Manipulation von XML-Dokumenten in Tamino

Dr. Michael Gesmann
Software AG, Darmstadt
michael.gesmann@softwareag.com

Abstract: Dieser Beitrag untersucht die Frage, wie XML-Dokumente, die in Datenbanksystemen abgespeichert sind, effektiv und effizient innerhalb dieser Systeme verändert werden können. Zunächst wird skizziert, welche Lösungsansätze bereits existieren. Danach wird erläutert, wie dieses Problem in der neuen Version des XML-Datenbanksystems Tamino von der Software AG gelöst wird. Dort wird ein sprachbasierter Ansatz auf der Basis von XQuery-Erweiterungen verfolgt.

1 Die Aufgabe

Mittlerweile ist unbestritten, dass auch XML-Dokumente in Datenbanken verwalten werden sollen. Auch die größten Datenbankhersteller arbeiten an effizienten und benutzerfreundlichen Schnittstellen zur Abspeicherung und Bearbeitung solcher Dokumente.
Als Anfragesprache erarbeitet das W3C zur Zeit eine Empfehlung für XQuery [XQ02]. Jede andere Funktionalität ausser Anfragen wurde dort explizit herausgelassen, um den Definitionsprozess nicht unnötig zu verlängern. Dies gilt auch für Änderungsoperationen in XML-Dokumenten. Für die XML-Technologie gilt daher zur Zeit, dass es keine dem SQL-Update vergleichbare standardisierte Operation gibt.
Anwendungen, die in einer Datenbank abgespeicherte XML-Dokumente verändern wollen, bleibt damit nur die Manipulation der Dokumente in Anwendungsprogrammen. Dokumente können mittels XQuery gelesen, dann von einer Anwendung verändert und abschließend wieder über die jeweilige Systemschnittstelle in die Datenbank zurückgeschrieben werden. Dabei wird die Performanz in vielen Fällen völlig inakzeptabel. Wenn zum Beispiel in einem Bericht ein einzelner Abschnitt eingefügt, entfernt oder einfach nur ein Rechtschreibfehler korrigiert werden soll, muss dazu der gesamte Bericht ausgelesen und abgespeichert werden. Der Dokumenttransfer von der Datenbank zur Anwendung und zurück verursacht viel zu hohen Zusatzaufwand, da viele nicht benötigte Teile des Dokuments auch gelesen und geschrieben werden müssen. Die schlechte Performanz zwingt alle Datenbankhersteller, zusätzliche Funktionalität bereitzustellen, mit der Dokumentänderungen direkt in der Datenbank möglich werden. Weil ein einheitlicher Standard fehlt und Lösungen schnell benötigt werden, entstehen viele verschiedene Lösungen.
Im folgenden Abschnitt werden bereits existierende Lösungen skizziert. Der Ansatz, auf den wir bei der Entwicklung von Tamino zurückgreifen, wird in den folgenden Abschnitten ausführlicher beschrieben. Grundlegende Kenntnisse über XML, das XQuery-Datenmodell und XQuery selber werden hier vorausgesetzt.

2 Lösungsansätze und eine grobe Bewertung

Während über Anfragesprachen für XML und die Abspeicherung von XML in den vergangenen Jahren eine Vielzahl an Publikationen erschienen ist, wurden Änderungen von XML-Dokumenten in einer Datenbank bisher sehr wenig untersucht. In diesem Abschnitt sollen einige Ansätze und Untersuchungen beschrieben und bewertet werden, soweit dies zur Zeit überhaupt möglich ist.

Bewertungskriterien
Für die Bewertung werden folgende wesentliche Kriterien herangezogen:

- Performanz: Ein Ansatz sollte einer möglichst großen Anzahl von Anwendungen eine gute Performanz ermöglichen. Dabei sollte die Performanz nicht allein auf effizienter Anwendungsimplementierung beruhen, sondern auch auf die Optimierungsmöglichkeiten eines Datenbanksystems zurückgreifen können.
- Ausdrucksstärke: Der zu realisierende Mechanismus sollte einer breiten Anwendungspalette die Möglichkeit geben, Änderungsoperationen zu spezifizieren. Dazu muss mindestens folgende Basisfunktionalität angeboten werden:
  - Navigation zu Knoten, deren Inhalte tatsächlich geändert werden sollen
  - Berechnung neuer Knoteninhalte
  - Einfügen neuer Knoten in ein Dokument
  - Löschen von Knoten im Dokument
  - Umhängen bereits existierender Knoten an eine andere Stelle im Dokument
  - Ändern von Knoteninhalten, zum Beispiel Text in einem Textknoten
  - Umbenennen von Knoten.

