

**XML**  
**Anfrage- und**  
**Transformationsprachen**

VIROR Tele-Blockseminar im SS 2001

**QUILT**

**Heiko Oberdiek**

**Christian Weiß**

27. Juli 2001

Universität Karlsruhe

AIFB

Prof. Stucky

M. Klein

D. Sommer

Universität Freiburg

Prof. Lausen

M. Ahmedi

Prof. Thiemann

J. Walter

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Anforderungen</b>	<b>3</b>
2.1	Anforderungen an das Datenmodell . . . . .	3
2.2	Generelle Anforderungen nach Heuer und Saake . . . . .	3
2.3	Generelle Anforderungen nach dem W3C . . . . .	5
<b>3</b>	<b>Sprachelemente von Quilt</b>	<b>7</b>
3.1	XPath . . . . .	7
3.2	Element-Konstrukturen . . . . .	7
3.3	Bedingte Ausdrücke . . . . .	7
3.4	FLWR-Ausdrücke . . . . .	8
3.5	Variablen-Bindungen . . . . .	8
3.6	Operatoren . . . . .	8
3.7	Funktionen . . . . .	8
3.8	Quantifizierer . . . . .	8
<b>4</b>	<b>Ansätze via SQL</b>	<b>9</b>
4.1	Quilt2Sql – 544project . . . . .	9
4.2	Das Agora-Projekt . . . . .	9
4.3	Das XPRESS-Projekt . . . . .	9
4.4	Zusammenfassung . . . . .	10
<b>5</b>	<b>Implementierung Kweelt am Beispiel Miete</b>	<b>11</b>
5.1	Kweelt . . . . .	11
5.2	Miete . . . . .	11
5.3	Eingabedaten . . . . .	11
5.4	Quilt-Anfrage . . . . .	14
5.4.1	Pfadausdrücke . . . . .	15
5.4.2	Funktionen . . . . .	16
5.4.3	Java-Funktionen . . . . .	16
5.4.4	Encoding . . . . .	17
5.5	Ausgabe . . . . .	17
5.5.1	TEX-Zwischenstufe . . . . .	17
5.5.2	Wandlung nach PDF . . . . .	18
<b>6</b>	<b>Diskussion Kweelt/Quilt</b>	<b>19</b>
6.1	Abweichungen von Kweelt . . . . .	19
6.2	Benutzerfreundlichkeit von Kweelt . . . . .	19
6.3	Vor- und Nachteile von Quilt . . . . .	19
6.4	Zukunft von Quilt . . . . .	20

<b>7</b>	<b>Anhang</b>	<b>21</b>
7.1	DTD-Dateien . . . . .	21
7.1.1	Mietkosten.dtd . . . . .	21
7.1.2	Miete.dtd . . . . .	21
7.2	Ausgabe-Dateien . . . . .	21
7.2.1	Rechnung.xml . . . . .	21
7.2.2	Rechnung.tex . . . . .	22
7.2.3	TeXOutputHandler.java . . . . .	22
	<b>Literatur</b>	<b>25</b>

## Abbildungsverzeichnis

1	Agora-Übersicht . . . . .	10
2	Allgemeiner Ansatz . . . . .	12
3	Workflow-Prozess für das Beispiel Miete . . . . .	12
4	Auszug aus der Datei Mietkosten.xml . . . . .	13
5	Auszug aus der Datei Miete.xml . . . . .	13
6	Eingabe-GUI zur monatlichen Dateneingabe . . . . .	14
7	Java-Funktions-Beispiel . . . . .	17
8	Die fertige Rechnung als Beispiel . . . . .	18

# 1 Einleitung

XML etabliert sich als Standard für die Speicherung von Informationen aller Art in der vernetzten Datenwelt. Ganz unterschiedliche Dokumente lassen sich mit XML flexibel strukturieren und so für jede beliebige Anwendung anpassen. Diese Flexibilität erhalten XML-Daten als semistrukturierte Dokumente durch die stets im Dokument selbst vorhandene Strukturinformationen. Deshalb sind sie für den elektronischen Datenaustausch (EDI: Electronic Data Interchange) ebenso geeignet wie für das automatische Bearbeiten und Transformieren von Dokumenten sowie vielfältigen anderen Applikationen.

Doch Sammlung und Speicherung von Information allein reichen für das erfolgreiche Informationsmanagement nicht aus. Mindestens genau so wichtig ist das Wiederauffinden bestimmter Daten. Das klassische Problem ist allseits bekannt: Im Internet gibt es zu (fast) jedem Thema viel Information. Doch sehr selten wird genau die Information gefunden, die gesucht wird. Dabei ist es unerheblich, welche der vielen Suchmaschinen genutzt wird. Auch die Literaturrecherche zu dieser Arbeit zeigte wieder, dass die Recherchestrategie eine hohe Kunst und die Qualität des Suchergebnisses keinesfalls sicher ist. Viel zu gering sind die Möglichkeiten, das Gesuchte genau zu spezifizieren.

Im Datenbankbereich wurden zu diesem Zweck die Anfragesprachen geschaffen. Heute ein gut erforschtes Gebiet bildend, stellen sie ihre Nützlichkeit täglich unter Beweis. Die mit Abstand bekanntesten Vertreter sind SQL (für die relationalen Datenbanksysteme) und OQL (für die objektorientierten Datenbanksysteme).

Diese Anfragesprachen arbeiten vorwiegend mit der Schemainformation der Datenbank. Sie sind kaum auf semi-strukturierte Daten wie XML anwendbar. Denn dort sind prinzipbedingt solche zentral gespeicherten Meta-Informationen nicht vorhanden. Auch die hohe Flexibilität des internen Aufbaus von XML-Dokumenten steht einer Übernahme der bestehenden Datenbankanfragesprachen entgegen. Zudem sind Anfragesprachen nur unzureichend für die Suche in hierarchischen Strukturen gerüstet. Die Information-Retrieval (IR) Funktionalität ist ebenfalls wenig bzw. gar nicht ausgeprägt.

Daher ist eine neue Generation von Anfragesprachen zu schaffen. Verschiedene Vorschläge und Prototypen existieren. In dieser Arbeit wird die Anfragesprache Quilt vorgestellt.

Dabei wird nach einer kurzen Einleitung zu Entstehung und Hintergrund von Quilt die Anforderungen an eine solche Sprache dargestellt, sowie die jeweilige Umsetzung in Quilt diskutiert. Daran schließt sich die Vorstellung der Sprachelemente und der Sprachdefinitionen an. Das darauf folgende Anwendungsbeispiel veranschaulicht die Umsetzung. Außerdem soll neben einer Schlussbetrachtung noch ein Ausblick in die Zukunft der Sprache Quilt gegeben werden.

Die XML-Anfragesprache Quilt wurde von den drei Autoren Jonathan Robie (Software AG), Don Chamberlin (IBM Almaden Research Center)

und Daniela Florescu (INRIA) verfasst. Diese stellten fest, dass alle bisherigen XML-Anfragesprachen-Kandidaten in einigen Bereichen ihre Stärken beweisen, für andere Anwendungsfelder jedoch weniger gut geeignet sind. Sie setzten sich den Ausspruch Igor Stravinskys „Mediocre composers borrow, great composers steal“ zum Motto und versuchten, die Stärken verschiedener Anfragesprachen zu kombinieren. Gestützt auf die Architektur von XML achteten sie jedoch gleichzeitig auf konzeptionelle Integrität.

