichenfluss);  
  
boolean b = matcher.find();
```

Liste 2.31 Die Muster typischer Benutzung von Java Pattern und Matcher Class

Nachdem der Ausdruck von Methode `compile()` kompiliert worden ist, ist die Zeichenkette `t*x` der Reguläre Ausdruck und dieser kompilierte Ausdruck wird anschliessend in `Pattern-Object` gespeichert. Durch Verwendung der Methode `find()` kann die importierte Zeichenkette mit dem kompilierten Regulären Ausdruck „`t*x`“ verglichen werden, wenn es die geeignete Zeichenkette gibt, dann gibt die Methode ein „`true`“ zurück. Die Methode fängt ab Anfang der `Matcher-Object` an, wenn beim letzten Mal eine geeignete Zeichenkette gefunden wurde, dann fängt ab der erste neue Charakter an.

Bei Kompilation eines Patterns dürfen verschiedene Attribute verwendet werden. Die Benutzung ist wie in Liste 2.32 dargestellt. Die Attribute bedeuten, dass es bei der Anpassung keinen Unterschied zwischen Großschreibung und Kleinschreibung gibt.

```
Pattern pattern = Pattern.compile(„t*x“, Pattern.CASE_INSENSITIVE)
```

Liste 2.32 Attribute von `Pattern-Object`

Die Liste 2.33 zeigt alle sechs Attribute, die von `Pattern-Object` unterstützt werden:

- `CASE_INSENSITIVE`
- `UNICODE_CASE`
- `MULTILINE`
- `UNIX_LINES`
- `DoTALL`
- `CANON_EQ`

Liste 2.33 Sechs von `Pattern-Object` unterstützte Attribute

Nachdem ein Pattern erzeugt wird, kann ein `Matcher` durch die Methode von `Pattern-Object` wie `pattern.matcher(importierte Zeichenkette)` hergestellt werden. Durch den `Matcher-Object` kann die geeignete Zeichenkette aus importiertem Zeichenfluss mit Methoden herausgefunden werden. Die Methoden sind wie in Liste 2.34 dargestellt.

- `public boolean find()`
- `public int start()`
- `public int end()`

Liste 2.34 Wichtige Methoden von `Matcher`

Das Ziel dieser Arbeit ist nicht das ganze Dokument mit einem Regulären Ausdruck anzupassen, sondern aus einem Dokument eine oder mehrere Zeichenketten auszusuchen, die mit einem Regulären Ausdruck identisch sind. Die Methode find() bietet einen Boolean-Wert, ob die eine passende Zeichenkette gefunden worden ist. Wenn der Wert „Ja“ ist, dann versucht die Methode die nächste passende Zeichenkette in dem restlichen Dokument zu finden. Bei jedem Ergebnis von Methode find() kann man durch Verwendung von Methode start() den Anfang der passenden Zeichenkette positionieren, oder durch Verwendung von Methode end() das Ende der passenden Zeichenketter positionieren. Das Verfahren kann durch den in Liste 2.35 gezeigten Code realisiert werden.

```
while(matcher.find()){  
  
    int start = matcher.start();  
  
    int end = matcher.end();  
  
    String match = tempString.substring(start, end);  
  
    result.add(match);  
}
```

Liste 2.35 Implementation für den Erhalt des Anapassungsergebnisses

Bei der Liste 2.35 benutzt man eine while-schleife, der Wert von Methode find() ist als Bedingung. Die Methode substring() nimmt die Werte von start() und end() als Attribute, um die gewünschte Zeichenkette zu bekommen.

2.7 XML Schema

XML Schema ist Nachfolger der XML DTD. XML Schema beschreibt die Struktur eines XML Dokuments. XML Schema basiert auf XML. Die XML-Schema-Sprache wird auch als XML Schema Definition (XSD) bezeichnet [W3C-Schema]. Das heißt, dass XML Schema benutzt wird, um die Teile eines XML Dokuments zu definieren. Ein XML Schema:

- definiert Elemente, die in einem XML-Dokument vorkommen können
- definiert Attribute, die in einem XML-Dokument vorkommen können
- definiert, welche Elemente die untergeordneten Elemente
- legt die Reihenfolge der untergeordneten Elemente
- definiert die Anzahl der untergeordneten Elemente
- legt fest, ob ein Element leer ist oder Text enthalten
- definiert Datentypen für Elemente und Attribute
- definiert Standard- und feste Werte für Elemente und Attribute

Liste 2.36 Funktionen von XML Schema [W3C-Schema]

XML Schema definiert verschiedene Datentypen. Die können in zwei große Mengen geteilt werden. Eine ist einfacher Datentyp, die andere ist komplexer Datentyp. Die einfachen Datentypen werden in Liste 2.37 gezeigt und wurden vom W3C vordefiniert.

- xs: string
- xs: decimal
- xs: integer
- xs: boolean
- xs: date
- xs: time

Liste 2.37 Die von W3C vordefinierten einfachen Datentypen [W3C-Schema]

Die Liste 2.38 zeigt für die einfachen Datentypen ein paar Beispiele:

- <name>Bill Gates</name>
- <today>2013-02-01</today>
- <menge>100</menge>

Liste 2.38 Beispiele für die einfachen Datentypen von XML Schema

Die Syntax zur Definition eines einfachen Elementes ist <xs:element name="..." type="...">. Die Liste 2.39 zeigt die geeigneten Definitionen der Elemente, die die einfachen Datentypen besitzen:

- <xs:element name="name" type="xs:string"/>
- <xs:element name="today" type="xs:date"/>
- <xs:element name="menge" type="xs:integer"/>

Liste 2.39 Die geeignete Definition für die Elemente von Liste 2.38

Ein einfaches Element darf nur Text aber keine anderen Elemente oder Attribute enthalten. Die sogenannten Texte sind die in Liste 2.37 gezeigten Datentypen.

Ein Komplex Element besteht aus anderen Elementen und/oder Attributen. Es gibt vier Arten von komplexen Elementen. Diese sind wie in Liste 2.40 beschrieben.

- A: leeres Element
- B: Element, das andere Elemente enthält
- C: Element, das nur Text enthält aber mit Attributen
- D: Element, das nicht nur andere Elemente sondern auch Text enthält

Liste 2.40 Vier Arten von komplexen Elementen

Die Liste 2.41 zeigt vier verschiedene komplexe Elemente durch vier Beispiele.

- A: <variable id="1"/>
- B: <name>
 <vorname>Bill</vorname>
 <nachname>Gates</nachname>
 </name>
- C: <title lang="en">Harry Potter</title>
- D: <name>
 Die CEO der Microsoft war <name>Bill Gates</name>
 </name>

Liste 2.41 Beispiele für vier verschiedene Arten von komplexen Elementen

Als Beispiel zeigt die Liste 2.42, wie ein Komplex Element "name" definiert wird.

```

<xs:element name="name">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Liste 2.42 Definition des komplexen Elementes "name"

Die Ermittlung der oben genannten verschiedenen Elemente von XML Schema ist wichtig für diese Ausarbeitung.

2.8 XML Beans

XMLBeans ist eine Technologie, die benutzt wird, um XML Dokument mit Java Object zu verbinden. XMLBeans liefert eine direkte Methode, damit XML Dokument durch Java leicht behandelt werden kann [XMLBeans].

Nachdem XMLBeans auf Rechner installiert wird, kann ein XMLBeansprojekt angefangen werden. XML Schema ist der Startpunkt für jedes XMLBeansprojekt. Für die Analysierung eines Schemas wird bei der DOS-Umgebung nur das Eintippen des Ausdrucks scomp benötigt. Die genaue Erklärung über scomp Ausdruck kann man bei [XMLBeans] finden.

Bild 2.6 zeigt das Prinzip der Analysierung für XML Schema durch Hilfe von XMLBeans.



Bild 2.6 Analysierung eines XML Schemas durch XMLBeans

Nachdem ein XML Schema von XMLBeans analysiert wird, wird eine geeignete JAR Datei erstellt. Dann können XML Elemente wie Java Object behandelt werden.

Im Kapitel 2 wurden das für die Ausarbeitung erfordernde Wissen und Technologien beschrieben. Im nächsten Kapitel wird der Fokus auf die Beschreibung des Algorithmus gelegt.

3. Algorithmus

Das Kapitel thematisiert die Technik, die DOM+XPath ersetzen wird, um Informationen aus einem XML-Dokument auszusuchen. Ich werde versuchen, bei dem Verfahren pure String-Operationen zu benutzen, damit auf der einen Seite die DOM+XPath Technik optimal ersetzt werden kann und auf der anderen Seite die Nachteile insbesondere der Zeitbedarf und der Rechenressourcenbedarf der Hardware minimiert werden können.

Die folgenden Abschnitte werden die Details der String-Operationen erklären:

- Abschnitt 3.1 „Fundamentale Idee“ beschreibt die Einleitung in den Algorithmus
- Abschnitt 3.2 „Transformierung eines XML-Dokuments“ beschreibt das Verfahren, wie XML-Dokument nach String-Fluss umgewandelt wird
- Abschnitt 3.3 „Entwickeln des Regulären Ausdrucks für das Verfahren“ zeigt vier entwickelte Reguläre Ausdrücke für vier Situationen
- Abschnitt 3.4 „String-Operationen für Erhalt der endgültigen Inhalte“ zeigt, dass die normalen String-Operationen benutzt werden, um endgültige Inhalte zu bekommen

3.1 Fundamentale Idee

In diesem Abschnitt werden wir die fundamentale Idee einleiten.

Die fundamentale Idee von der Ersatztechnik ist wie in Bild 3.1 dargestellt. Das Verfahren liest zuerst XML-Dokument, XML-Schema und XPath-Ausdruck ein. Dazwischen wird XML-Dokument als String-Fluss eingelesen und als String-Object gespeichert. Parallel wird XPath-Ausdruck durch Hilfe von StringTokenizer in individuelle Einheiten eingeteilt. Gleichzeitig wird XML-Schema durch Hilfe von XML Beans analysiert, um die geeignete Struktur von XML-Dokument zu ermitteln. Nachdem die Struktur eines XML-Dokuments ermittelt wurde, kann das XML-Dokument durch einmaligen oder mehrmaligen Anwendung der anpassenden Implementierungen analysiert werden, die wie in Liste 3.19 dargestellt werden.