  Außerordentlich hilfreich sind folgende Eigenschaften:
  - Ausführung mehrerer Basisfunktionen in einer Änderungsoperation
  - Schachtelung von Anfragen in die Änderungsoperationen
  - Temporäre Akzeptanz von Dokumenten, die gemäss einem gegebenen XML-Schema ungültig oder sogar nicht wohlgeform sind

  Im Datenbanksystem ist außerdem wünschenswert:
  - Das Sichtfeld einer Änderungsoperation sollte einzelne Dokumente übersteigen, das heißt, für Änderungen in einem Dokument "A" sollten auch Inhalte anderer Dokumente in der Datenbank mit einbezogen werden können (vgl. Join-Funktionalität in XQuery).
- Einfachheit: Eine Sprache zur Beschreibung von Änderungsoperationen sollte für Benutzer möglichst einfach verständlich und anwendbar sein.

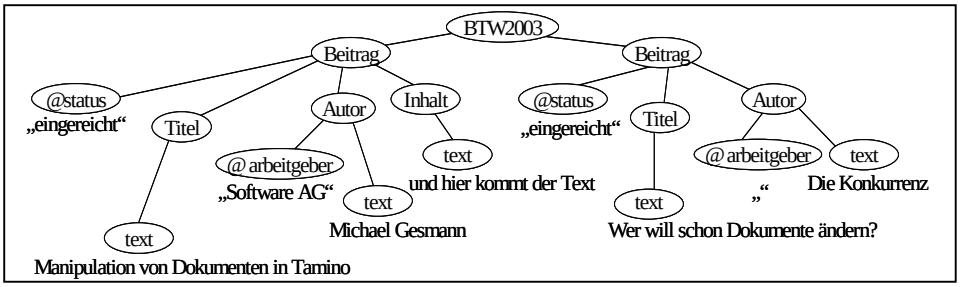
Weiterhin sollte jeder Ansatz mit anderen XML Empfehlungen des W3C konform sein. Ansätze, die zum Beispiel mit den Empfehlungen für XPath [XP99] oder Namespaces [XNs99] nicht zu vereinbaren sind, werden nur als proprietäre und schwer vermittelbare Speziallösungen Bestand haben können.
Die Forderung aus den XUpdate-Anforderungen von XML:DB [Xup02R], dass die Formulierung in Form eines XML-Dokuments erfolgen muss, wird hier explizit weggelassen. Die Erfahrungen mit XQuery zeigen, dass es sehr schwer ist, eine übersichtliche und einfach verständliche XML-Syntax für eine solche Sprache zu definieren.

Beispiel
Anhand des folgenden einfachen XML-Dokuments sollen Änderungsoperationen illustriert werden:

```
<?xml version="1.0"?>
<BTW2003>
  <Beitrag status="eingereicht"> <Titel>Manipulation von Dokumenten in Tamino</Titel>
                                 <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
                                  <Inhalt> und hier kommt der Text </Inhalt>          </Beitrag>
  <Beitrag status="eingereicht"> <Titel>Wer will schon Dokumente ändern?</Titel>
                                 <Autor arbeitgeber="">Die Konkurrenz</Autor>       </Beitrag>
</BTW2003>
```

Dokument 1: Ausgangsdokument zur Erläuterung von Beispielen

Zur Illustration von durchzuführenden Änderungsoperationen ist es häufig günstiger, das Dokument als Baum aufzuzeigen:



In den nachfolgend skizzierten Ansätzen soll jeweils der Titel des Beitrags vom Autor "Michael Gesmann" auf "Änderung von Dokumenten in DBMS" modifiziert werden.

2.1 Offen gelegte Datenbankfunktionalität

Die XML-Erweiterungen der relationalen Datenbanksyteme von IBM und Oracle ermöglichen derzeit Änderungen "lediglich" über vorgegebene Schnittstellen von UDFs (user defined functions) an[1].