So flossen u. a. aus XPath die Auswahl von Knotenmengen, aus XQL das Navigieren in der Struktur von XML-Dokumenten und die komplexen Knotenauswahlbedingungen sowie aus XML-QL das Binden von Variablen und die Konstruktion des Ergebnisses ein. Quilt ist somit gleichermaßen gut geeignet für die Suche in flachen (also relationalen) Hierarchien als auch hierarchischen Datenstrukturen; auch die Konstruktion eines hierarchischen Dokumentes aus einem DB-View stellt keine Schwierigkeit dar. Sogar die Definition von eigenen Funktionen ist möglich. Die Sprache beherrscht die Gruppierung sowie das Ergebnis-Design!

## 2 Anforderungen

### 2.1 Anforderungen an das Datenmodell

An dieser Stelle sollen die Anforderungen an eine XML-Anfragesprache wie Quilt dargestellt werden, wie sie vom W3C in „XML query requirements“ ([1]) gefordert werden. Außerdem haben Angela Bonifati und Stefano Ceri ([2]) einige Anforderungen definiert. In beiden Quellen werden spezielle Anforderungen an das Datenmodell spezifiziert.

So soll sich das XML-Anfragesprachen-Datenmodell ausschließlich auf die Information stützen, die es von den XML- und Schema-Prozessoren erhält. Es muss sichergestellt werden, dass keine weiteren Informationen benötigt werden. Alle Konstrukte sollen in eigenen Datenmodell-Konstrukten abgebildet werden. Für Quilt werden Element-, Attribut- und Wert-Datenkonstrukte beschrieben.

Es müssen sowohl XML 1.0 „character data“ als auch einfache und komplexe Datentypen der XML-Schema-Spezifikation darstellbar sein. Unter „character data“ versteht man unter XML 1.0 alle Textdaten (grundsätzlich vom Typ „string“) eines XML-Dokumentes mit Ausnahme der Metainformation (tags, Deklarationen, „processing instructions“, Entitäten, Kommentare, etc. siehe auch [3]). Quilt unterstützt sowohl den Datentyp „string“ als auch „ID“ und „IDREF“.

Sowohl Sammlungen von Dokumenten als auch von einfachen und komplexen Werten müssen darstellbar sein. Sammlungen werden auch als Multi-Mengen bezeichnet, d. h. Mengen, in denen Objekte auch mehrmals auftreten können. Diese Sammlung von Werten wird von Quilt geboten. Jedoch wird in der Literatur nichts zur Sammlung von Dokumenten erwähnt.

Das XML-Anfragesprachen-Datenmodell muss dokumentinterne sowie externe Referenzen unterstützen. Die Literatur zu Quilt geht auf Referenzen als besondere Objekte ein. Die Frage zu externen Referenzen bleibt allerdings offen.

Eine Anfrage muss möglich sein, unabhängig davon, ob ein Schema (XML-Schema bzw. DTD) vorliegt oder nicht. Falls ein solches vorhanden ist, muss das Datenmodell sämtliche – für die Instanzen definierte – Konstrukte darstellen. Leider geht auch hier die Literatur weder darauf ein, ob XML-Dokumente ohne Schema möglich sind als auch, ob alle (z. B. in einer DTD) vordefinierten Konstrukte dargestellt werden.

Genau so wenig wird in der Literatur über die geforderte Beachtung von Namespaces ausgesagt.

### 2.2 Generelle Anforderungen nach Heuer und Saake

Nun werden die Umsetzungen der generellen Anforderungen betrachtet, die Andreas Heuer und Gunter Saake in [4] erhoben haben.

**Ad-hoc-Formulierung:** Auch bei XML-Anfragesprachen liegt ein wichtiger Punkt darin, schnell, einfach und flexibel Anfragen stellen zu

können. Ein komplettes Programm wäre viel zu aufwändig, ein kompakter einzelner Befehl scheint angemessener. Dies wird auch deutlich, wenn man die typischen Anwendungen betrachtet: Meist werden XML-Anfragen eingebettet (in XML-Dokumenten, EDI- oder sonstigen Programmen) bzw. interaktiv genutzt. Bei Quilt reicht in der Regel die Formulierung einer einzelnen Anweisung; eine bis fünf Zeilen sollten ausreichen. Für komplexere Anfragen bzw. für eine bessere Übersichtlichkeit sollte auf eine Verschachtelung mehrerer Instanzen dieser Anweisung zurückgegriffen werden. Das spätere Beispiel zeigt auch, dass mit relativ wenig Schreibaufwand und einer übersichtlichen Syntax die gesuchten Dokumentteile spezifiziert werden können.

**Deskriptivität:** Der Nutzer soll genau und komfortabel beschreiben können, was er sucht. Dagegen soll es nicht notwendig sein, dass er weiß, wie man zu diesen Informationen kommt. Das XML-Anfragesystem muss selbst in der Lage sein, entsprechend den ihm vorgelegten Dokumentbeständen, seinen eigenen Regeln und Optimierungsalgorithmen zu jeder syntaktisch korrekt gestellten Anfrage einen Anfrageausführungsplan zu generieren, der zu der gesuchten Information führt. Auch diese Forderung erfüllt Quilt. Bei dieser deskriptiven Sprache steht die Beschreibung der gewünschten Eigenschaften im Vordergrund und nicht die Realisierungsart der Abfrage.

**Mengenorientiertheit:** Auch bei XML soll parallel auf den Dokumentbeständen gearbeitet werden. Im Gegensatz zu relationalen Datenbankmanagementsystemen, die auf reinen Mengen arbeiten, ist jedoch auch meist die Reihenfolge der Daten und Dokumentelemente wichtig. So stellt dies bei XML-Anfragesprachen naturgemäß ein größeres Problem dar. Hier liegt ein entscheidender Vorteil von Quilt! Die Autoren von Quilt unterstreichen in [5] die Unterteilung in drei Bearbeitungsschritte, zwischen denen Tupel-Mengen übergeben werden. In der `FOR`- bzw. `LET`-Klausel werden die gesuchten XML-Dokument-Teilstücke und ihre Herkunft spezifiziert. Je nachdem, ob das Ergebnis als einzelne Tupel oder als Menge von Tupeln betrachtet werden soll, wird `FOR` oder `LET` als Schlüsselwort gewählt. Die `WHERE`-Klausel schränkt die Zwischenergebnismenge weiter ein und die `RETURN`-Klausel ermöglicht die Konstruktion des Ergebnisses. Es kann also von einer konsequenten Mengenorientiertheit ausgegangen werden.

**Abgeschlossenheit:** Bei XML-Anfragesprachen muss das Ergebnis wieder ein wohlgeformtes XML-Dokument ergeben. Das Ergebnis einer Quilt-Anfrage ist zwar nicht implizit ein gültiges XML-Dokument, jedoch ist ein solches durch das simple Hinzufügen eines umschließenden Dokument-Root-Elementes zu erreichen.

**Adäquatheit:** Alle Konstrukte des zugrunde liegenden Datenmodells müssen unterstützt werden. Das heißt, dass nicht nur Anfragen auf Elementen sondern auch auf ihre Attribute, Struktur, Reihenfolge und Hierar-

chie möglich sein müssen. Es wurde bereits schon vorne erwähnt, dass Quilt diese Funktionalität leistet. Des weiteren bestehen Forderungen nach

**Orthogonalität:** Alle Operatoren müssen beliebig kombinierbar sein; einzige Voraussetzung: die erwarteten Datenstrukturen müssen eingehalten werden.