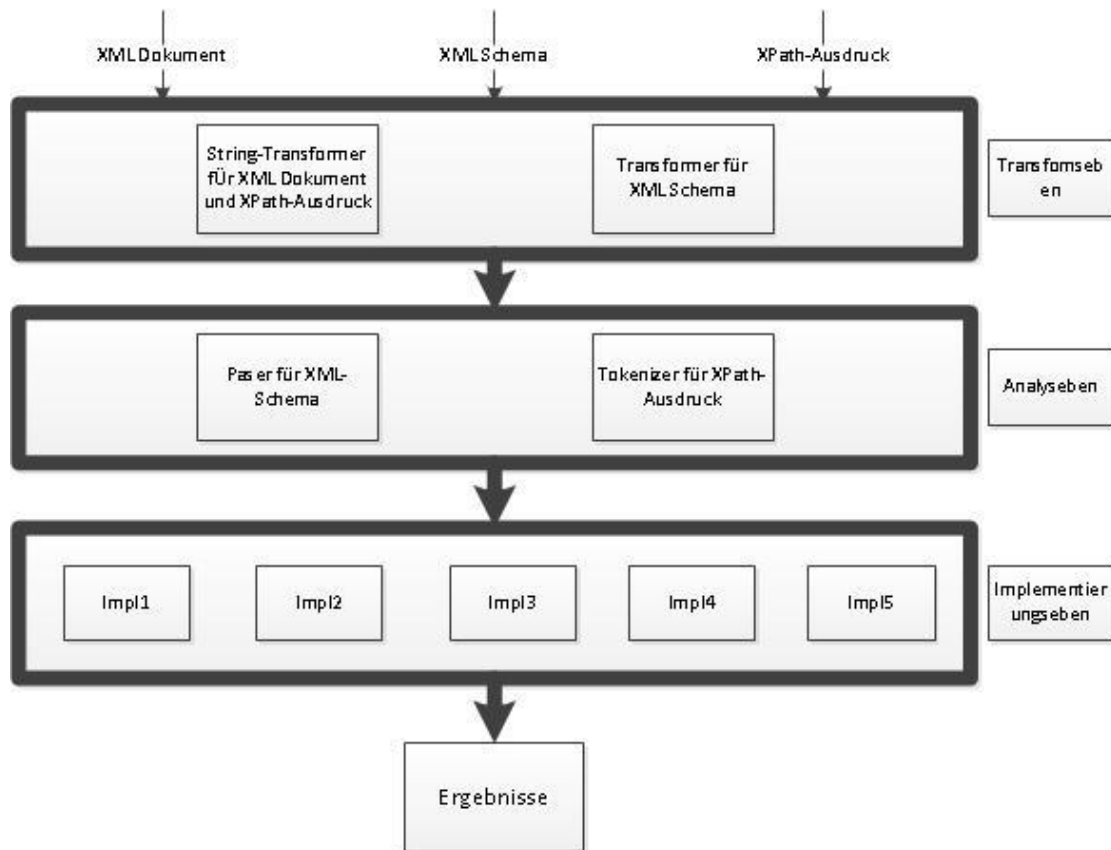


Bild 3.1 Drei Ebenen der pure String-Operation

Das Verfahren besteht aus drei Ebenen. Diese lauten Transformsebene, Analyseebene und Implementierungsebene. Die drei Ebenen werden durch das Bild 3.1 beschrieben.

Um diese Idee zu realisieren, muss der XPath-Ausdruck „//Suchobject“ nach geeignetem Regulären Ausdruck definiert werden. Die Regulären Ausdrücke, die für verschiedene Situationen passend sind, sind durch Liste 3.19 beschrieben. Wir nehmen die mit XPath-Ausdruck „//Suchobject“ passenden Regulären Ausdrücke als Grundlage für unsere Ersatztechnik. Durch den richtigen Ausdruck kann das gewünschte Ergebnis aus dem String-Fluss ausgefiltert werden und in eine ArrayList als String abgespeichert werden.

Das Verfahren sollte aus den vier in Liste 3.1 gezeigten Phasen bestehen.

- Umwandlung von XML-Dokument nach String-Fluss
- Auswahl des jeweilig geeigneten Regulären Ausdrucks für das entsprechende XML-Dokument
- Filterung und Speicherung des gewünschten Ergebnisses mit Hilfe von Pattern
- Verwendung der normalen String-Operationen, um die Kennzeichen vom XML-Dokument abzuschneiden, um die gewünschte Information zu erhalten

Liste 3.1 die Vierphasen von Ersatztechnik

3.2 Transformierung eines XML-Dokuments

In diesem Abschnitt spreche ich über die Transformierung vom XML-Dokuments zu einem String-Fluss.

Das Bild 3.1 zeigt den Vorgang.

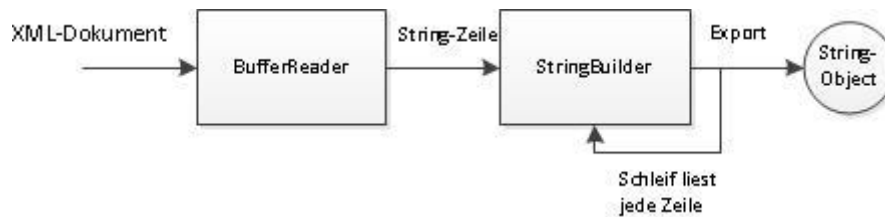


Bild 3.2 Vorgang der Umwandlung

Die in Liste 3.2 gezeigte Methode `readFile(String fileName)` wandelt das XML-Dokument zum String-Fluss um.

```
String readFile(String fileName) throws IOException {  
  
    File file = new File(fileName);  
  
    BufferedReader bf = new BufferedReader(new FileReader(file));  
  
    String content = "";  
    StringBuilder sb = new StringBuilder();  
  
    while(content != null){  
        content = bf.readLine();  
  
        if(content == null){  
            break;  
        }  
    }  
}
```

```

        sb.append(content.trim());
    }

    bf.close();

    return sb.toString();
}

```

Liste 3.2 Transformierung vom XML-Dokument zum String-Fluss

Die Methode importiert zuerst das XML-Dokument als File-Fluss. Im zweiten Schritt wird der File-Fluss von BufferedReader-Object Zeile für Zeile eingelesen. Im letzten Schritt werden die eingelesenen XML-Daten durch StringBuilder-Object nach String-Object transformiert.

3.3 Entwicklung von Regulären Ausdrücken für das Verfahren

In diesem Abschnitt werden wir für das Verfahren die geeigneten Ausdrücke entwickeln und durch die Ausdrücke die Patterns erhalten. Die wichtigste Arbeit in dem Verfahren ist es die geeigneten Regulären Ausdrücke zu entwickeln, weil das Verfahren nur mit den korrekten Patterns, die aus Regulären Ausdrücken erzeugt werden, das gewünschte Ergebnis erzielen kann. Die Behauptung ist selbstverständlich, denn ohne korrekte Patterns kann überhaupt kein richtiges Ergebnis erzielt werden.

Es ist bekannt, dass die Struktur eines XML-Dokuments vom Entwickler selbst definiert wird. Deshalb gibt es nicht nur eine einzige Struktur für jedes XML-Dokument. Das heißt, dass jedes XML-Dokument eine eigene Struktur hat. Beispielsweise enthalten Knoten in manchen Dokumenten Text-Knoten, in anderen Dokumenten jedoch keine. Deshalb habe ich für verschiedene Situationen verschiedene Reguläre Ausdrücke entwickelt, damit die Suche korrekt durchgeführt werden kann.

Liste 3.3 zeigt vier verschiedene Situationen für das Erstellen der Regulären Ausdrücke. Mit Hilfe der Ausdrücke kann die gewünschte Information aus dem XML-Dokument extrahiert werden.

- gesuchter Knoten hat keine Text-Werte und keine Attribute
- gesuchter Knoten hat keine Text-Werte, enthält aber Attribute
- gesuchter Knoten hat Text-Werte, aber keine Attribute
- gesuchter Knoten hat Text-Werte und Attribute

Liste 3.3 Vier verschiedene Situationen für das Erstellen der regulären Ausdrücke

Liste 3.4 zeigt vier Reguläre Ausdrücke, die nach den in Liste 3.3 genannten Situationen entwickelt werden. Die Regulären Ausdrücke können auch andere gegenwärtigen Form sein.

- “<” +Suchobject+ “/>”;
- “<” +Suchobject +“ ” + “/>”;
- “<” +Suchobject + “. *? </” + Suchobject + “>”;
- “<” +Suchobject + “ ” + “. *? </” + Suchobject + “>”.

Liste 3.4 Geeignete Ausdrücke für die vier verschiedenen Situationen

3.3.1 Ausdruck für Knoten ohne Werte und Attribute

Für die erste Situation ist der in Liste 3.5 gezeigte Ausdruck entwickelt worden. Mit Hilfe des Ausdrucks können alle Knoten, die mit gewünschtem Kennzeichen keine eigenen Text-Werte und neben Kennzeichen keine eigene Attribute enthalten, im XML-Dokument erhalten.

- “<” +Suchobject+ “/>”;

Liste 3.5 Regulärer Ausdruck für Knoten ohne Werte und Attribute

Ein XML-Dokument, das als Beispiel in Liste 3.6 dargestellt wird:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<variables>
<variable/>
</variables>
```

Liste 3.6 XML-Dokument Beispiel

Das in Liste 3.6 gezeigte XML-Dokument kann durch den Regulären Ausdruck “<” +Variable+ “/>” behandelt werden, wobei der in Liste 3.5 gezeigte Ausdruck als Pattern verwendet wird. Das Ergebnis ist in Liste 3.7 dargestellt.

- <variable/>

Liste 3.7 Ergebnis durch Verwendung des in Liste 3.7 gezeigten Ausdrucks

Für die Situation ist das Ergebnis das endgültige Ergebnis, weil XML-Dokument in der Situation keinen Inhalt hat. Deshalb ist das Ergebnis nur das Kennzeichen <variable/>. Natürlich braucht bei der Situation keine „Substring-Operation“ verwendet werden. Aber mit DOM+XPath Technologie kann nur Leerzeichen als Ergebnis erhalten werden.

3.3.2 Ausdruck für Knoten ohne Werte aber mit Attributen

Für die zweite Situation ist der in Liste 3.8 gezeigte Ausdruck entwickelt worden. Mit Hilfe des Ausdrucks können alle Knoten, die mit gewünschtem Kennzeichen keine eigenen Text-Werte aber mit eigenen Attributen enthalten, im XML-Dokument erhalten.

- “<” + Suchobject + “.*?/>”

Liste 3.8 Regulärer Ausdruck für Knoten ohne Werte aber mit Attributen

Ein XML-Dokument, das als Beispiel in Liste 3.9 dargestellt wird.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<variables>
<variable name="cd1" type="string"/>
<variable name="cd2" type="string"/>
<variable name="cd3" type="string"/>
</variables>
```

Liste 3.9 XML-Dokument Beispiel des Knotens mit Attributen aber ohne Werte

Das in Liste 3.9 gezeigte XML-Dokument kann durch den Regulären Ausdruck “<” + Variable + “.*?/>” behandelt werden, wobei der in Liste 3.8 gezeigte Ausdruck als Pattern verwendet wird. Das Ergebnis ist wie in Liste 3.10 dargestellt.

- <variable name="cd1" type="string"/>
- <variable name="cd2" type="string"/>
- <variable name="cd3" type="string"/>

Liste 3.10 Ergebnis durch Verwendung von dem in Liste 3.8 gezeigten Ausdruck

Für die Situation ist das Ergebnis das endgültige Ergebnis, weil XML-Dokument in der Situation keinen Inhalt hat. Deshalb ist das Ergebnis nur das Kennzeichen wie <variable name="cd1" type="string"/>. Natürlich wird bei der Situation ebenfalls keine „Substring-Operation“ verwendet.

3.3.3 Ausdruck für Knoten mit Text-Werten aber ohne Attribute

Für die dritte Situation ist der in Liste 3.11 gezeigte Ausdruck entwickelt worden. Mit Hilfe von dem Ausdruck können alle Knoten, die mit gewünschtem Kennzeichen eigene Text-Werte aber keine eigene Attribute enthalten, im XML-Dokument erhalten.

- “<” + Suchobject + “>” + “.*? </” + Suchobject + “>”

Liste 3.11 Regulärer Ausdruck Knoten mit Werten aber ohne Attribute

Ein XML-Dokument, das als Beispiel in Liste 3.12 gezeigt wird.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book>
<title>Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
</bookstore>
```

Liste 3.12 XML-Dokument Beispiel des Knotenes mit Werten aber ohne Attribute

Das in Liste 3.12 gezeigte XML-Dokument kann durch den Regulären Ausdruck “<” + title + “>” + “.*? </” + title + “>” behandelt werden, wobei der in Liste 3.11 gezeigte Ausdruck als Pattern verwendet wird. Das Ergebnis wird in Liste 3.13 gezeigt.