Update() von DB2
Die "update(Dokument, LocationPath, Wert)"-Funktion auf Spalten vom Typ "XML UDT" in [Db2] bekommt drei Argumente; zuerst das zu verändernde Dokument, dann einen LocationPath, über den eine Menge von Knoten in dem Dokument bestimmt wird, und als drittes noch einen Wert, der in allen gefundenen Knoten deren alten Wert ersetzt. Eine Änderungsoperation sieht hier folgendermaßen aus:

[1]Dokumente können in diesen Systemen auch in Tabellen und Spalten strukturiert werden anstatt sie in Textform in einem CLOB abzulegen. Dann bleibt als besonders effizienter Weg auch die Möglichkeit, die generierten SQL-Strukturen direkt zu manipulieren. Dieser Weg wird hier nicht weiter berücksichtigt, weil er keine echte Alternative darstellt. Dieser Ansatz gewährleistet nicht ohne weiteres, dass die Dokumente gemäß einem gegebenen Schema gültig bleiben. Ausserdem führen Änderungen im XML-Schema zu veränderter Abbildung auf Tabellen, wodurch Anwendungen teilweise unnötigerweise überarbeitet werden müssen.

```
UPDATE TabelleMitTagungsDokumenten
SET dokumentSpalte = db2xml.update(dokumentSpalte,
                          '/BTW2003/Beitrag[Autor/@name="Michael Gesmann"]/Titel',
                          "Manipulation von Dokumenten in DBMS")
WHERE tagungSpalte="BTW" and jahrSpalte=2003
```

Änderung 1: Beispieländerung in DB2

Prädikate im LocationPath kann DB2 nur auf Attributen auswerten. Deshalb ist eine Änderung des Prädikates und eine andere Modellierung der Dokumente erforderlich.

UPDATEXML() von Oracle

[Ora9i2] erlaubt die Abspeicherung von XML-Dokumenten in Tabellen oder Spalten vom Typ XMLType. Dokumente können in ihrer ursprünglichen Form als CLOBs abgespeichert werden. Da das System in diesem Fall die Struktur der Dokumente nicht kennt, können diese Dokumente nur in ihrer Textform ganz oder gar nicht verändert werden. Alternativ können Dokumente strukturiert abgelegt werden, wenn ein XML-Schema vorliegt. In diesem Fall können Änderungen mittels der XML member-Funktion "UPDATEXML(Dokument, Paare von LocationPath und Wert)" verändert werden. Die Schnittstelle hier ist etwas mächtiger als die von DB2. Es können mehrere XPath-Ausdrücke mit neuen Werten angegeben werden. Ferner dürfen sich Prädikate in den XPath-Ausdrücken auch auf Elemente beziehen. Somit kann eine Änderung wieder auf dem ursprünglichen Dokument ausgeführt werden:

```
UPDATE TabelleMitTagungsDokumenten
SET dokumentSpalte = UPDATEXML(dokumentSpalte,
                          '/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel',
                          "Manipulation von Dokumenten in DBMS")
WHERE tagungSpalte="BTW" and jahrSpalte=2003
```

Änderung 2: Beispieländerung in Oracle

UPDATEXML() beschreibt nur die Veränderung eines gegebenen Dokuments. Damit die Änderung auch in die Datenbank einfließt, muss der Aufruf genauso wie bei DB2 in eine SQL Update-Anweisung eingebunden werden.

Server Extensions von Tamino

Tamino [Tam41] bietet einen noch offeneren Mechanismus an, die sogenannten Server Extensions. Eine Server Extension wird als Anwendungsfunktion implementiert. Nachdem sie von einem Administrator in einer Datenbank registriert wurde, kann sie genauso wie vordefinierte Funktionen aufgerufen und im Server ausgeführt werden. Eine Server Extension kann beispielsweise zunächst eine XQuery ausführen, anschließend die Ergebnisdokumente modifizieren und schließlich wieder zurückschreiben.