**Optimierbarkeit:** Die XML-Anfrageoperatoren müssen definiert werden und es müssen Optimierungsregeln gefunden werden. Effizienz: Die Komplexität aller XML-Anfrageoperatoren muss gering gehalten werden. Nur so könne akzeptable Antwortzeiten auch auf große Datenmengen garantiert werden. Um die Bedeutung in bezug auf die Datenmenge zu unterstreichen denke man nur einmal daran, wie enorm viele Dokumente heute schon in HTML vorliegen.

**Sicherheit:** Keine Anfrage, die syntaktisch korrekt ist darf in eine Endlosschleife geraten bzw. ein unendliches Ergebnis liefern.

**Eingeschränktheit:** Aus Sicherheit, Optimierbarkeit und Effizienz folgt, dass auch eine XML-Anfragesprache niemals die Mächtigkeit einer kompletten Programmiersprache aufweisen kann.

Diese Umsetzung dieser letztgenannten Anforderungen ist zu diesem Entwicklungs- und Erforschungsstandpunkt schwer abzuschätzen und nachprüfbar. Selbstverständlich müssen diese Anforderungen aber den Entwicklern und Implementierern mit auf den Weg gegeben werden.

### 2.3 Generelle Anforderungen nach dem W3C

Auch das W3C hat in [1] noch weitere generelle Anforderungen formuliert:

**Syntax:** Die Syntax soll bequem zu lesen und zu schreiben sein. Außerdem soll eine XML-Anfrage auch in XML ausdrückbar sein und zwar in der Form, welche die Struktur widerspiegelt. Wie das später folgende Beispiel zeigt, stellt Quilt eine leicht zu lesende Syntax zur Verfügung, die jedoch nicht in XML dargestellt wird.

**Struktur-Erhaltung:** Bei XML-Dokumenten spielen Hierarchie und Sequenz eine große Rolle. So ist es oft von elementarer Wichtigkeit, die Originalstrukturen zu erhalten. Quilt bietet die Möglichkeit sowohl Teilstrukturen als auch Reihenfolgen zu erhalten.

**Anpassung/Erweiterung/Universalität:** Zu dieser Anforderung gehören sowohl die Unterstützung abstrakter Datentypen als auch die Möglichkeit eigene Funktionen zu definieren. Diese Definition eigener Funktionen ist in Quilt möglich. Abstrakte Datentypen werden jedoch nicht unterstützt.

**Fehlerzustände:** Eine XML-Anfragesprache muss Standard-Fehlerfälle definieren, die bei der Anfragebearbeitung auftreten können. Dazu gehören z. B. Fehler bei der Bearbeitung von Ausdrücken, Nicht-Verfügbarkeit von externen Funktionen, von externen Funktionen generierte Bearbeitungsfehler etc. Diese Standard-Fehlerzustände werden von Quilt allerdings nicht definiert.

**Protokoll-Unabhängigkeit und Updatefähigkeit:** Diese beiden Forderungen werden sich in Zukunft beweisen müssen. Aber generell spricht in Quilt nichts gegen eine Unabhängigkeit von Protokollen. Die Updatefähigkeit wird sich zeigen, wenn ein Nachfolger von XML 1.0 auf den Markt kommt.

## 3 Sprachelemente von Quilt

Wie schon der Name Quilt (Steppdecke) andeuten soll, vereinigt die Anfragesprache Quilt viele Konzepte anderer Sprachen unter einem Dach.

### 3.1 XPath

Aus der XPath-Spezifikation ([9]) wurde das Datenmodell abgeleitet. Dabei wird ein XML-Dokument als Baum von Noden, mehrere Dokumente oder Fragmente als geordneten Wald angesehen. XPath erlaubt dann eine Navigation durch diese Strukturen.

Ein Weg wird dabei in Einzelschritte aufgeteilt. Diese wiederum bestehen aus:

```
Achse::Test [Bedingung]
```

Die *Achse* legt dabei die Navigationsrichtung fest. Es gibt *child*, *parent*, *following*, *attribute*, ...

Der *Test* kann aus einem Nodennamen bestehen, oder aus einer Nodentypangabe wie `node()`, `comment()` oder ...

Optional können ein oder mehrere Bedingungen angegeben werden, die nacheinander überprüft werden und somit die Auswahl zusätzlich einschränken. Beispiel:

```
[attribute::foo="bar"]
```

Mit *attribute* ist wieder eine Achse angegeben, es werden also nur die Elemente betrachtet, die ein Attribut *foo* mit Wert *bar* besitzen.

Abschnitt 5.4.1 wird weitere XPath-Beispiele enthalten.

### 3.2 Element-Konstruktoren

Ausgabe von XML-Anfragen sollen wieder im XML-Format sein. Element-Konstruktoren helfen dabei:

```
<NeuesElement EinAttribut="Wert">  
Text oder weitere Elemente:  
<${EingebettetesElement}/>  
</NeuesElement>
```

Wie man am eingebetteten Element sieht, können die Namen auch über Variablen zugewiesen werden.

### 3.3 Bedingte Ausdrücke

An bedingten Ausdrücken werden nur klassische If-Then-Else-Verzweigungen unterstützt.

```
IF $var = "foo"  
THEN <ja/>  
ELSE <nein/>
```

Da sowohl `THEN` als auch `ELSE` angegeben werden müssen, gibt es keine Interpretationsprobleme bei Verschachtelungen.

### 3.4 FLWR-Ausdrücke

Ein mächtiges Feature sind die `FOR-LET-WHERE-RETURN`-Ausdrücke, kurz `FLWR`-Ausdrücke (gesprochen wie das englische „flower“). Sie weisen eine große Verwandtschaft zu den `SELECT-FROM-WHERE`-Konstrukten aus `SQL` auf. Das `SELECT` korrespondiert mit dem `RETURN`, das `FROM` mit `FOR` und `LET`. Am engsten entsprechen sich die gleichnamigen `WHERE`-Teile.

Evaluierungen über `FOR` iterieren dabei über eine Menge von Knoten, bei `LET`-Zuweisungen erfolgt die Zuordnung zu Variablen auf einmal, so dass eine Variable auch eine Menge von Knoten enthalten kann. Mit `WHERE` werden einschränkende Bedingungen angegeben und über `RETURN` der Ergebnisausdruck geformt.

Der Abschnitt 5.4 mit einer praktischen Anfrage enthält einige Beispiele, ebenso die Quilt-Beschreibung in [10].

### 3.5 Variablen-Bindungen

`LET`-Ausdrücke sind auch außerhalb von `FLWR`-Konstrukten erlaubt. Deren Evaluierung wird dabei explizit mit `EVAL` angestoßen.

### 3.6 Operatoren

Die klassischen logischen und arithmetischen Operatoren stehen zur Verfügung. Aus `SQL` wurden die Mengenoperatoren `UNION` (Vereinigung), `INTERSECT` (Schnitt) und `EXCEPT` (Minus) aufgenommen. Aus `XQL` stammen `BEFORE` und `AFTER`.