```
<title>Harry Potter</title>
```

Liste 3.13 Ergebnis durch Verwendung des in Liste 3.11 gezeigten Ausdrucks

Für die Situation ist das in Liste 3.13 gezeigte Ergebnis kein endgültiges Ergebnis, weil XML-Dokument in der Situation Inhalt hat. Deshalb sollte das endgültige Ergebnis nur der Inhalt zwischen Kennzeichen <title>...</title> sein. Es muss bei der Situation „Substring-Operation“ verwendet werden, um das endgültige Ergebnis zu erhalten.

3.3.4 Ausdruck für Knoten mit Text-Werten und Attributen

Für die vierte Situation ist der in Liste 3.14 gezeigte Ausdruck entwickelt worden. Mit Hilfe des Ausdrucks können alle Knoten, die mit gewünschtem Kennzeichen eigene Text-Werte und Attribute enthalten, im XML-Dokument erhalten.

- “<” + Suchobject + “.*? </” + Suchobject + “>”

Liste 3.14 Regulärer Ausdruck für Situation der Knoten mit Werten und Attributen

Ein XML-Dokument, das als Beispiel in Liste 3.15 gezeigt wird.

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="children">

<title lang="en">Harry Potter</TITLE>

<author>J K. Rowling</author>

<year>2005</year>

<price>29.99</price>

</book>

<book category="cooking">

<title lang="en">Everyday Italian</TITLE>

<author>Giada De Laurentiis</author>

<year>2005</year>

<price>30.00</price>

</book>

</bookstore>

```

Liste 3.15 XML-Dokument Beispiel des Knotenes mit Attributen und Werten

Das in Liste 3.15 gezeigte XML-Dokument kann durch den Regulären Ausdruck “<” + book + “ .*? </” + book + “>” behandelt werden, wobei der in Liste 3.14 gezeigte Ausdruck als Pattern verwendet wird. Das Ergebnis ist wie in Liste 3.16 dargestellt.

```

<book category="children">

<title lang="en">Harry Potter</title>

<author>J K. Rowling</author>

<year>2005</year>

<price>29.99</price>

</book>

<book category="cooking">

<title lang="en">Everyday Italian</title>

```

```
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
```

Liste 3.16 Ergebnis durch Verwendung des in Liste 3.14 gezeigten Ausdrucks

Für die Situation ist das in Liste 3.16 gezeigte Ergebnis kein endgültiges Ergebnis, weil XML-Dokument in der Situation Inhalt hat. Deshalb sollte das endgültige Ergebnis nur der Inhalt zwischen Kennzeichen `<book category="children">...</book>` sein. Es muss bei der Situation „Substring-Operation“ verwendet werden, um das endgültige Ergebnis zu erhalten.

3.4 String-Operationen für Erhalt der endgültigen Inhalte

Abschnitt 3.3 zeigt vier verschiedene Reguläre Ausdrücke. Man kann durch verschiedene Situationen aus den vier Regulären Ausdrücken den geeigneten Regulären Ausdruck wählen, um die gewünschten Ergebnisse zu erhalten. Die Ergebnisse für die dritte und vierte Situation sind noch keine endgültigen Ergebnisse, weil es bei den Situationen Texte-Inhalte der gewünschte Knoten gibt. Diese Ergebnisse benötigen zusätzliche Bearbeitung, um als endgültige Ergebnisse festzustehen. Die alle entstehenden Zwischenergebnisse werden in einem String-ArrayList Object gespeichert, damit die Zwischenergebnisse von String-Operationen weiter bearbeitet werden können.

Abschnitt 3.4 beschreibt die zusätzlichen String-Operationen, mit denen die endgültigen Ergebnisse erhalten werden können.

Die Liste 3.17 zeigt ein Code-Fragment, um einen gewünschten Inhalt abzuschneiden.

```
while(matcher.find()){
    int start = matcher.start();
    int end = matcher.end();
    String match = tempString.substring(start, end);
    int m = match.indexOf(">")+1;
    int n = match.indexOf("</"+var+">");
    match = match.substring(m, n);
    result.add(match);
}
```

```

    i++;
}

```

Liste 3.17 Code für das Abschneiden gewünschter Inhalte

In der Liste 3.17 wird ein Java Code-Fragment gezeigt. Das ist eine While-Schleife. In der Voraussetzung von der Schleife ist matcher Pattern, das von im Abschnitt 3.3 erklärtem Ausdruck abgeleitet wird. Die Methode matcher.find() prüft den String-Fluss, der aus dem XML-Dokument transformiert wird. Wenn in der Prüfung mit gewünschten Charakteren getroffen hat, dann positioniert durch Hilfe von den Methoden match.start() und match.end() die genauen Stellen der gewünschten Inhalte. Das Statement tempString.substring(start, end) schneidet die gewünschten Inhalte aus dem String-Fluss ab. Die endgültigen Ergebnisse werden durch Hilfe von normalen String-Operationen match.indexOf(">")+1 und indexOf("</"+var+">") positioniert. Und durch die String-Operation substring(m, n) werden die endgültigen Ergebnisse erhalten. Am Ende jeder Schleife wird ein endgültiges Ergebnis in eine ArrayList<String> Object result eingespeichert.

3.5 Das vollständige Verfahren

In den Abschnitten 3.2-3.4 wurden die wichtigsten Teile des Verfahrens erklärt. Im folgenden Abschnitt wird durch das Verfahren-Diagramm das vollständige Verfahren beschrieben.

Bild 3.3 zeigt Teil 1 vom Verfahren.

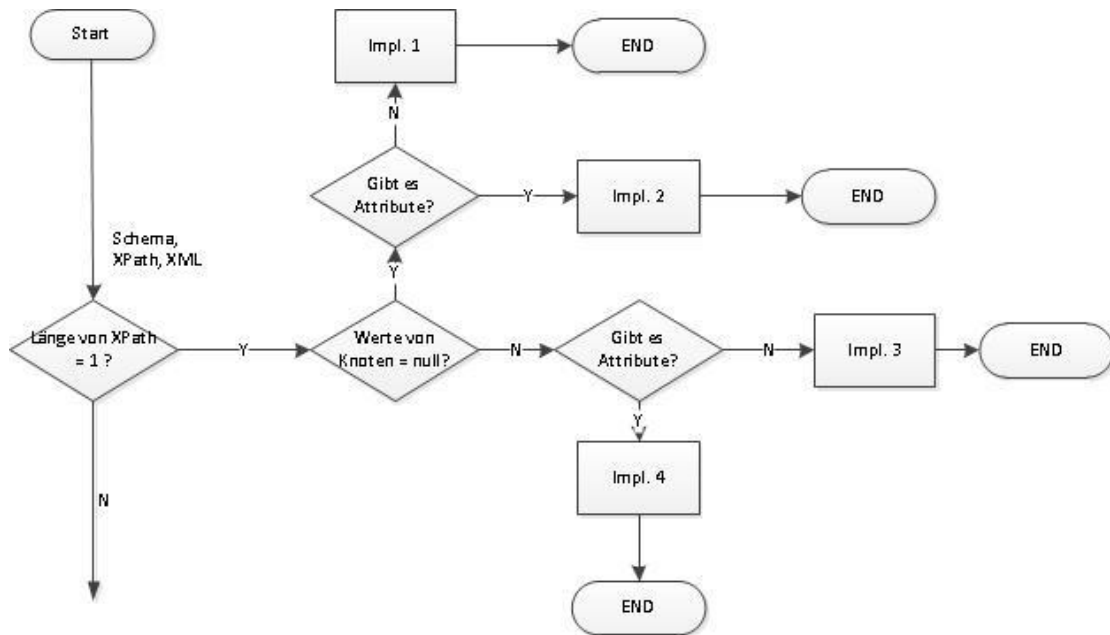


Bild 3.3 Teil 1 vom Verfahren

Im Bild 3.3 wird dargestellt, dass das Verfahren XPath-Ausdruck, XML-Dokument und XML-Schema eingelesen werden. Dann prüft das Verfahren die Länge des importierten XPath-Ausdrucks. Diese Prüfung verwendet `java.Util.StringTokenizer`-Class, um XPath-Ausdruck in einzelne Schritte zu trennen. Die geteilten Schritte werden individuell in einem Array gespeichert. Durch Länge der Array kann die Länge der Schritte vom importierten XPath-Ausdruck erhalten werden. Das Verfahren wird von dem in Liste 3.18 gezeigten Code realisiert.

```

import java.util.StringTokenizer;

public class strOpA {

    /*
     * die Class benutzt StringTokenizer, die Xpath Expression zu schneiden.
     */

    private String[] xpRead(String xpath){

        StringTokenizer tok = new StringTokenizer(xpath,"/[ ]//");

        String[] xp = new String[tok.countTokens()];//def eine array fuer Tokens;

        int i = 0;

        //verteiln die Expression in die Array.

        while(tok.hasMoreTokens()){

            xp[i] = tok.nextToken();

            System.out.println(xp[i]);

            i++;

        }

        return xp;

    }

}

```

Liste 3.18 Code für den Erhalt der Länge von XPath-Ausdruck

In der Liste 3.18 gezeigte Methode `xpRead(String xpath)` liest den XPath-Ausdruck als String-Object ein. Durch Hilfe einer vordefinierte Zeichengruppe, die in der Liste 3.18 „/[]//“ dargestellt ist, kann der XPath-Ausdruck von StringTokenizer-Object zerlegt werden. Das heißt, dass die alle Zeichen in der Zeichengruppe aus dem XPath-Ausdruck weggelassen werden. Die restlichen Inhalte vom XPath-Ausdruck werden als Such-

Schlüsselworte in eine Array als String gespeichert. Die Schlüsselwörter werden als Suchziele bei nächsten Schritten verwendet.

Nachdem die Schlüsselwörter erhalten wurden, wird geprüft, ob es keinen Text bei dem gewünschte Knoten gibt, wenn nein, wird geprüft, ob Knoten Attribute enthält. Falls nein, wird die Impl. 1 durchgeführt, sonst wird die Impl. 2 durchgeführt. Wenn Knoten Text und Attribute enthält, dann wird Impl. 4 durchgeführt. Wenn Knoten Text enthält aber keine Attribute, dann wird Impl. 3 durchgeführt.

In Liste 3.19 erklärt Impl. 1–Impl. 5

Impl. 1	Der in Liste 3.5 gezeigte Reguläre Ausdruck wird benutzt, um den Knoten ohne Inhalt und Attribute zu bearbeiten.
Impl. 2	Der in Liste 3.8 gezeigte Reguläre Ausdruck wird benutzt, um den Knoten ohne Inhalt aber mit Attributen zu bearbeiten.
Impl. 3	Der in Liste 3.11 gezeigte Reguläre Ausdruck wird benutzt, um den Knoten mit Inhalt aber ohne Attribute zu bearbeiten.
Impl. 4	Der in Liste 3.14 gezeigte Reguläre Ausdruck wird benutzt, um den Knoten mit Inhalt und Attributen zu bearbeiten.
Impl. 5	Durch Situation wird oben genannte Aktion verwendet und verglichen.

Liste 3.19 Erklärung für die Bearbeitung von 1 – 5

Wir nehmen das in Liste 3.15 gezeigte XML-Dokument als Beispiel, um Impl. 5 zu erklären. Beispielsweise wenn wir das Buch suchen möchten, das den Preis 30.00 hat. Der XPath-Ausdruck ist wie in 3.20 beschrieben.

● `/bookstore/book[price=30.00]`

Liste 3.20 XPath-Ausdruck für Suche des Buches, das price=30.00 hat

Für den Ersatz des XPath-Ausdrucks wird zuerst die Impl. 3 für Knoten <bookstore> benutzt, weil Knoten <bookstore> Text-Werte enthält, aber keine Attribute. Danach wird Impl. 4 für Knoten <book> verwendet, weil Knoten <book> Inhalte und Attribute enthält. Danach kann man Impl. 3 für Knoten <price> noch mal verwenden und das Ergebnis im anderen ArrayList<String> Object zu speichern und mit 30.00 zu vergleichen. Gibt das Platz-Nummer vom price Knoten aus, der mit Werte 30.00 ist. Dann kann man durch die Platz-Nummer aus dem ersten ArrayList<String> Object das Buch finden.