Da die Funktionen in allen drei Fällen innerhalb des DBMS ausgeführt werden, entfallen Kommunikationskosten mit der Anwendung. Man kann also von diesen Ansätzen eine bessere Performanz als bei der anwendungsseitigen Verarbeitung erwarten. Bei den relationalen Erweiterungen ist die angebotene Funktionalität aber sehr stark eingeschränkt. So werden neue Werte wieder absolut vorgegeben und relative Änderungen wie das Einfügen eines neuen zusätzlichen Knotens werden nicht angeboten. Relationale Anfragen werden hier mit XPath-Ausdrücken gemischt. Die

Server Extensions von Tamino dagegen erfordern komplexe Anwendungsprogrammierung auf den Dokumenten. Man kann erwarten, dass einfachere Funktionalität auch effektiver optimiert werden kann. Auf zusätzliche Problem wie Zugriffskontrolle und Anfrageoptimierung soll hier nicht weiter eingegangen werden.

2.2 Eigenständige Beschreibungssprachen

Eine weitere Klasse von Lösungsansätzen entwickelt eigenständige Sprachen zur Beschreibung von Änderungsoperationen. Auf diese Sprachen wird als nächstes eingegangen.

XML:DB
Die XML:DB Initiative publiziert zur Zeit gleich zwei Sprachvorschläge. SiXDML (Simple XML Data Manipulation Language) [SiXDML02] ist eng an SQL angelehnt, und umfaßt auch viele Administrations- und Schemaoperationen. Im Gegensatz dazu konzentriert sich XUpdate [Xup00] auf die Manipulation von Dokumentinhalten. Änderungsoperationen werden dort in XML-Dokumenten formuliert. Für die Auswahl der zu verändernden Knoten benutzt XUpdate XPath-Ausdrücke. Elemente in einem XUpdate-Dokument beschreiben Änderungen, die in einem gegebenen Dokument ausgeführt werden sollen. Es können mehrere solcher Änderungen angegeben werden, die dann sequentiell ausgeführt werden. Die Spezifikation legt nicht fest, wie ein einzelnes oder mehrere zu verändernde Dokumente bestimmt werden.
Die Beispieländerung wird hier folgendermaßen beschrieben:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:update select='/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel'>
    Änderungen von Dokumenten in XML-Datenbanken
  </xupdate:update>
</xupdate:modifications>
```

Änderung 3: Beispieländerung mit XUpdate

Updategrams in eXcelon
Updategrams in eXcelons Information Server [XIS312] umfassen eine gegenüber XUpdate erweiterte Funktionalität. Dort können Teilbäume eines Dokuments kopiert werden, Änderungsanweisungen lassen sich ineinander schachteln, und mit Hilfe von Variablen und Schleifenkonstrukturen ("foreach") können auch komplexere Update-Anwendungen definiert werden. Auch Updategrams beschreiben nur, wie einzelne Dokumente geändert werden. Um eine Menge von Dokumenten zu verändern, müssen zusätzliche Mechanismen herangezogen werden.

Updategrams in Microsofts SQL-Server
Im Unterschied zu eXcelons Updategrams geben Updategrams im SQL Server von Microsoft [SQLServ2000] keine Berechnungsvorschriften an, mittels derer die Änderungen durchgeführt werden sollen. Stattdessen beschreiben Before- und After-Elemente eines Updategrams die durchzuführenden Änderungen. Ist kein Before-

Element, sondern nur ein After-Element angegeben, dann wird ein neues Dokument eingefügt. Ist nur ein Before-Element angegeben, aber kein After-Element, dann wird das im Before-Element identifizierte Dokument gelöscht. Sind sowohl ein Before-Element als auch After-Element angegeben, dann wird das im Before-Element identifizierte Dokument entsprechend den Beschreibungen im After-Element geändert. Die beschriebenen Änderungen werden in SQL-Befehle umgesetzt, mit denen die zugrundeliegenden Tabellen modifiziert werden. Wenn für die Abbildung von XML-Elementen und -Attributen auf Tabellen und Spalten kein Default-Verfahren ausreicht, muss ein Schema mit entsprechenden Informationen angegeben werden. Für die selektive Veränderung einzelner Elemente wird ein eindeutiger Identifikator benötigt. Inhalte müssen aber immer vollständig angegeben werden. Es ist zum Beispiel nicht möglich, in einen Knoten, der bereits mehrere Kinder hat, einen weiteren Knoten einzeln einzufügen. Vielmehr muss der Knoten mit seinem gesamten Inhalt neu angegeben werden.

2.3 Spracherweiterungen für XQuery

XUpdate und Updategrams sind unabhängig von der neuen Anfragesprache XQuery entstanden. Mit der Entwicklung von XQuery besteht nun auch die Möglichkeit Änderungsoperationen in XQuery zu integrieren. So beschreibt [TIHW01] zunächst eine kleine Menge von einfachen Änderungsoperationen und wie diese Operationen in einer Erweiterung von XQuery ausgedrückt werden können. Anschließend wird untersucht, mit welchen Mitteln die Operationen bei einer Abbildung der Dokumente auf relationale Tabellen ausgeführt werden können. [Le01] beschreibt einen zu [TIHW01] leicht erweiterten Funktionsumfang, der ebenfalls als Erweiterung von XQuery formuliert wird. Die in der Arbeit beschriebenen Spracherweiterungen wurden in einen XQuery Prototypen namens QuiP [QuiP00] integriert. Nach [Le01] wird die Beispieländerung folgendermaßen formuliert ([TIHW01] hat lediglich eine etwas andere Syntax):