Eine Spezialität ist der `FILTER`-Operator, der aus einem Baum gewünschte Nodearten extrahiert. Dabei bleibt die relative Zuordnung dieser Knoten erhalten, so dass man einen geordneten Wald erhält. Auf diese Weise kann man beispielsweise ein Inhaltsverzeichnis aus einem Dokument erstellen.

### 3.7 Funktionen

Es werden die von `SQL` bekannten Aggregatfunktionen unterstützt. Es gibt eine Reihe eingebauter Funktionen. Zusätzlich kann man selber Funktionen definieren. Rekursion ist dabei erlaubt.

Außerdem können auch Funktionen hinzugefügt werden, die in anderen Programmiersprachen geschrieben sind.

### 3.8 Quantifizierer

Es werden der Existenzquantor `SOME` und der Allquantor `EVERY` unterstützt. Negation ist nicht erlaubt und geht so den semantischen Problemen aus dem Weg, die eine Negation zur Folge hat.

## 4 Ansätze via SQL

An Implementierungen listet die Quilt-Beschreibungsseite ([11]) deren vier auf, von denen drei den Weg über SQL beschreiten. SQL ist weit verbreitet, gut erforscht und es gibt zahlreiche Implementierungen. Wichtig ist auch, dass Anfragen in eine Algebra abgebildet und dort sehr gut optimiert werden können, um die Antwortzeiten klein zu halten.

Im Prinzip verwenden alle drei Gruppen, die den SQL-Weg beschreiten, das gleiche Schema:

1. Die Quilt-Anfrage wird nach SQL übersetzt.
2. Dort, in SQL, wird die Anfrage mit bekannter Technik ausgeführt.
3. Das Ergebnis wird schließlich nach XML zurücktransformiert.

Unterschiede gibt es vor allem in der Datenspeicherung und den unterstützten Features.

### 4.1 Quilt2Sql – 544project

Die Gruppe `544project` ([12] und [13]) sieht das XML-Speicherproblem als ungelöst an. Sie verwendet dabei relationale Tabellen, erlaubt aber keine XML-Attribute oder -Referenzen.

Die Rekursivität von Funktionen wird nicht unterstützt, die Funktionsunterstützung selbst ist eingeschränkt.

### 4.2 Das Agora-Projekt

Die Vorgehensweise des Agora-Projekts ([14]) ist differenzierter. Zuerst wird die Anfrage normalisiert, um sie dann leicht nach SQL konvertieren zu können. Als Datenmodell dient hierbei ein virtuelles allgemeines Schema. Gemäß dem realen Speicher-Schema wird die Anfrage umgeschrieben, optimiert und ausgeführt.

Dabei erlaubt eine zusätzliche „Middleware“-Schicht „LeSelect“ die Anbindung verschiedenster Speicherarten, wie relationale Datenbanken, DOM-Bäume oder XML-Dateien. Die „LeSelect“-Schicht übernimmt dabei die jeweilige Transformation nach SQL und zurück.

Das erhaltene Ergebnis wird mit Informationen aus der virtuellen Schicht zum gewünschten XML-Ergebnisformat umgewandelt.

Abbildung 1 enthält die Vorgehensweise nochmal in graphischer Form.

### 4.3 Das XPRESS-Projekt

Im XPRESS-Projekt (Xml Processing and Relaxation in rELational Storage System, [15] und [16]) wird wie bei der `544project`-Gruppe die Daten in einem relationalen Modell abgelegt, die Anfrage nach SQL gewandelt, dort ausgeführt und das Ergebnis wieder nach XML transferiert.

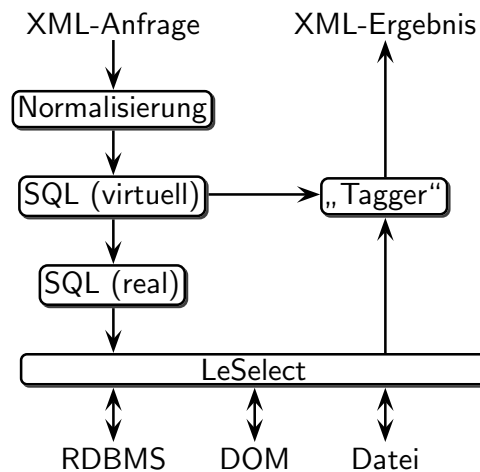


Abbildung 1: Agora-Übersicht

Für die Allgemeinheit verfügbar ist hier ein `QuiltParser`, geschrieben in Java. Damit können Quilt-Anfragen auf syntaktische Korrektheit überprüft werden.

#### 4.4 Zusammenfassung

Die großen Vorteile des Weges über SQL sind zum einen die bekannten und guten Optimierungsmöglichkeiten im Bereich SQL. Zum anderen stehen bereits zahlreiche relationale Datenbanksysteme zur Verfügung.

Nachteilhaft sind die zusätzlich notwendigen Konvertierungen nach SQL und zurück zu XML. Durch die SQL-Anfrageoptimierung wird das aber wieder wettgemacht. Als Hauptnachteil bleiben aber Informationsverluste bei der Konvertierung von XML-Dateien (DOM-Struktur) zum tabellenorientierten relationalen Modell, das einer SQL-Anfrage zugrunde liegt. Dies begründet die Haupteinschränkung des Weges der Evaluierung einer Quilt-Anfrage über SQL.

## 5 Implementierung Kweelt am Beispiel Miete

Für die Allgemeinheit verfügbar ist die Implementierung Kweelt, so dass im Rahmen dieses Seminars ein praktisches Beispiel erstellt werden konnte. Dabei wurde eine Anwendung ausgewählt, die bisher auf einem Uralt-DOS-Tabellenkalkulationsprogramm lief, und zwar die Erstellung einer Mietrechnung, monatlich benötigt von einem der Autoren. Die Daten werden dabei nun portabel in XML-Dateien gehalten, die Auswahl der benötigten Daten erfolgt über die Anfragesprache Quilt. So soll dieses praktische Beispiel zum einen mal den gesamten XML-basierten Prozess von Ein- bis -ausgabe zeigen, zum anderen, wie Quilt, bzw. die Implementierung Kweelt sich darin nutzbringend einbettet.

### 5.1 Kweelt

Zunächst aber ein paar Worte zum verwendeten Kweelt. Das Ziel des Kweelt-Projekts ([17]) ist die Implementierung von Quilt in Java. Sowohl der Sourcecode, als auch eine Jar-Datei wurden im Rahmen dieses Projektes veröffentlicht. Konzipiert wurde Kweelt dabei als Referenzimplementierung. So soll es als Basis für weitere Forschungen, Verbesserungen und Erweiterungen dienen. Schwerpunkte wurden daher auf Vollständigkeit und der Erprobung des Sprachvorschlages Quilt in der Praxis, jedoch noch nicht auf Optimalität und schnelle Antwortzeiten gelegt.

### 5.2 Miete

Zurück zum praktischen Beispiel. Abbildung 2 illustriert den dahinterstehenden allgemeinen Ansatz.

Die Eingaben und Daten werden dabei in XML-Dateien gespeichert. Eine Quilt-Anfrage extrahiert die gewünschten Informationen und liefert das Ergebnis in Form einer XML-Ausgabedatei. Diese wird nun um die Layoutinformationen ergänzt in das gewünschte Format des Endergebnisses konvertiert.