Bild 3.4 zeigt Teil 2 vom Verfahren. Das Bild zeigt, wenn es beim XPath-Ausdruck Vergleichszeichen gibt, dann wird Impl. 5 durchgeführt, wenn es im XPath-Ausdruck keine Vergleichszeichen gibt, dann wird geprüft, ob Knoten Attribute enthält. Falls ja, wird Impl. 4 durchgeführt, sonst wird Impl. 3 durchgeführt.

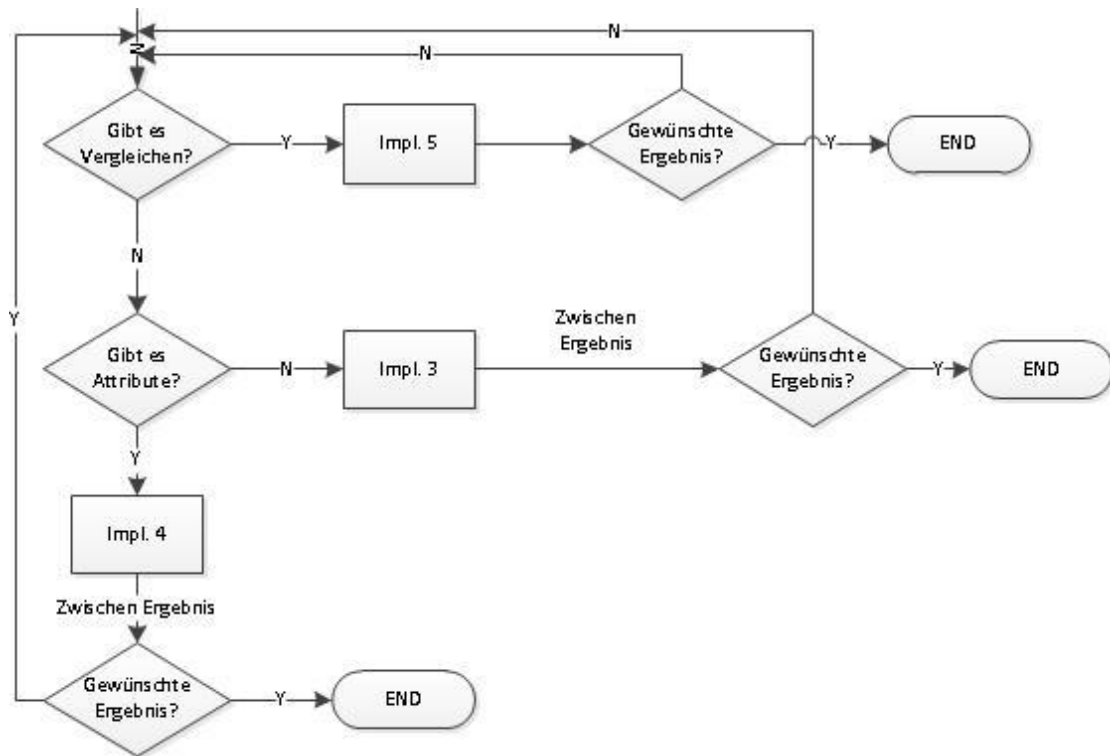


Bild 3.4 Teil 2 vom Verfahren

3.6 Das verkürzte Verfahren

Normalerweise wenn wir alle Knoten <title>, die in Liste 3.15 gezeigt werden, erhalten möchten, müssen wir zuerst den Regulären Ausdruck "<" + bookstore + ">" + ".?</" + bookstore + ">" benutzen. Dann wird der Reguläre Ausdruck "<" + book + ".?</" + book + ">" auf dem Zwischenergebnis verwendet. Anschliessend wird der Reguläre Ausdruck "<" + author + ">" + ".?</" + author + ">" noch mal benutzt. Dann kann man das endgültige Ergebnis bekommen. Aber wenn ein Knoten in einem XML-Dokument wohl definiert wird, beispielsweise wie das der Knoten <author> in Liste 3.15 zeigt, wird im XML-Dokument nur von Knoten <book> enthalten. Dann kann man für den Knoten <author> die fundamentale String-Operation direkt verwenden. Das heisst, dass wir nur die geeigneten String-Operationen zu verwenden brauchen, die den XPath-Ausdruck „//author“ ersetzen können. Liste 3.21 zeigt die Verkürzung.

```
<code>"<" + author + ">" + ".?</" + author + ">"</code>
```

Liste 3.21 XPath-Ausdruck für Suche aller in Liste 3.15 gezeigten Knoten <author>

In diesem Kapitel werden alle XML-Dokumente in vier Arten eingeordnet und es werden vier geeignete Reguläre Ausdrücke entwickelt. Mit den Ausdrücken werden XML-Dokumente korrekt behandelt. Im nächsten Kapitel wird die String-Operation implementiert und anschliessend mit der DOM+XPath Technik verglichen. Ziel ist es herauszufinden welche Technik effizienter ist.

4. Implementation

In diesem Kapitel wird der im Kapitel 3 gezeigte Algorithmus implementiert, um nachzuprüfen, ob der Algorithmus funktionsfähig ist. Und wir werden in diesem Kapitel den Algorithmus mit DOM+XPath vergleichen, um zu erfahren, welche Technik effizienter ist.

Wir werden in der gleichen Hardware- und Softwareumgebung die in Kapitel 3 genannten fünf Situationen testen und mit dem gleichen XML-Dokument auf der selben Plattform die Technik DOM+XPath ebenfalls testen. Ziel ist der Vergleich.

Das Testen besteht aus zwei Teilen. Einen Teil des Testens stellt die Gebrauchszeit für jeden Testfall dar. Beim zweiten Testteil wird die Anforderung des Arbeitsspeichers bei der Implementation getestet.

Wir werden für jeden Testfall die beiden Techniken verwenden. Für jede Technik führen wir den Test mit einem XML-Dokument zuerst ein einziges Mal durch. Anschliessend führen wir mit dem selben XML-Dokument für jede Technik 10000 mal durch. Und die durchschnittlichen Zeiten werden berechnet.

4.1 Hintergrund und Technologie der Implementation

In diesem Abschnitt werden die beim Test verwendete Plattform und die entsprechende Technologie beschrieben.

4.1.1 Umgebung für den Test

Die Testplattform wird wie in Liste 4.1 gezeigt aus folgender Hardware- und Software bestehen.

- CPU: Interl(R) Atom(TM) N270 @ 1.60 GHz
- Arbeitsspeicher: DDR2 800 1G x 2 = 2 G
- Betriebssystem: Windows XP Home Edition SP3
- Anwendungssoftware: Eclipse GALILEO; JDK 1.6.22

Liste 4.1 Die Testumgebung für die Implementation des Verfahrens

Wir werden das in Liste 4.2 gezeigte Code-Fragment verwenden, um die Laufzeit eines Verfahrens zu messen.


```

long start = System.currentTimeMillis();

...

Code

...

long end = System.currentTimeMillis();

long time = end - start;

```

Liste 4.2 Das Code-Fragment für die Messung der Laufzeit

Wie die Liste 4.2 zeigt, bevor ein Verfahren durchgeführt wird, kann die Anfangszeit durch die Methode `System.currentTimeMillis()` ermittelt werden. Nachdem das Verfahren durchgeführt wurde, dann kann die Endzeit durch die Methode erhalten werden. Danach kann die Laufzeit von der Implementation eines Verfahrens berechnet werden.

Wir werden das in Liste 4.3 gezeigte Code-Fragment verwenden, um den benötigten Arbeitsspeicher bei der Laufzeit eines Verfahrens zu messen.

```

long eMem = Runtime.getRuntime().freeMemory();

...

Code

...

long eMem = Runtime.getRuntime().freeMemory();

System.out.println("Gebrauchte Memory" + (sMem - eMem) + "Bytes");

```

Liste 4.3 Das Code-Fragment für die Messung des benötigten Arbeitsspeichers

Wie die Liste 4.3 zeigt, bevor ein Verfahren durchgeführt wird, kann der freie Arbeitsspeicher durch die Methode `Runtime.getRuntime().freeMemory()` ermittelt werden. Nachdem das Verfahren durchgeführt wurde, kann der freie Arbeitsspeicher durch die Methode nochmals ermittelt werden. Danach kann der benötigte Arbeitsspeicher von der Implementation eines Verfahrens berechnet werden.

4.1.2 Der Code für Analyse eines XML-Dokuments mit DOM+XPath

Wir werden das in Liste 4.4 gezeigte Code-Fragment verwenden, um ein XML-Dokument zu analysieren. Das Code-Fragment verwendet die populäre Technik DOM+XPath.

```

import java.io.*;

import java.net.URL;

import java.util.*;

import javax.xml.XMLConstants;

import javax.xml.namespace.NamespaceContext;

import javax.xml.parsers.*;

import javax.xml.xpath.*;

import org.w3c.dom.*;

import org.xml.sax.SAXException;

public class dom4xp1 {

    /**
     * Evaluate an XML input stream.
     * @param in The XML input stream.
     */
    Document eval(InputStream in)
        throws ParserConfigurationException, SAXException, IOException,
        XPathExpressionException {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        docFactory.setNamespaceAware(true);
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document doc = docBuilder.parse(in);

        return doc;
    }

    /**
     * Evaluate an String Object using the given xpath.
     * @param xpathExpr the input xpath Expression.

```

```

*/
XPathExpression xp(String xpathExpr) throws XPathExpressionException{
    XPath xpath = XPathFactory.newInstance().newXPath();
    XPathExpression expr = xpath.compile(xpathExpr);
    return expr;
}

/**
 * Evaluate an XML Document using the compiled XPathExpression Object.
 *
 */
void suchEval(XPathExpression expr, Document doc, Collection<String> result) throws
XPathExpressionException{
    NodeList nodeList = (NodeList) expr.evaluate(doc, XPathConstants.NODESET);
    for (int i = 0; i < nodeList.getLength(); ++i) {
        Node node = nodeList.item(i);
        result.add(node.getNodeValue());
    }
    for (String name : result) {
        System.out.println(name);
    }
}

/**
 * Print the result.
 *
 * @param result The collection of results.
 * @param out The print stream to use.
 */

```

```

void printResult(Collection<String> result, PrintStream out) {

    // print result

    for (String name : result) {

        out.println(name);

    }

}

/**

 * Read all 'price' elements from an cd XML file. The names are
 * kept sorted and unique using a TreeSet.

 */

public void readPrice(XPathExpression befehl, Document doc) throws Exception {

    Collection<String> result = new TreeSet<String>();

    suchEval(befehl, doc, result);

    printResult(result, System.out);

}

public static void main(String[] args) throws Exception {

    long sMem = Runtime.getRuntime().freeMemory();

    dom4xp1 xpReader = new dom4xp1();

    long s1=System.currentTimeMillis();

        InputStream in = new FileInputStream("c:/books.xml");

    Document doc = xpReader.eval(in);

    XPathExpression xp = xpReader.xpath("//TITLE/text()");

    long e1=System.currentTimeMillis();

    long s2 = e1 - s1;

    long start=System.currentTimeMillis();

    for(int i = 0; i < 10000; i++){

```

```

    xpReader.readPrice(xp, doc);
}

    long end=System.currentTimeMillis();

    long eMem = Runtime.getRuntime().freeMemory();

    System.out.println("gebraucht memory: "+(sMem-eMem)+"bytes");

    System.out.println("laufzeit: "+(end-start)+"ms");

    System.out.println("*****: " + s2 + "ms");

    System.out.close();
}
}

```

Liste 4.4 Code-Fragment für das Analysieren eines XML-Dokuments

4.2 Implementation

In diesem Abschnitt führen wir mit verschiedenen XML-Dokumenten verschiedene Verfahren durch. Und wir werden durch Diagramme zeigen, was die Unterschiede zwischen DOM+XPath und String-Operationen bei der Anforderung von Rechenzeit und Arbeitsspeicher sind.