```
update
replace input()/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel/text()
        with "Manipulation von Dokumenten in DBMS"
```

Änderung 4: Beispieländerung nach [Le01]

oder alternativ:

```
update
for $text in input()/BTW2003/Beitrag[Autor="Michael Gesmann"]/Titel/text()
replace $text with "Manipulation von Dokumenten in DBMS"
```

Änderung 5: Alternative Formulierung der Beispieländerung nach [Le01]

[TIHW02] und [Le01] decken die in den Bewertungskriterien aufgeführte Basisfunktionalität ab und erlauben es, mehrere Änderungsoperationen in einer Anfrage anzugeben. Der Anfrageteil bestimmt die zu verändernden Dokumentteile, ein Update-Teil spezifiziert die durchzuführenden Änderungen. Für die verschiedenen Suchanteile der Operationen wird immer XQuery selbst verwendet.

2.4 Zusammenfassung und weitere Aspekte

Die anwendungsseitige Verarbeitung von Änderungsoperationen ist die einzige Möglichkeit, Dokumente zu verändern, wenn das Datenbanksystem die XML-Semantik der Dokumente nicht versteht. Alle in diesem Abschnitt aufgeführten Ansätze „verstehen XML“ und wurden in DBMS implementiert, so dass die Kommunikationskosten zwischen Anwendung und DBMS reduziert werden können. Die relationalen Erweiterungen fallen vor allen Dingen durch die deutlich eingeschränkten Funktionalitäten auf. Die Beschreibung von Änderungsoperationen in Updategrams liefert eine deutlich mächtigere Funktionalität. Dort fehlt eine enge Einbindung in die mittlerweile vorhandene Anfragesprache XQuery. Dieser Punkt ist wichtig, weil die Suche der zu verändernden und der neu einzutragenden Knoten wesentlicher Bestandteile jeder Änderungsoperation ist. Der zuletzt aufgeführte Ansatz über Spracherweiterungen von XQuery, weist diese Schwäche nicht auf. Gleichzeitig erfüllt er Kriterien nach Performanz und Ausdrucksstärke mindestens genauso gut wie die anderen Ansätze. Aus diesen Grüunden wurde Spracherweiterungen von XQuery als Basis für die Weiterentwicklung von Tamino ausgewählt. Die folgende Tabelle soll die Bewertung noch einmal illustrieren und für jeden aufgeführten Ansatz festhalten, wo Stärken und Schwächen liegen („+“ verdeutlicht Stärken, „-“ deutet auf Schwächen hin):

Ansatz	Performanz	Ausdrucksstärke	Einfachheit
UDF von DB2	kann Optimierungen des DBMS nutzen (+)	Nur sehr eingeschränkte einzelne Wertveränderungen in einzelnen Dokumenten (-)	XPath, Dokumentzugriff via SQL (+/-)
UDF von Oracle9i Version 2	s.o. (+)	Ebenfalls sehr eingeschränkter Funktionsumfang (mehr als DB2), Änderungen eingeschränkt auf einzelne Dokumente (-)	XPath, Dokumentzugriff via SQL (+/-)
Server Extension von Tamino	schwer in Optimierung zu integrieren (+/-)	Anwendungsprogrammierung (+/-)	XQuery incl. XPath und DOM werden typischer-weise benötigt (-)
XML:DB - XUpdate	kann Optimierungen des DBMS nutzen (+)	XPath, Änderungen eingeschränkt auf einzelne Dokumente, mehrere Änderungsanweisungen in einer Anforderung möglich (+/-)	XPath plus Änderungsoperationen und Konstruktoren, zusätzliche Einbindung in Queryfunktionalität notwendig (+/-)
Updategram von eXcelon	s.o. (+)	Wie XUpdate aber reichhaltigerer Funktionsumfang (+/-)	Einbindung in Queryfunktionalität notwendig (+/-)
Updategram von SQLServer	s.o. (+)	Einfügen und Löschen möglich, alle anderen Operationen erfordern vollständig neue Definition des Inhalts (-)	Abbildung auf relationale Tabellen muss beachtet werden (-)
Erweiterungen von XQuery	s.o. (+)	Volle XQuery-Mächtigkeit auch über mehrere Dokumente, mehrere Anweisungen in einer Anforderung möglich (+)	XQuery plus einige einfache Änderungs-operationen und Konstruktoren (+)

Tabelle 1: Zusammenfassende Bewertung der Ansätze

3 Spracherweiterungen von XQuery in Tamino

Mit der Version 4.1 von Tamino werden Anwendungen verbesserte Mechanismen zur Modifikation von Dokumenten in der Datenbank zur Verfügung gestellt. Grundlegende XQuery Funktionalität steht in der Version 4.1 dafür zur Verfügung. Mit [Le01] war einige Grundlagenarbeit bereits durchgeführt, auf deren Basis die weitere Entwicklung angegangen wurde. Syntax und Semantik der in Tamino implementierten XQuery-Spracherweiterungen werden in diesem Abschnitt vorgestellt.
Jede **Änderungsoperation** beginnt mit dem Schlüsselwort "update". Darauf folgen eine oder mehrer UpdateExprs. Jede UpdateExpr kann entweder direkt eine der **Elementaroperation** Einfügen, Löschen, Umbenennen oder Ersetzen beschreiben, oder sie kann komplexer zusammengesetzt sein und dazu in einer FLWUExpr formuliert werden. Mit den Elementaroperationen läßt sich die gesamte in Abschnitt 2 aufgeführte Basisfunktionalität realisieren. Eine FLWUExpr sieht fast genauso aus wie eine normale XQuery, lediglich die result-Klausel einer XQuery wird durch Änderungsanweisungen ersetzt.
Abbildung 1 zeigt alle Syntaxerweiterungen. Syntaxelemente, die im XQuery Working Draft [XQ02] definiert sind, werden kursiv wiedergegeben.