Abbildung 3 konkretisiert diesen allgemeinen Ansatz für das hier entwickelte Beispiel Miete.

Die einzelnen Elemente aus dieser Abbildung 3 werden nun in den einzelnen Unterabschnitten besprochen.

### 5.3 Eingabedaten

An Eingabedaten gibt es zum einen die relativ statischen Grunddaten, wie Grundmiete, Grundgebühren. Zum anderen fallen jeden Monat der Stromzählerstand, sowie die jeweils vertelefontierten Einheiten an. Gespeichert werden diese Daten im ersten Fall in der Datei `Mietkosten.xml`, im letzteren in `Miete.xml`. Diese Dateien werden in den Abbildungen 4, sowie 5 auszugsweise gezeigt (aus Datenschutzgründen sind die Zahlen verändert.)

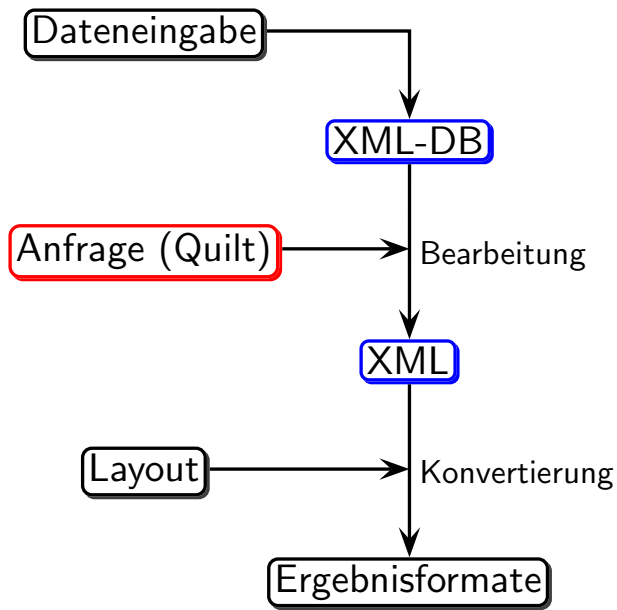


Abbildung 2: Allgemeiner Ansatz

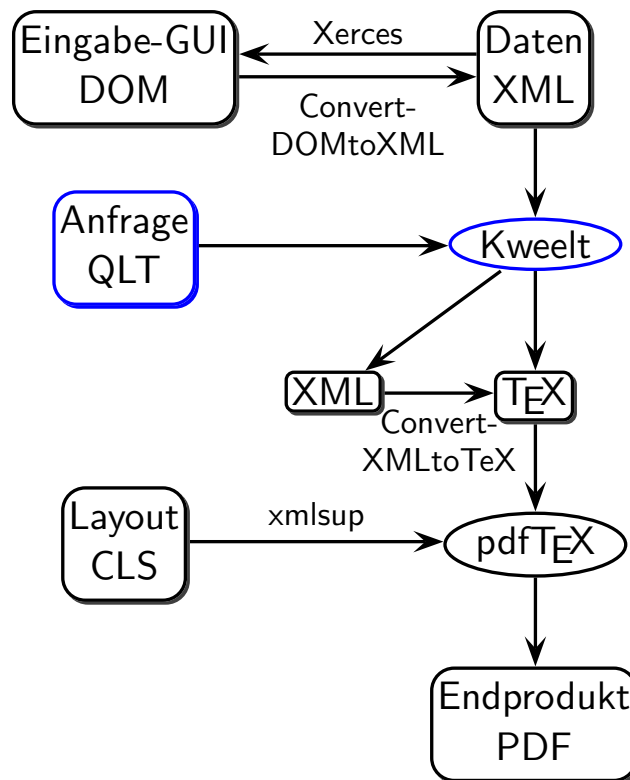


Abbildung 3: Workflow-Prozess für das Beispiel Miete

```

<?xml version="1.0"?>
<!DOCTYPE Mietkosten SYSTEM
  "Mietkosten.dtd">
<Mietkosten>
<Grundmiete>
  <Eintrag Datum="1999/12/28"
    Preis="300"/>
</Grundmiete>
...
<Telefoneinheit>
  <Eintrag Datum="1999/12/28"
    Preis="0.30"/>
...
  <Eintrag Datum="2001/04/30"
    Preis="0.121"/>
</Telefoneinheit>
</Mietkosten>

```

Abbildung 4: Auszug aus der Datei Mietkosten.xml

```

<?xml version="1.0"?>
<!DOCTYPE Miete SYSTEM
  "Miete.dtd">
<Miete>
<Strom Datum="1999/12/28"
  Stromstand="1388"/>
<Monat Datum="2000/01/31"
  Monat="2000/02"
  Stromstand="1487" .../>
<Monat ... />
...
</Miete>

```

Abbildung 5: Auszug aus der Datei Miete.xml

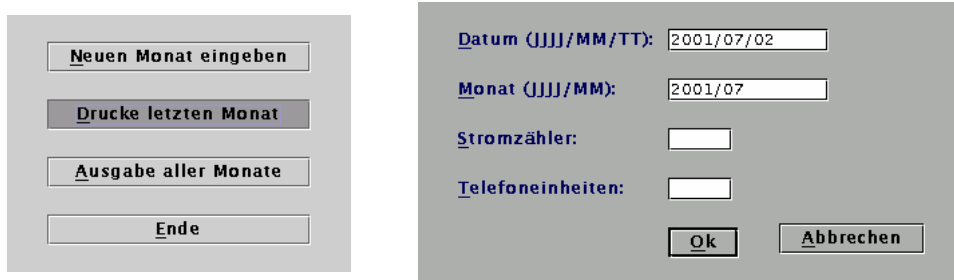


Abbildung 6: Eingabe-GUI zur monatlichen Dateneingabe

Zur komfortablen Eingabe der monatlichen Daten wurde ein einfaches Java-GUI geschrieben, das auf den Swing-Klassen basiert (siehe Abbildung 6).

Mit Hilfe von Xerces, einem in Java geschriebenen XML-Parser der Apache Software Foundation, werden die monatlichen Daten (*Miete.xml*, siehe Abb. 5) in einen DOM-Baum geparkt und mittels dem aus Beispieldateien selbstentwickelten Programm *ConvertDOMtoXML* wieder in die XML-Datei zurückgeschrieben.