Liste 4.5 zeigt den XPath-Ausdruck, der von verschiedenen Regulären Ausdrücken nach verschiedenen Situationen simuliert wird. Dann können wir diese Regulären Ausdrücke als Grundlage nehmen, um die Analysetechnik bzw. DOM+XPath für XML-Dokument zu ersetzen.

● //Suchobject

Liste 4.5 Der von regulären Ausdrücken simulierte XPath-Ausdruck

4.2.1 Implementation für Knoten ohne Werte und Attribute

In diesem Abschnitt wird das Verfahren durchgeführt, um Knoten ohne Werte und Attribute zu behandeln. Das Verfahren ist die in Liste 3.19 gezeigte Impl1. Das wird benutzt, den geeigneten XPath-Ausdruck zu ersetzen.

Liste 4.6 zeigt das XML-Dokument, das als Testfall verwendet wird.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<variables><variable/> </variables>
```

Liste 4.6 XML-Dokument als Testfall

Liste 4.7 zeigt den Regulären Ausdruck, der benutzt wird, um den in Liste 4.5 gezeigten XPath-Ausdruck zu simulieren, Knoten ohne Werte und Attribute auszusuchen.

```
● "<" +Suchobject+ ">";
```

Liste 4.7 Regulärer Ausdruck für Knoten ohne Werte und Attribute

In dem Bild 4.1 wird der Vergleich der Laufzeit zwischen DOM+XPath und String-Operation gezeigt. Das verwendete XML-Dokument steht in Liste 4.6. Der gesuchte Zweck ist „varibale“.

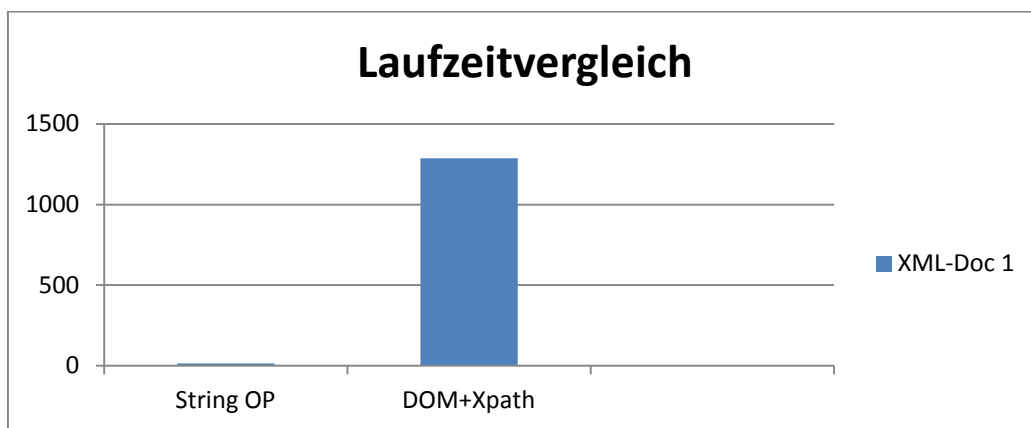


Bild 4.1 Laufzeitvergleich mit Suchzweck ohne Attribute und Werte (Einheit: ms)

Bei dem Suchverfahren braucht String-Operation 15 ms, aber DOM+XPath braucht 1287 ms. Das Suchergebnis sollte leer sein, weil es bei dem Knoten keinen Wert gibt.

Im Bild 4.2 zeigt der Vergleich die Anforderung an den Arbeitsspeicher zwischen beiden Techniken. Das verwendete XML-Dokument ist die Liste 4.6. Der gesuchte Zweck ist „varibale“.

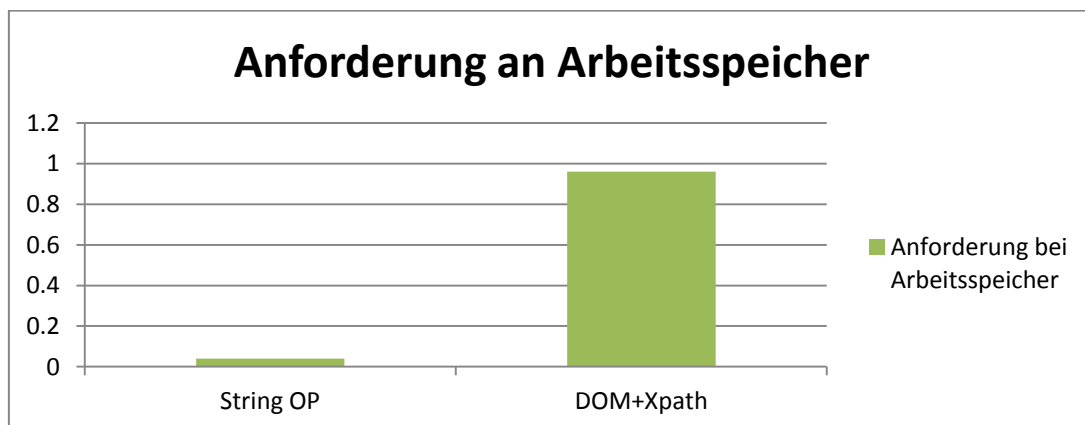


Bild 4.2 Anforderung an Arbeitsspeicher für beide Techniken (Einheit: MB)

Bei dem Test benötigt die String-Operation beim Arbeitsspeicher 0.04MB, aber DOM+XPath braucht beim Arbeitsspeicher 0.96MB.

In dem Bild 4.3 zeigt der Vergleich die durchschnittliche Laufzeit zwischen DOM+XPath und String-Operation nach 10000 mal. Das verwendete XML-Dokument ist in der Liste 4.6 dargestellt. Der gesuchte Zweck ist „varibale“.

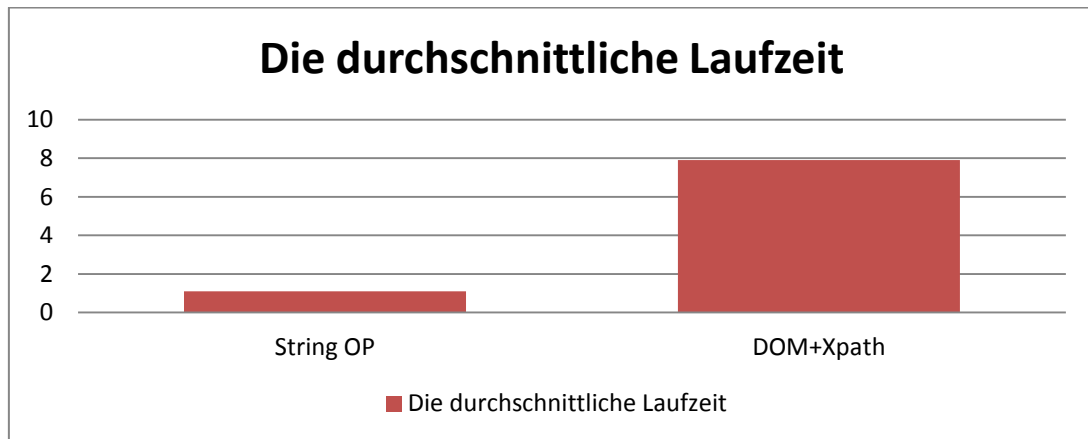


Bild 4.3 Die durchschnittliche Laufzeit nach 10000 Läufen (Einheit: ms)

Bei dem Suchverfahren braucht die String-Operation durchschnittlich 1.1 ms, aber DOM+XPath braucht 7.9 ms. Das Suchergebnis sollte leer sein, weil es bei dem Knoten keinen Wert gibt.

4.2.2 Implementation für Knoten mit Attributen aber ohne Werte

In diesem Abschnitt wird das Verfahren durchgeführt, um den Knoten ohne Werte aber mit Attributen zu behandeln. Das Verfahren ist die in Liste 3.19 gezeigte Impl2. Das wird benutzt, den geeigneten XPath-Ausdruck zu ersetzen.

Liste 4.8 zeigt das XML-Dokument, das als Testfall verwendet wird.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<variables>
<variable name="cd1" type="string"/>
<variable name="cd2" type="string"/>
<variable name="cd3" type="string"/>
</variables>
```

Liste 4.8 XML-Dokument als Testfall

Liste 4.9 zeigt den Regulären Ausdruck, der benutzt wird, um den in Liste 4.5 gezeigten XPath-Ausdruck zu simulieren, den Knoten ohne Werte aber mit Attributen auszusuchen.

● “<” + Suchobject + “.*?/>”;
Liste 4.9 Regulärer Ausdruck für Knoten ohne Werte aber mit Attributen

Das Bild 4.4 zeigt den Vergleich der Laufzeit zwischen DOM+XPath und String-Operation. Das verwendete XML-Dokument ist die Liste 4.8. Der gesuchte Zweck ist „variable“.

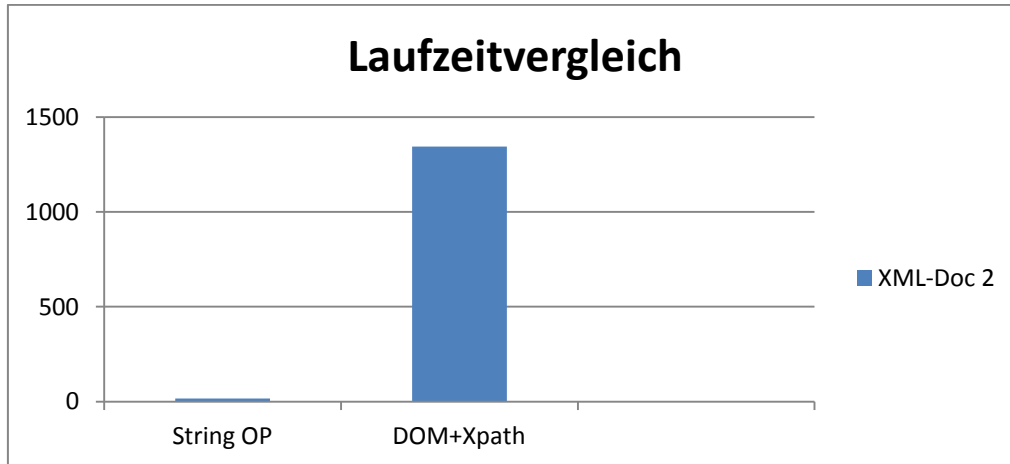


Bild 4.4 Laufzeitvergleich mit Suchzweck ohne Werte aber mit Attributen (Einheit:ms)

Bei dem Suchverfahren benötigt die String-Operation 16 ms, aber DOM+XPath braucht 1344 ms. Das Suchergebnis sollte leer sein, weil es bei dem Knoten keinen Wert gibt.

In dem Bild 4.5 zeigt der Vergleich die Anforderung an den Arbeitsspeicher zwischen beiden Techniken. Das verwendete XML-Dokument ist in Liste 4.8 dargestellt. Der gesuchte Zweck ist „variable“.