```
Query           ::= QueryProlog ( [ ExprSequence] | 'update' UpdateSequence ) 'Eof'
UpdateSequence  ::= UpdateExpr +
UpdateExpr      ::= InsertClause |DeleteClause |RenameClause |ReplaceClause |FLWUExpr

InsertClause    ::= 'insert' Expr ('following'|'preceding'|'into'|) Expr
DeleteClause    ::= 'delete' Expr
RenameClause    ::= 'rename' Expr 'as' QName
ReplaceClause   ::= 'replace' Expr 'with' Expr

FLWUExpr        ::= ( ForClause | LetClause )* [ WhereClause ]
                        'do' ( UpdateExpr | Lpar UpdateSequence Rpar )
```

Abbildung 1: Syntaxerweiterungen für XQuery

Die vorhandendene XQuery Funktionalität wird für zwei Teilaufgaben benötigt. Zunächst einmal müssen diejenigen Knoten bestimmt werden, an denen Änderungen durchgeführt werden sollen. Dazu können einfache oder durch komplexere Ausdrücke mit For-, Let- und Where-Klauseln von XQuery genutzt werden. Die so bestimmten Knoten werden im folgenden **Änderungsknoten** genannt.
Zusätzlich werden die eigentlichen Änderungen beschrieben. Dabei müssen die neuen Werte keine Konstanten sein, sondern können ebenfalls durch XQuery-Ausdrücke berechnet werden. Diese Knoten, die neue Werte repräsentieren, werden im folgenden **Werteknoten** genannt.

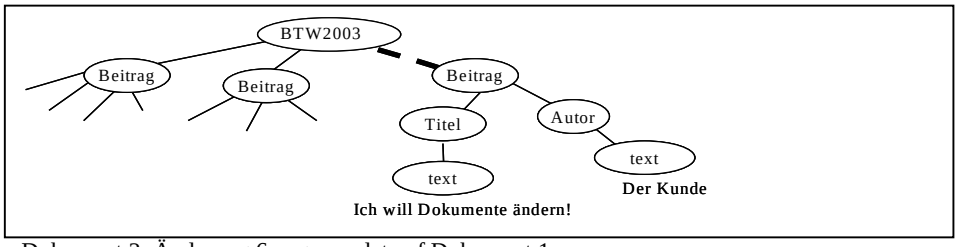
3.1 Elementaroperationen

Insert
Die Einfügeoperation dient dazu, Werteknoten an eine bestimmte Position in ein Dokument einzutragen. Mit der ersten Expr in der InsertClause werden einer oder eine Sequenz einzufügender Werteknoten bestimmt. Mit der zweiten Expr in der InsertClause

werden die Elternknoten als Änderungsknoten und die Einfügeposition bestimmt. Abhängig von dem dazwischen liegenden Schlüsselwort wird die Einfügeoperation ausgeführt.