#### 5.4 Quilt-Anfrage

Der Kern des gesamten Prozesses bildet die Quilt-Anfrage. Beispielsweise möchte man die letzte Monatsrechnung erhalten. In folgender Anfrage wird dazu in dem inneren FLWR-Ausdruck die benötigten Daten gesammelt und als Element *Mietrechnung* dem äußerem FLWR-Ausdruck übergeben. Hier wird nur noch summiert und auf Pfennige gerundet:

```

FUNCTION getPreis($e, $d) {
  ( FOR $x in $e/Eintrag
    [@Datum .<= . $d]
    RETURN $x
    SORTBY(@Datum DESCENDING)
  )[1]/@Preis
}
FUNCTION roundPreis($x) {
  (($x + "0.005") * 100 - ((($x + "0.005") * 100) % 1)) div 100
}
<Mietrechnung>
FOR $a in
  FOR $m in document("Miete.xml")/Monat[position() = last()]
  LET $x := document("Mietkosten.xml")
  LET $d := $m/@Datum
  LET $v := $m/preceding-sibling::node()
    [name() = "Monat" or name() = "Strom"][1]
    /@Stromstand
  LET $s := $m/@Stromstand - $v

```

```

LET $te := getPreis($x/Telefoneinheit, $d)
LET $se := getPreis($x/Stromeinheit, $d)
RETURN
  <Mietrechnung
    Datum = $d
    Monat = $m/@Monat
    Grundmiete = getPreis($x/Grundmiete, $d)
    Nebenkosten = getPreis($x/Nebenkosten, $d)
    Telefongebuehr = getPreis($x/Telefongebuehr, $d)
    Telefoneinheiten = $m/@Telefoneinheiten
    Telefoneinheit = $te
    Telefonsumme = $m/@Telefoneinheiten * $te
    Stromstand = $m/@Stromstand
    Stromeinheiten = $s
    Stromeinheit = $se
    Stromssumme = $s * $se
  />
RETURN
  <Rechnung
    Telefonsumme = roundPreis($a/@Telefonsumme)
    Stromssumme = roundPreis($a/@Stromssumme)
    $a/@*[name() != "Telefonsumme" and
      name() != "Stromssumme"]
    Gesamtsumme = roundPreis($a/@Grundmiete
      + $a/@Nebenkosten
      + $a/@Telefongebuehr
      + $a/@Telefonsumme
      + $a/@Stromssumme)
  />
</Mietrechnung>

```

Im folgenden werden nun einzelne Elemente aus der Quilt-Anfrage besprochen.

#### 5.4.1 Pfadausdrücke

```
document("Miete.xml")/Monat[position() = last()]
```

Der erste Schritt `document("Miete.xml")` des obigen XPath-Ausdrucks liefert die Root-Node des Dokuments `Miete.xml`. Der nächste Schritt beschränkt die Suche auf direkte `Monat`-Elementkinder, von denen jedoch durch die Zusatzbedingung `[position() = last()]` alle bis auf das letzte wieder verworfen werden. In XPath selbst kann die Bedingung zu `[last()]` abgekürzt werden. Dies wird aber nicht von Kweelt unterstützt.

Das gefundene Element `Monat` wird dann in der inneren `FOR`-Anweisung der Variablen `$m` zugewiesen, die im nächsten XPath-Ausdruck einer `LET`-Anweisung als erster XPath-Schritt verwendet wird:

```

$m/preceding-sibling::node()
[name() = "Monat" or name() = "Strom"][1]
/@Stromstand

```

Im zweiten Schritt ist die Achsenbezeichnung `preceding-sibling` angegeben, es werden also die vorangegangenen Geschwister durchsucht. Durch `node()` wird der gesuchte Typ spezifiziert. Mit der ersten Zusatzbedingung wird die Auswahl auf `Monat`- oder `Strom`-Elemente beschränkt und das erste solche vorangegangene Element selektiert. Der letzte Schritt `@Stromstand` ist eine Kurzschreibweise für das Attribut `Stromstand`.

Insgesamt erhält man so den letzten vorangegangenen Stromstand eines Monats, so dass man dann später durch die Differenz mit dem Stromstand des Monats selbst den Stromverbrauch bekommt.

### 5.4.2 Funktionen

Die prozeduralen Elemente der Quilt-Anfragesprache finden ihren Ausdruck in den beiden Funktionsdefinitionen am Anfang der Anfrage. Die erste Funktion `rundeAufpfennig` lässt sich in anderen Programmiersprachen eleganter lösen, wie später noch gezeigt wird. Zu bemerken bleibt, dass die Gleitkommazahlen als Strings übergeben werden müssen, da sie anderenfalls von Kweelt nicht akzeptiert werden. Desweiteren wird der `mod`-Operator von Quilt in Kweelt zum `%`-Operator.

Die zweite Funktion enthält nochmals Beispiele für einen FLWR-Ausdruck und weitere XPath-Bezeichnungen. Auch in dieser Funktion gibt es eine Kweelt-Besonderheit: Der Vergleichsausdruck `<=` wird in Kweelt durch Punkte eingerahmt, um das Parsen zu vereindeutigen, da es sonst Interpretationskonflikte mit Elementkonstruktoren geben kann.

### 5.4.3 Java-Funktionen

Die Benutzerfreundlichkeit von Kweelt wird erhöht, indem externe Java-Funktionen importiert werden können. Abbildung 7 zeigt ein Beispiel für eine solche Java-Funktion zur Formatierung der Datumsangabe. Kweelt stellt dabei Template-Basisklassen zur Verfügung. Zur Fehlerbehandlung kann man sich eine `Exception` von `QuiltException` ableiten.

In die Anfrage fügt man solche Funktionen einfach durch die `import`-Deklaration hinzu:

```

import formatiereDatum
  as FunFormatDate;

...
<Rechnung
  Datum =
    formatiereDatum($a/@Datum)
  ...
</Rechnung>

```

```

import xacute.quilt.FunStringTemplate;
public class FunFormatDate
    extends FunStringTemplate {

    public String [] exec(String [] args) {
        // etwa "2001/07/13"
        String input = args[0];
        String result = "";
        try {
            // Berechnung des Ergebnisses:
            result = "13. Juli 2001";
        }
        catch (Exception e) {
            // Fehlerbehandlung
        }
        return new String[] { result };
    }
}

```

Abbildung 7: Java-Funktions-Beispiel

#### 5.4.4 Encoding

Die Sprache Quilt sieht Elementkonstruktoren vor, es gibt aber keine Möglichkeit, das Encoding anzugeben, das hier für den Monat März gebraucht wird. Auch Kweelt unterstützt dies nicht direkt. Vielmehr muss man den XML-AusgabeHandler um eine Encoding-Angabe erweitern. Diese Technik wird in Abschnitt 5.5.1 noch ausführlicher erläutert.

### 5.5 Ausgabe

Um das Ergebnis, eine Mietrechnung komfortabel und portabel drucken zu können, wurde das PDF-Format für die Endausgabe gewählt.

Aus den vielen Möglichkeiten, ausgehend von XML PDF zu erreichen, wurde hier der Weg über eine  $\text{T}_{\text{E}}\text{X}$ -Datei und deren Bearbeitung mit  $\text{pdfT}_{\text{E}}\text{X}$  gewählt, da über  $\text{T}_{\text{E}}\text{X}$  eine automatische Bearbeitung bei hoher Satzqualität möglich ist. Das Programm  $\text{pdfT}_{\text{E}}\text{X}$  ([18]) erzeugt dabei auf direktem Weg das PDF-Endprodukt.

#### 5.5.1 $\text{T}_{\text{E}}\text{X}$ -Zwischenstufe

$\text{T}_{\text{E}}\text{X}$  kann XML-Dateien direkt verarbeiten, das Parsen von  $\text{T}_{\text{E}}\text{X}$  aus ist aber relativ umständlich. Daher wurde in einem ersten Ansatz die XML-Ausgabe von Kweelt mittels dem selbstgeschriebenen `ConvertXMLtoTeX` in eine leicht von  $\text{T}_{\text{E}}\text{X}$  lesbare Syntax umgeschrieben. Für die Elementattribute wurde dabei auf das Pakets `keyval` zurückgegriffen, das hier eine 1:1-Abbil-

## Miete Juli 2001

Datum: 29. Juni 2001

Berechnung: Heiko Oberdiek

Stromzähler: 3062 kWh

Grundmiete:	300,00 DM
Nebenkosten:	6,80 DM
Telefongrundgebühr:	16,30 DM
Telefoneinheiten:	$153 \text{ E} \times 0,121 \text{ DM/E} = 18,51 \text{ DM}$
Stromeinheiten:	$45 \text{ kWh} \times 0,33 \text{ DM/kWh} = 14,85 \text{ DM}$
	<hr/>
	<b>356,46 DM</b>

Abbildung 8: Die fertige Rechnung als Beispiel

derung ermöglicht. Ein Beispiel ist im Anhang mit `Rechnung.xml` (7.2.1) und `Rechnung.tex` (7.2.2) aufgelistet.