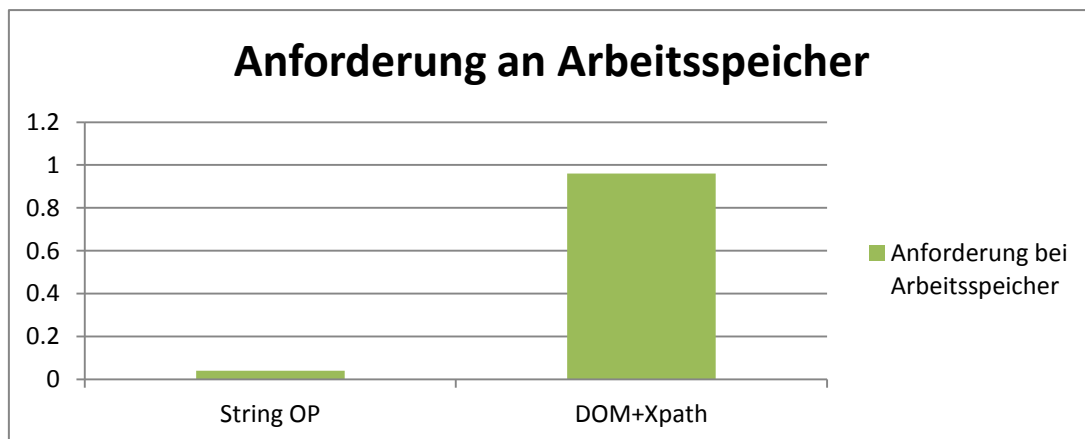


Bild 4.5 Anforderung an Arbeitsspeicher für beide Techniken (Einheit: MB)

Bei dem Test benötigt die String-Operation den Arbeitsspeicher 0.041MB, aber DOM+XPath braucht den Arbeitsspeicher 0.97MB.

In dem Bild 4.6 zeigt der Vergleich eine durchschnittliche Laufzeit zwischen DOM+XPath und String-Operation nach 10000 mal. Das verwendete XML-Dokument ist in Liste 4.8 dargestellt. Der gesuchte Zweck ist „variable“.

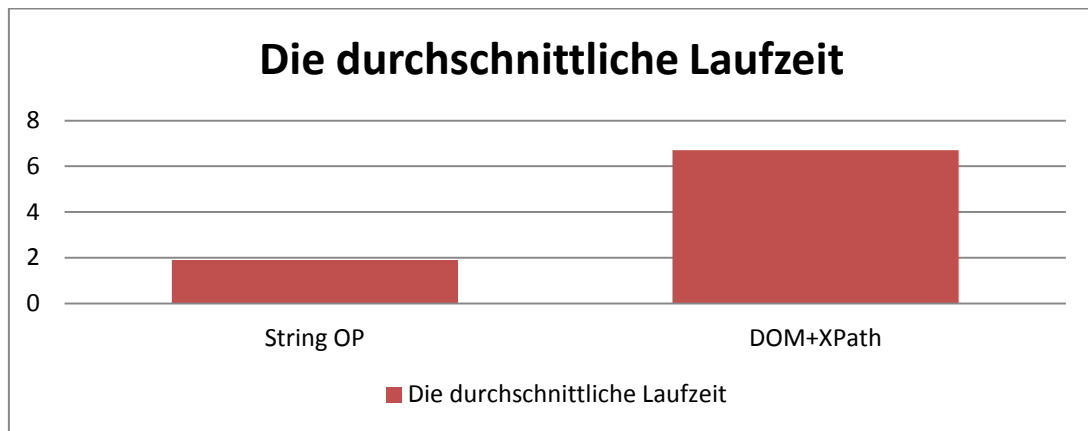


Bild 4.6 Die durchschnittliche Laufzeit nach 10000 Läufen (Einheit: ms)

Bei dem Suchverfahren braucht String-Operation durchschnittlich 1.9 ms, aber DOM+XPath braucht 6.7 ms. Das Suchergebnis sollte leer sein, weil es bei dem Knoten keinen Wert gibt.

4.2.3 Implementation für Knoten mit Wert aber ohne Attribute

In diesem Abschnitt wird das Verfahren durchgeführt, um den Knoten mit Werten aber ohne Attribute zu behandeln. Das Verfahren ist die in Liste 3.19 gezeigte Impl3. Das wird benutzt, um den geeigneten XPath-Ausdruck zu ersetzen.

Der Anhang A zeigt die XML-Dokumente, die als Testfall verwendet werden.

Die Liste 4.10 zeigt den Regulären Ausdruck, der benutzt wird, um den in Liste 4.5 gezeigten XPath-Ausdruck zu simulieren, den Knoten mit Werten aber ohne Attribute auszusuchen.

- “<”+Suchobject + “.*? </” + Suchobject + “>”;

Liste 4.10 Regulärer Ausdruck für Knoten mit Werten aber ohne Attribute

Im Bild 4.7 wird der Vergleich der Laufzeit zwischen DOM+XPath und String-Operation gezeigt. Die verwendeten XML-Dokumente sind im Anhang. Der gesuchte Zweck ist der Wert von Knoten „TITLE“ von cd.xml, Knoten „price“ von simple.xml, sowie Knoten „author“ von books.xml. Wobei cd.xml 5KB Grösse hat, simple.xml 2KB Grösse und books.xml die Grösse von 1KB besitzt.

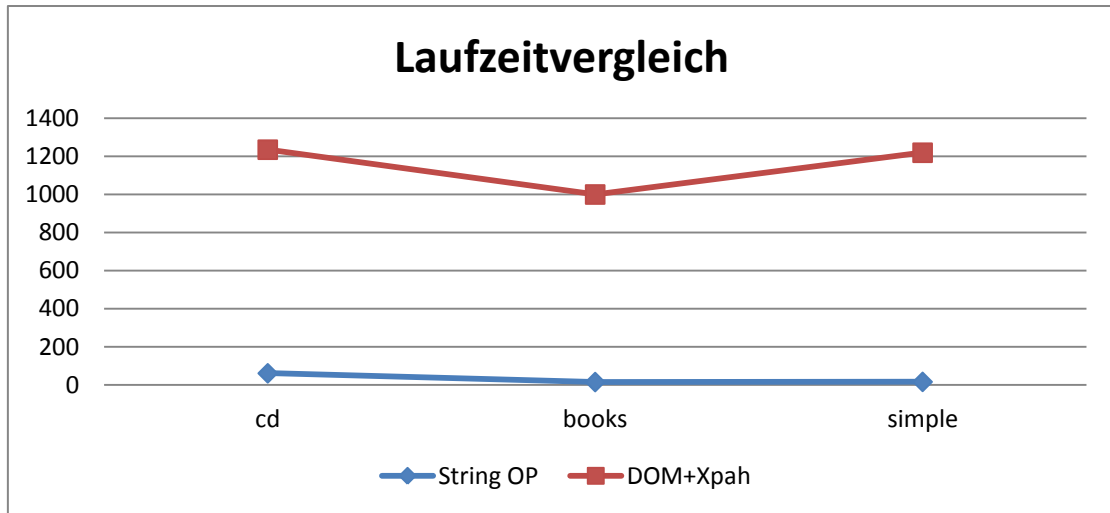


Bild 4.7 Laufzeitvergleich für Suchzweck ohne Attribute aber mit Werten (Einheit: ms)

Bei dem Suchverfahren benötigt String-Operation 62ms für cd.xml, 16ms für books.xml und 17ms für simple.xml. Aber DOM+XPath benötigt 1235 ms für cd.xml, 1000ms für books.xml und 1219ms für simple.xml.

Bild 4.8 zeigt den Vergleich bei der Anforderung an den Arbeitsspeicher zwischen beiden Techniken. Das verwendete XML-Dokument befindet sich im Anhang.

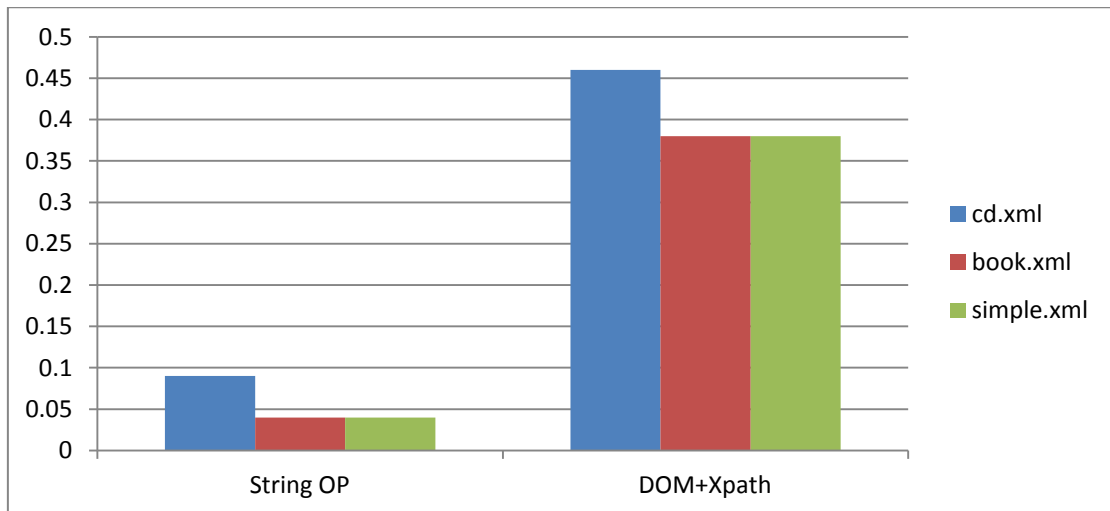


Bild 4.8 Anforderung an Arbeitsspeicher für beide Techniken (Einheit: MB)

Bei dem Test benötigt die String-Operation den Arbeitsspeicher 0.09MB, 0.04MB und 0.04MB. DOM+XPath braucht den Arbeitsspeicher 0.46MB, 0.38MB und 0.38MB.

Das Bild 4.9 zeigt den Vergleich der durchschnittlichen Laufzeit zwischen DOM+XPath und String-Operation nach 10000 mal. Die verwendeten XML-Dokumente sind im Anhang dargestellt.

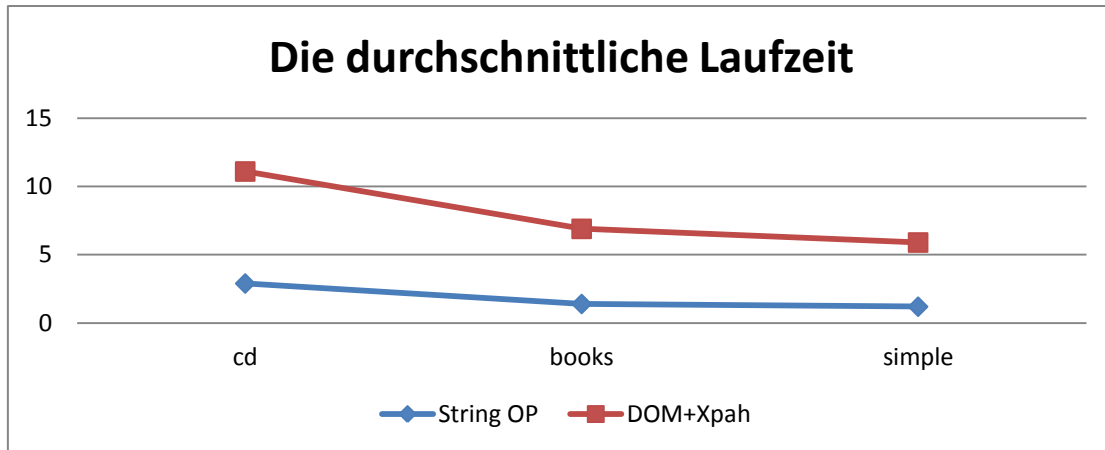


Bild 4.9 Die durchschnittliche Laufzeit nach 10000 Läufen (Einheit: ms)

Bei dem Suchverfahren benötigt String-Operation durchschnittlich 2.9ms für cd.xml, 1.4ms für books.xml und 1.2ms für simple.xml. DOM+XPath braucht 11.1 ms, 6.9ms sowie 5.9ms.

4.2.4 Implementation für Knoten mit Werten und Attributen

In dem Abschnitt wird das Verfahren durchgeführt, um den Knoten mit Werten und Attributen zu behandeln. Das Verfahren ist die in Liste 3.19 gezeigte Impl4. Das wird benutzt um den geeigneten XPath-Ausdruck zu ersetzen.