Dank des modularen Aufbaus von Kweelt wurde in einer zweiten Version ein neuer Ausgabe-Handler `TeXOutputHandler.java` (siehe Anhang 7.2.3) geschrieben, der auf direktem Wege diese  $\text{\TeX}$ -Datei schreibt, ohne den Umweg über XML gehen zu müssen. Der Aufruf von Kweelt lautet dann:

```
kweelt -output=TeXOutputHandler ...
```

### 5.5.2 Wandlung nach PDF

Die Verarbeitung der XML-ähnlichen Syntax erfolgt durch die Klasse `xml-sup.cls`. Durch Makros unterstützt diese die Zuordnung der Elemente und Attribute zu Layoutelementen durch die Layoutklasse `Mietrechnung.cls`. Beide Klassen sind aufgrund ihrer Größe hier nicht aufgelistet worden.

Schließlich ergibt dann die pdf $\text{\TeX}$ -Compilierung die gewünschte PDF-Ausgabe, wie in Abbildung 8 zu sehen.

## 6 Diskussion Kweelt/Quilt

### 6.1 Abweichungen von Kweelt

Beim praktischen Experimentieren mit der Implementation Kweelt fielen ein paar kleinere Abweichungen von der Sprachdefinition Quilt auf:

- Für Kommentare sieht Quilt den SQL-Kommentartyp mit zwei Bindestrichen vor: `- Quilt`  
Kweelt unterstützt dagegen den C-Typ: `/* Kweelt */`
- Um Vergleichsoperatoren gegen Elementkonstruktoren besser abgrenzen zu können, werden in Kweelt die Vergleichsoperatoren durch Punkte abgegrenzt: `.<.`, `.<=.`, ...
- Die Integerzahl als 0 eingegeben wird in eine Fließkommazahl `0.0` umgewandelt, erst durch Eingabe als String `"0"` bleibt die Null erhalten.
- Fließkommazahlen können nur als Strings, etwa `"2.4"` eingegeben werden.
- Für die Restbildung sieht Quilt den Operator `mod` vor, Kweelt verwendet statt dessen `%`.
- Es werden nicht alle XPath-Möglichkeiten unterstützt. So funktioniert `position()=last()`, jedoch nicht `last()` alleine.
- Eine Namenskontrolle bei Endtags findet nicht statt.

### 6.2 Benutzerfreundlichkeit von Kweelt

- Kweelt ist modular aufgebaut, so können einzelne Teile leicht überschrieben oder erweitert werden, was am Beispiel des Ausgabehandlers gezeigt wurde.
- Der Benutzer hat die Möglichkeit, die Sprache durch eigene Java-Funktionen zu erweitern.
- Kweelt sieht eine erweiterte Dereferenzierungssyntax vor, um die Dereferenzierung zu vereindeutigen.

### 6.3 Vor- und Nachteile von Quilt

Ohne besondere Rücksicht auf Implementierungslimitierungen zu nehmen vereinigt der Sprachvorschlag Quilt sehr viele mächtige Features. Durch Aufnahme der verschiedensten Sprachelemente wurde Quilt sehr reichhaltig und bekam dadurch das Potential, die anderen Anfragesprachen abzulösen. Somit bildet Quilt einen wichtigen Schritt bei der Entwicklung eines allgemeinen Standards für eine Anfragesprache.

Durch die unterschiedlichsten Sprachkonstrukte ist Quilt als Sprache jedoch auch sehr inhomogen geworden, was sich u. a. in Parserproblemen äußert (s. beispielsweise Vergleichsoperatoren weiter oben).

Es fehlt ein Typsystem, das insbesondere bei der Dereferenzierung gute Dienste leisten würde. Dies wurde später in XQuery nachgeholt.

Höhere Freiheitsgrade für den Benutzer führen zu größerer Verantwortlichkeit. So kann eine Anfrage rekursive Funktionen enthalten, so dass die Anfrage nicht terminiert, ohne dass eine Implementierung die Chance hat, diese Situation abzufangen.

In der SQL-Welt können Anfragen auf eine Algebra abgebildet und so optimiert werden. Diese Möglichkeit fehlt derzeit für Quilt.

#### **6.4 Zukunft von Quilt**

Substanzielle Spuren, dass Quilt oder Kweelt weiterentwickelt wird, ließen sich außer einer Ankündigung einer zweiten Kweelt-Version aus dem letzten Jahr nicht finden. Im Gegenteil lässt etwa das Fehlen von neuen Vorträgen und Papers oder eine leere Mailingliste zu Kweelt die Vermutung zu, dass Quilt über den Status eines Sprachvorschlags für die XQuery-Gruppe nicht hinauskommen wird.

Wenn man die von dieser Gruppe entwickelte Sprache XQuery ([19]) mit Quilt vergleicht, fällt auf, dass praktisch die meisten Sprachelemente von Quilt unverändert übernommen wurden. Ergänzend kam etwa das bei Quilt fehlende Typsystem hinzu.

So besteht der Verdienst von Quilt in wichtigen Vorarbeiten für den Nachfolger, den sich entwickelnden Sprachstandard XQuery als einer allgemeinen XML-Anfragesprache.

## 7 Anhang

### 7.1 DTD-Dateien

#### 7.1.1 Mietkosten.dtd

```
<!-- Mietkosten DTD -->
<!ELEMENT Mietkosten (Grundmiete, Nebenkosten,
                       Telefongebuehr, Telefoneinheit,
                       Stromeinheit)>
<!ELEMENT Eintrag EMPTY>
<!ATTLIST Eintrag Datum CDATA #REQUIRED
                  Preis CDATA #REQUIRED
>
<!ELEMENT Grundmiete (Eintrag)+>
<!ELEMENT Nebenkosten (Eintrag)+>
<!ELEMENT Telefongebuehr (Eintrag)+>
<!ELEMENT Telefoneinheit (Eintrag)+>
<!ELEMENT Stromeinheit (Eintrag)+>
```

#### 7.1.2 Miete.dtd

```
<!-- Miete DTD -->
<!ELEMENT Miete (Monat|Strom)*>
<!ELEMENT Monat EMPTY>
<!ATTLIST Monat Monat CDATA #REQUIRED
               Datum CDATA #REQUIRED
               Telefoneinheiten CDATA #REQUIRED
               Stromstand CDATA #REQUIRED
>
<!ELEMENT Strom EMPTY>
<!ATTLIST Strom Datum CDATA #REQUIRED
                 Stromstand CDATA #REQUIRED
>
```