Im Anhang werden XML-Dokumente gezeigt, die als Testfall verwendet werden.

Liste 4.11 zeigt den Regulären Ausdruck, der benutzt wird, um den in Liste 4.5 gezeigten XPath-Ausdruck zu simulieren, den Knoten mit Werten und Attributen auszusuchen.

● “<”+Suchobject + “ ” + “.*? </” + Suchobject + “>”

Liste 4.11 Regulärer Ausdruck für Knoten mit Werten und Attributen

Bild 4.10 zeigt den Vergleich der Laufzeiten zwischen DOM+XPath und String-Operation. Die verwendeten XML-Dokumente sind im Anhang dargestellt. Der gesuchte Zweck sind die Werte von Knoten „ARTIST“ von cd.xml, von Knoten „name“ von simple.xml, sowie von Knoten „TITLE“ von books.xml. Wobei cd.xml eine 5KB Grösse hat, simple.xml eine 2KB Grösse hat und books.xml hat eine Grösse von 1KB.

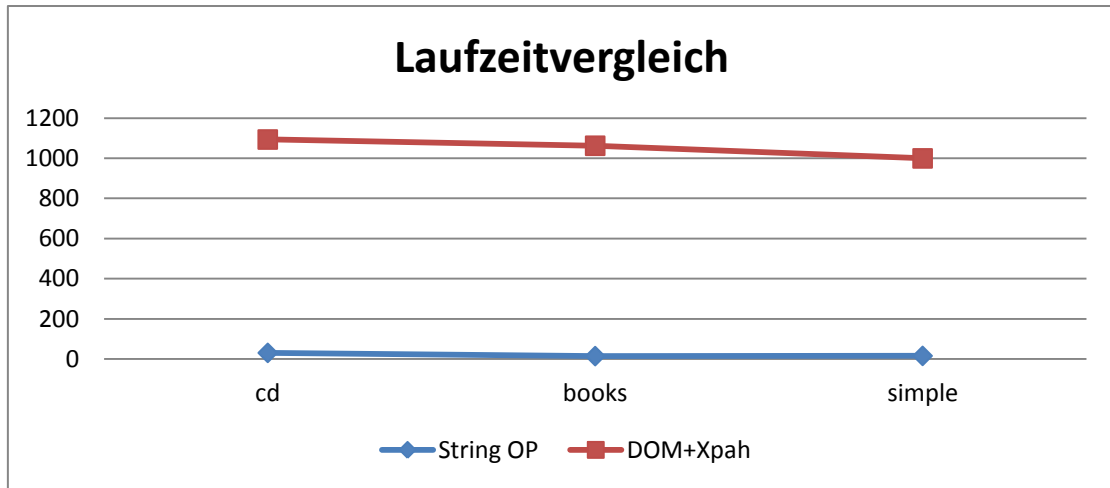


Bild 4.10 Laufzeitvergleich für Knoten mit Attributen und Werten (Einheit: ms)

Bei dem Suchverfahren benötigt String-Operation 32ms für cd.xml, 15ms für books.xml und 16ms für simple.xml. Aber DOM+XPath braucht 1094 ms für cd.xml, 1063ms für books.xml und 1000ms für simple.xml.

Das Bild 4.11 zeigt den Vergleich bei der Anforderung an Arbeitsspeicher zwischen beiden Techniken. Das verwendete XML-Dokument steht im Anhang.

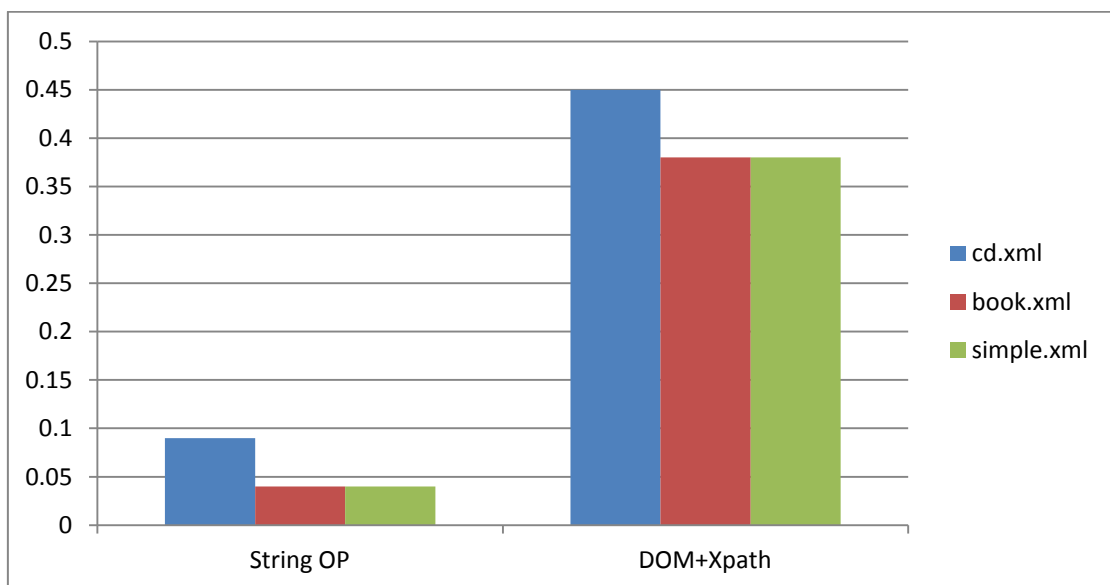


Bild 4.11 Anforderung an Arbeitsspeicher für beide Techniken (Einheit: MB)

Bei dem Test benötigt die String-Operation den Arbeitsspeicher von 0.09MB, 0.04MB und 0.04MB. Aber DOM+XPath benötigt den Arbeitsspeicher 0.45MB, 0.38MB und 0.38MB.

Bild 4.11 zeigt den Vergleich der durchschnittlichen Laufzeit zwischen DOM+XPath und String-Operation nach 10000 mal. Die verwendeten XML-Dokumente sind im Anhang.

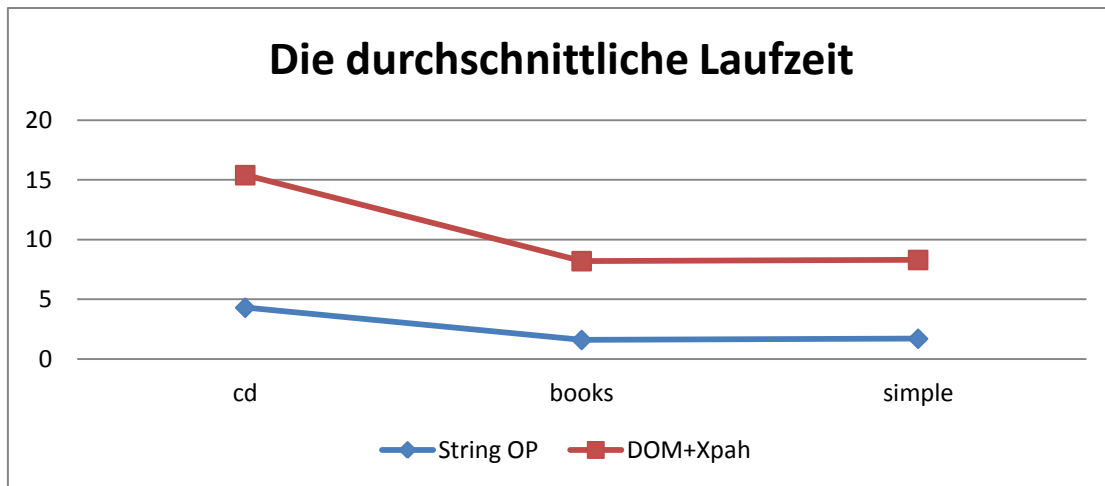


Bild 4.12 Die durchschnittliche Laufzeit nach 10000 Läufen (Einheit: ms)

Bei dem Suchverfahren benötigt String-Operation durchschnittlich 4.3ms für cd.xml, 1.6ms für books.xml und 1.7ms für simple.xml. Aber DOM+XPath braucht 15.4 ms, 8.2ms sowie 8.3ms.

4.2.6 Implementation für Suche mit Vergleichszeichen

Im Abschnitt wird das Verfahren durchgeführt, um Knoten mit Werten und Attributen zu behandeln. Das Verfahren ist die in Liste 3.19 gezeigte Impl5. Das wird benutzt, den geeigneten XPath-Ausdruck zu ersetzen.

Anhang zeigt das XML-Dokument, das als Testfall verwendet wird.

Diese Implementation besteht aus zwei Teilen. Erster Teil ist wie normal, dass das grunde XPath-Ausdruck „//“ durch oben gennante vier Situationen simuliert wird. Zweiter Teil ist der Vergleichteil. Man kann durch eine Schleife jedes Ergebnis von erstem Teil mit Kondition vergleichen, dann kann man endgültige Ergebnis bekommen.

Wir nehmen das XML-Dokument cd.xml und XPath-Ausdruck wie in Liste 4.12 gezeigt als Beispiel. Das XML-Dokument cd.xml steht im Anhang.

- /CATALOG/CD/TITLE/text()
- /CATALOG/CD[2]/TITLE/text()
- /CATALOG/CD[PRICE > 10.8]/TITLE/text()

Liste 4.11 Beispiel für XPath-Ausdruck mit Vergleichszeichen

Die Ersatztechnik wird vom in Liste 4.12 gezeigten Verfahren realisiert.

- Suche alle Knoten TITLE aus und in eine ArrayList zu speichern
- Suche alle Knoten PRICE aus und in eine ArrayList zu speichern
- Suche die Knoten PRICE aus, Seine Werte gröÙe als 10.8. und die Position in der ArrayList zu ermitteln
- ruf dann die Knoten TITLE, die mit gleicher Position in ArrayList wie die Knoten PRICE sind

Liste 4.11 Verfahren, um den XPath-Ausdruck mit Vergleichszeichen zu simulieren

Bild 4.13 zeigt den Vergleich der Laufzeit zwischen DOM+XPath und String-Operation. Die verwendeten XML-Dokumente stehen im Anhang.

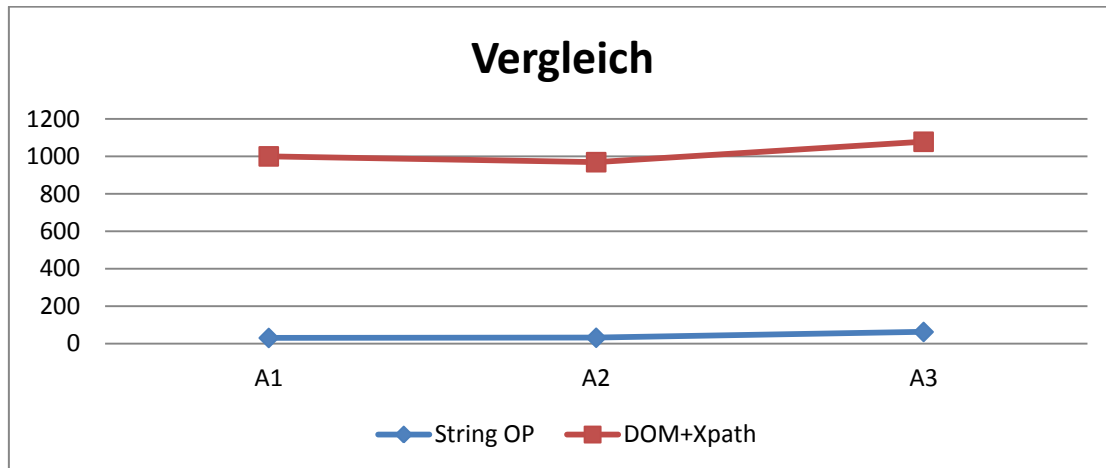


Bild 4.3 Simulation des komplizierten XPath-Ausdrucks (Einheit: ms)

Bei dem Suchverfahren benötigt String-Operation A1 31ms, A2 32ms, A3 63ms durchzuführen. Aber DOM+XPath braucht 1000ms um A1, 969ms um A2, 1078ms um A3 durchzuführen.