### 7.2 Ausgabe-Dateien

#### 7.2.1 Rechnung.xml

```
<?xml version="1.0"?>
<Mietrechnung>
  <Rechnung Datum="2001/06/29"
            Gesamtsumme="356.46"
            Grundmiete="300"
            Monat="2001/07"
            Nebenkosten="6.80"
            Stromeinheit="0.33"
            Stromeinheiten="45.0"
  >
```

```

        Stromstand="3062"
        Stromsumme="14.85"
        Telefoneinheit="0.121"
        Telefoneinheiten="153"
        Telefongebuehr="16.30"
        Telefonsumme="18.51"
    </Rechnung>
</Mietrechnung>
<!-- end of document -->

```

### 7.2.2 Rechnung.tex

```

\NeedsTeXFormat{LaTeX2e}
\documentclass[Mietrechnung]{xmlsup}
\begin{document}
\ELEM{Mietrechnung}{}%
\ELEM{Rechnung}{}%
    Datum={2001/06/29},%
    Gesamtsumme={356.46},%
    Grundmiete={300},%
    Monat={2001/07},%
    Nebenkosten={6.80},%
    Stromeinheit={0.33},%
    Stromeinheiten={45.0},%
    Stromstand={3062},%
    Stromsumme={14.85},%
    Telefoneinheit={0.121},%
    Telefoneinheiten={153},%
    Telefongebuehr={16.30},%
    Telefonsumme={18.51},%
}%
\MELE{Rechnung}%
\MELE{Mietrechnung}%
\end{document}

```

### 7.2.3 TeXOutputHandler.java

```

import java.io.PrintWriter;
import java.util.StringTokenizer;

import org.xml.sax.AttributeList;
import org.xml.sax.DocumentHandler;
import org.xml.sax Locator;

public class TeXOutputHandler
    implements DocumentHandler

```

```

{
    private boolean firstElement = true;
    private PrintWriter pw;

    public TeXOutputHandler()
    {
        this.pw = new PrintWriter( System.out );
    }

    public TeXOutputHandler(PrintWriter pw)
    {
        this.pw = pw;
    }

    public String toString()
    {
        pw.flush();
        return "";
    }

    public void characters(char [] ch, int start , int length)
    {
        StringTokenizer tok =
            new StringTokenizer(new String(ch), "\n\r");
        while(tok.hasMoreTokens())
        {
            pw.print( tok.nextToken() );
        }
        pw.println();
        return;
    }

    public void endDocument()
    {
        pw.println("\\end{document}");
    }

    public void endElement(String name)
    {
        pw.println("\\MELE{" + name + "}");
    }

    public void ignorableWhitespace(char [] ch, int start , int length)
    {
        return;
    }
}

```

```

public void processingInstruction(String target, String data)
{
    return;
}

public void setDocumentLocator(Locator locator)
{
    return;
}

public void startDocument() {
    pw.println("\\NeedsTeXFormat{LaTeX2e}");
}

public void startElement(String name, AttributeList atts)
{
    // the first element should be the document node
    if (firstElement) {
        firstElement = false;
        pw.println("\\documentclass[" + name + "]{xmlsup}");
        pw.println("\\begin{document}");
    }
    pw.print("\\ELEM{" + name + "}");
    if (atts != null && atts.getLength() > 0) {
        pw.println("{%");
        for(int i = 0; i < atts.getLength(); ++i) {
            pw.println("  " + atts.getName(i) + "={ " +
                atts.getValue(i) + "},%");
        }
        pw.println("}%");
    } else {
        pw.println("{%");
    }
}
}
}

```

## Literatur

- [1] Don Chamberlin, Peter Fankhauser, Massimo Marchiori und Jonathan Robie. *XML Query Requirements, W3C Working Draft 15 February 2001*. <<http://www.w3.org/TR/2001/WD-xmlquery-req-20010215>>, 15. Februar 2001.
- [2] Angela Bonifati und Stefano Ceri. *Comparative Analysis of Five XML Query Languages*. Published in ACM Sigmod Record, März 2000.
- [3] Tim Bray, Jean Paoli und C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0*. <<http://www.w3.org/TR/1998/REC-xml-19980210>>, 10. Februar 1998.
- [4] Andreas Heuer und Gunter Saake. *Datenbanken: Konzepte und Sprachen*, Bd. 1, Thomson Publishing, 1995.
- [5] Jonathan Robie, Don Chamberlin und Daniela Florescu. *Quilt: An XML query language*. <[http://www.almaden.ibm.com/cs/people/chamberlin/quilt\\_euro.html](http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html)>, 31. März 2000.
- [6] Nikolai Karcher. *XML-Query: Abfragesprachen für XML-Datenbanken*. <<http://www.ipd.ira.uka.de/~oosem/WebDB0001/ausarbeitung/NicolaiKarcher.pdf>>, 1. März 2001.
- [7] Pia Schneider. *XML Extensible Markup Language*. <<http://www.tm.informatik.uni-frankfurt.de/PSWWW/ausarbeitung/XMLVortragPiaSchneider.pdf>>, 16. Januar 2001.
- [8] Mary Fernandez und Jonathan Robie. *XML Query Data Model, W3C Working Draft 11 May 2000*. <<http://www.w3.org/TR/2000/WD-query-datamodel-20000511>>, 11. Mai 2000.
- [9] James Clark und Steve DeRose. *XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999*. <<http://www.w3.org/TR/1999/REC-xpath-19991116>>, 16. November 1999.
- [10] Don Chamberlin, Jonathan Robie und Daniela Florescu. *Quilt: An XML Query Language for Heterogeneous Data Sources*. <[http://www.almaden.ibm.com/cs/people/chamberlin/quilt\\_lncs.pdf](http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf)>, 10. September 2000.
- [11] Don Chamberlin. *Quilt*. <<http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html>>, 2000.

- [12] Justin Campbell, Dan Grossman und Ann-Marie Popescu. *Quilt2Sql*. <<http://www.cs.washington.edu/homes/grossman/projects/544project/>>, 27. Juni 2000.
- [13] Justin Campbell, Dan Grossman und Ann-Marie Popescu. *Quilt2Sql—An XML Storage Schema and Query Engine for the Quilt Query Language*. <<http://www.cs.washington.edu/homes/grossman/projects/544project/Quilt2Sql.ps>>, 5. Juni 2000.
- [14] Daniela Florescu, Ioana Manolescu, Donald Kossmann u. a. *The Agora Project*. <<http://www-caravel.inria.fr/~ioana/AGORA/index.html>>, 2001.
- [15] *XPRESS Project, Project Overview*. <<http://www.cobase.cs.ucla.edu/projects/xpress/overview.html>>, 20. September 2000.
- [16] Henry Chiu und Dongwon Lee. *QuiltParser: JavaCC Grammar for Quilt XML Query Language*. <<http://www.cobase.cs.ucla.edu/projects/xpress/quilt>>, 17. August 2000.
- [17] Arnaud Sahuguet. *KWEELT, Querying XML in the New Millennium*. <<http://db.cis.upenn.edu/Kweelt/>>, 14. September 2000.
- [18] Sebastian Rahtz. *PDFTeX support*. <<http://www.tug.org/applications/pdftex/>>, 20. November 2000.
- [19] Don Chamberlin, James Clark, Daniela Florescu u. a. *XQuery 1.0: An XML Query Language, W3C Working Draft 07 June 2001*. <<http://www.w3.org/TR/2001/WD-xquery-20010607>>, 7. Juni 2001.