Durch die oben veranschaulichten Testfälle wird deutlich, dass die Ersatztechnik bzw. die pure String-Operation viel weniger Ressourcen des Rechners benötigt. Das heißt, dass die pure String-Operation weniger Rechenzeitaufwand und Arbeitsspeicherressourcen als DOM+XPath braucht.

5. Zusammenfassung und Ausblick

In dieser Diplomarbeit wird der Fokus auf die Entwicklung einer neuen Abfragetechnik für XML-Dokument gelegt, um die klassische Abfragetechnik DOM+XPath zu ersetzen. Das Ergebnis ist die reine String-Operation.

Durch den Vergleich der beiden Abfragetechniken wird klar, dass die reine String-Operation bei der Abfrage viel weniger Zeit und Arbeitsspeicher als die DOM+XPath Technik benötigt. Das heißt, dass die reine String-Operation effizienter als DOM+XPath ist. Bei einmaliger Durchführung einer Größenordnung werden beim Anwenden der String-Operation weniger Ressourcen eines Rechners verbraucht als bei DOM+XPath. Mit Erhöhung der Größe eines XML-Dokuments und der Komplexität einer Abfrage werden die angeforderte Zeit und Arbeitsspeicher nur linear erhöht.

Im Vergleich zu DOM+XPath hat die reine String-Operation als Abfragetechnik nicht nur einen Effizienzvorteil. Bei einer Abfrage kann DOM+XPath nur den Inhalt von Knoten bekommen. Beispielsweise kann für den XPath-Ausdruck `/CD/TITLE/text()` nur der Inhalt von TITLE erhalten werden. Im Gegensatz dazu kann mit der reinen String-Operation leicht der Erhalt eines kompletten Knotens mit Knotenwert und Knotenkennzeichnung realisiert werden.

In der Zukunft sollte aufbauend auf dieser Arbeit mehr Testfälle erstellt und getestet werden, damit das Algorithmus vervollständigt werden kann. Eine Methode kann probiert werden, dass eine Behandlung vorab durchgeführt, alle andere Geschwister-Knoten vom gezielte Knoten gelöscht zu werden, damit Suchungszeit verkürzt werden kann, und Fehler zu vermeiden.

In der Zukunft können aufbauend auf dieser Arbeit bessere Reguläre Ausdrücke weiter entwickelt werden. Vielleicht kann ein Regulärer Ausdruck für alle Situationen z.B. Knoten mit Attributen oder ohne Attributen entwickelt werden.

Ein anderer Bereich könnte auch sein, dass die Schnittstelle SAX benutzt wird, um entweder für XML-Dokument oder XML-Schema zu analysieren.

Anhang A cd.xml [W3C-XML]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<CATALOG>
```

```
  <CD category="COOKING">
```

```
    <TITLE>Empire Burlesque</TITLE>
```

```
      <ARTIST nr="1">Bob Dylan</ARTIST>
```

```
      <COUNTRY>USA</COUNTRY>
```

```
      <COMPANY>Columbia</COMPANY>
```

```
      <PRICE>10.90</PRICE>
```

```
      <YEAR>1989</YEAR>
```

```
  </CD>
```

```
  <CD category="COOKING">
```

```
    <TITLE>Hide your heart</TITLE>
```

```
      <ARTIST nr="2">Bonnie Tyler</ARTIST>
```

```
      <COUNTRY>UK</COUNTRY>
```

```
      <COMPANY>CBS Records</COMPANY>
```

```
      <PRICE>9.90</PRICE>
```

```
      <YEAR>1988</YEAR>
```

```
  </CD>
```

```
  <CD category="CHILDREN">
```

```
    <TITLE>Greatest Hits</TITLE>
```

```
      <ARTIST nr="3">Dolly Parton</ARTIST>
```

```
      <COUNTRY>USA</COUNTRY>
```

```
      <COMPANY>RCA</COMPANY>
```

```
      <PRICE>9.90</PRICE>
```


<YEAR>1982</YEAR>

</CD>

<CD category="WEB">

<TITLE>Still got the blues</TITLE>

<ARTIST nr="4">Gary Moore</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Virgin records</COMPANY>

<PRICE>10.20</PRICE>

<YEAR>1990</YEAR>

</CD>

<CD category="COOKING">

<TITLE>Eros</TITLE>

<ARTIST nr="5">Eros Ramazzotti</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>BMG</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1997</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>One night only</TITLE>

<ARTIST nr="6">Bee Gees</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Polydor</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1998</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Sylvias Mother</TITLE>
<ARTIST nr="7">Dr.Hook</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS</COMPANY>
<PRICE>8.10</PRICE>
<YEAR>1973</YEAR>

</CD>

<CD category="WEB">

<TITLE>Maggie May</TITLE>
<ARTIST nr="8">Rod Stewart</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Pickwick</COMPANY>
<PRICE>8.50</PRICE>
<YEAR>1990</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Romanza</TITLE>
<ARTIST nr="9">Andrea Bocelli</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Polydor</COMPANY>
<PRICE>10.80</PRICE>
<YEAR>1996</YEAR>

</CD>

<CD category="COOKING">

<TITLE>When a man loves a woman</TITLE>
<ARTIST nr="11">Percy Sledge</ARTIST>
<COUNTRY>USA</COUNTRY>

<COMPANY>Atlantic</COMPANY>

<PRICE>8.70</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Black angel</TITLE>

<ARTIST nr="12">Savage Rose</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Mega</COMPANY>

<PRICE>10.90</PRICE>

<YEAR>1995</YEAR>

</CD>

<CD category="WEB">

<TITLE>1999 Grammy Nominees</TITLE>

<ARTIST nr="13">Many</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Grammy</COMPANY>

<PRICE>10.20</PRICE>

<YEAR>1999</YEAR>

</CD>

<CD category="WEB">

<TITLE>For the good times</TITLE>

<ARTIST nr="14">Kenny Rogers</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Mucik Master</COMPANY>

<PRICE>8.70</PRICE>

<YEAR>1995</YEAR>

</CD>

<CD category="WEB">

<TITLE>Big Willie style</TITLE>

<ARTIST nr="15">Will Smith</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Columbia</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1997</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Tupelo Honey</TITLE>

<ARTIST nr="16">Van Morrison</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Polydor</COMPANY>

<PRICE>8.20</PRICE>

<YEAR>1971</YEAR>

</CD>

<CD category="COOKING">

<TITLE>The very best of</TITLE>

<ARTIST nr="17">Cat Stevens</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>Island</COMPANY>

<PRICE>8.90</PRICE>

<YEAR>1990</YEAR>

</CD>

<CD category="WEB">

<TITLE>Stop</TITLE>

<ARTIST nr="18">Sam Brown</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>A and M</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1988</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Bridge of Spies</TITLE>
<ARTIST nr="19">T'Pau</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Siren</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>

</CD>

<CD category="COOKING">

<TITLE>Private Dancer</TITLE>
<ARTIST nr="20">Tina Turner</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Capitol</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1983</YEAR>

</CD>

<CD category="WEB">

<TITLE>Midt om natten</TITLE>
<ARTIST nr="21">Kim Larsen</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Medley</COMPANY>

<PRICE>7.80</PRICE>

<YEAR>1983</YEAR>

</CD>

<CD category="WEB">

<TITLE>Pavarotti Gala Concert</TITLE>

<ARTIST nr="22">Luciano Pavarotti</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>DECCA</COMPANY>

<PRICE>9.90</PRICE>

<YEAR>1991</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>The dock of the bay</TITLE>

<ARTIST nr="23">Otis Redding</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>Atlantic</COMPANY>

<PRICE>7.90</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD category="CHILDREN">

<TITLE>Picture book</TITLE>

<ARTIST nr="24">Simply Red</ARTIST>

<COUNTRY>EU</COUNTRY>

<COMPANY>Elektra</COMPANY>

<PRICE>7.20</PRICE>

<YEAR>1985</YEAR>

</CD>

<CD category="WEB">

<TITLE>Red</TITLE>

<ARTIST nr="25">The Communards</ARTIST>

<COUNTRY>UK</COUNTRY>

<COMPANY>London</COMPANY>

<PRICE>7.80</PRICE>

<YEAR>1987</YEAR>

</CD>

<CD category="WEB">

<TITLE>Unchain my heart</TITLE>

<ARTIST nr="26">Joe Cocker</ARTIST>

<COUNTRY>USA</COUNTRY>

<COMPANY>EMI</COMPANY>

<PRICE>8.20</PRICE>

<YEAR>1987</YEAR>

</CD>

</CATALOG>

Anhang B simple.xml [XMLSpy]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v2007 (http://www.altova.com) -->
<breakfast_menu>
  <food>
    <name nr="1">Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous Belgian Waffles with plenty of real maple
      syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name nr="2">Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>light Belgian waffles covered with strawberries and whipped
      cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name nr="3">Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>light Belgian waffles covered with an assortment of fresh
      berries and whipped cream </description>
    <calories>900</calories>
  </food>
  <food>
```


<name nr="4">French Toast</name>

<price>\$4.50</price>

<description>thick slices made from our homemade sourdough bread
</description>

<calories>600</calories>

</food>

<food>

<name nr="5">Homestyle Breakfast</name>

<price>\$6.95</price>

<description>two eggs, bacon or sausage, toast, and our ever-popular hash
browns </description>

<calories>950</calories>

</food>

</breakfast_menu>

Anhang C books.xml [W3C-XML]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="children">
<TITLE lang="en">Harry Potter</TITLE>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="cooking">
<TITLE lang="en">Everyday Italian</TITLE>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="web" cover="paperback">
<TITLE lang="en">Learning XML</TITLE>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
<book category="web">
<TITLE lang="en">XQuery Kick Start</TITLE>
<author>James McGovern</author>
<author>Per Bothner</author>
```

<author>Kurt Cagle</author>

<author>James Linn</author>

<author>Vaidyanathan Nagarajan</author>

<year>2003</year>

<price>49.99</price>

</book>

</bookstore>

Literaturverzeichnis

- [AIM] Advanced Information Management, Holger Schwarz
URL: <http://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/lehreveranstaltungen/vorlesungen/SS12/aim2012>
- [Core Java 2] Core Java 2, Volume I – Fundamentals Seventh Edition, Cay S. Hostmann
Gary Cornell
- [LR00] Frank Leymann and Dieter Roller, Production Workflow. Concepts and
Techniques, Prentice-Hall, 2000
- [W3C-DOM] DOM, W3C URL: <http://www.w3schools.com/dom>
Verifizierung am 19.02.2013
- [W3C-Schema] Schema, W3C URL: <http://www.w3schools.com/schema>
Verifizierung am 24.02.2013
- [W3C-XML] XML, W3C URL: <http://www.w3schools.com/xml>
Verifizierung am 19.02.2013
- [W3C-XPath] XPath, W3C URL: <http://www.w3schools.com/xpath>
Verifizierung am 19.02.2013
- [WiKi] XPath, WIKIPEDIA URL: <http://de.wikipedia.org/wiki/XPath>
Verifizierung am 24.02.2013

[XMLBeans] Welcome to XMLBeans, Apache URL: <http://xmlbeans.apache.org>
Verifizierung am 24.02.2013

[XMLSpy] XMLSpy, Altova URL: <http://www.altova.com/xmlspy>
Verifizierung am 24.02.2013

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 05. März 2013 _____