Bei einem "insert ... into ... " werden die Werteknoten an das Ende der aktuellen Kinderliste von jedem Änderungsknoten angehängt, wenn es sich bei den Werteknoten nicht um Attribute handelt. Falls ein Änderungsknoten noch keine Kinder hat, wird die Kinderliste neu angelegt. Die Reihenfolge der Werteknoten bleibt erhalten. Wenn es sich bei den Werteknoten um ein oder mehrere Attribute handelt, dann werden diese bei einem "insert ... into ..." in den Änderungsknoten eingetragen.
Auch bei einem "insert ... preceding ..." werden die Werteknoten in den Elternknoten eingefügt. Wiederum ist der Elternknoten der Änderungsknoten. Die zweite Expr in der InsertClause charakterisiert die Eltern-Kind-Beziehung vor die unmittelbar die neuen Kinder eingefügt werden sollen. Im Fall von "insert ... following ... " werden die Werteknoten in den Elternknoten unmittelbar hinter die durch die zweite Expr charakterisierte Eltern-Kind-Beziehung eingefügt. Da es auf Attributen keine Reihenfolge gibt, sind diese beiden Formen des Einfügens nicht mit Attributen möglich.
Es folgen einige Beispiele, die auf der Basis von Dokument 1 aus Abschnitt 2 ausgeführt werden. Zuerst wird ein neuer Beitrag an das Ende des Knotens BTW2003 eingetragen:

```
update insert <Beitrag><Titel>Ich will Dokumente ändern!</Titel>
                        <Autor>Der Kunde</Autor></Beitrag>
            into input()/BTW2003
```

Änderung 6: Neues Dokument einfügen

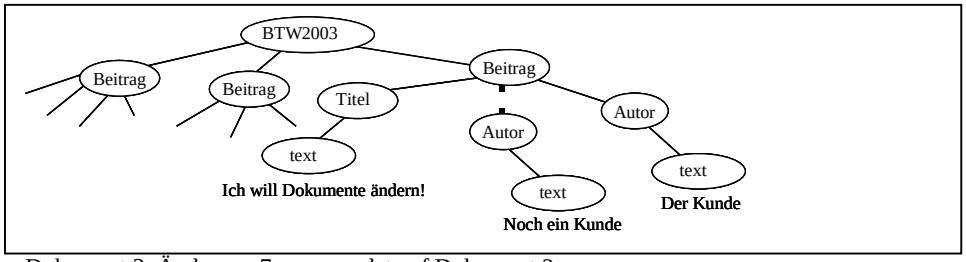


Dokument 2: Änderung 6 angewendet auf Dokument 1

Trage in das soeben neu eingefügte Beitrag-Element einen neuen Autor an den Anfang der Autorenliste ein:

```
update insert <Autor>Noch ein Kunde</Autor>
            preceding input()/BTW2003/Beitrag[Titel="Ich will Dokumente ändern!"]/Autor[1]
```

Änderung 7: Zusätzlichen Autor einfügen



Dokument 3: Änderung 7 angewendet auf Dokument 2

Trage in in den neuen Beitrag noch das Attribute „status“ ein:

```
update insert attribute status {"eingereicht"}
            into input()/BTW2003/Beitrag[Titel="Ich will Dokumente ändern!"]
```

Änderung 8: Zusätzliches Attribut einfügen

Diese drei Einfügeoperationen angewandt auf Dokument 1 (abzüglich einiger Kürzungen) führen zu folgendem Dokument:

```
<?xml version="1.0"?>
<BTW2003> <Beitrag status="eingereicht"> ... gekürzt ... </Beitrag>
              <Beitrag status="eingereicht"> ... gekürzt ... </Beitrag>
   <Beitrag status="eingereicht"> <Titel>Ich will Dokumente ändern!</Titel>
          <Autor>Noch ein Kunde</Autor> <Autor>Der Kunde</Autor></Beitrag>
</BTW2003>
```

Dokument 4: Änderungen 6 bis 8 angewandt auf Dokument 1

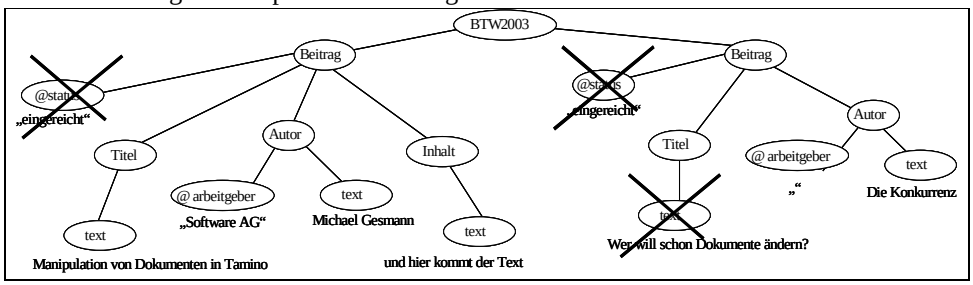
Delete

Mit Hilfe der Löschoperation können Knoten aus Dokumenten entfernt werden. Mit der angegebenen Expr werden die zu löschenden Knoten (Änderungsknoten) berechnet.
Die folgende Anweisung löscht aus allen Beiträgen das status-Attribut und löscht zusätzlich den Inhalt des Titel-Elements im Beitrag mit dem Titel „Wer will schon Dokumente ändern?“.

```
update delete input()/BTW2003/Beitrag/@status
       delete input()/BTW2003/Beitrag/Titel[. = "Wer will schon Dokumente ändern?"]/text()
```

Änderung 9: Löschen von Attributen und Textknoten

Die Ausführung dieser update-Anweisung auf Dokument 1 liefert:



Dokument 5: Änderung 9 angewandt auf Dokument

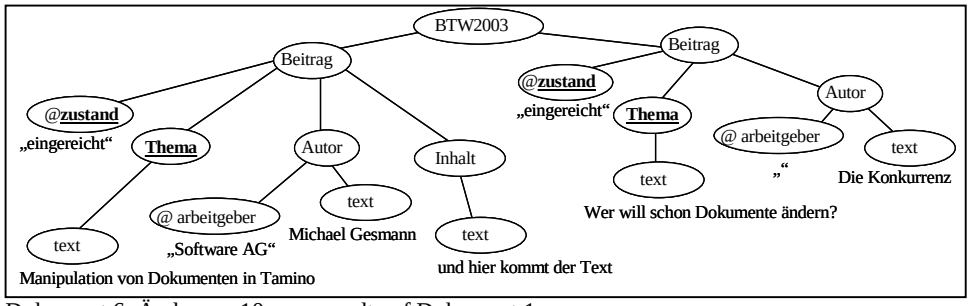
Rename

Für die Umbenennungsfunktion müssen alle Änderungsknoten Elemente, Attribute oder Knoten mit Verarbeitungsinstruktionen sein, die nach Ausführung der Funktion den angegebenen Namen haben.
Die folgende Anweisung nennt Elemente um, die über den Pfad BTW2003/Beitrag/Titel erreichbar sind, und macht Thema-Elemente daraus. Alle status-Attribute im Dokument werden in „zustand“ umbenannt.

```
update rename input()/BTW2003/Beitrag/Titel as"Thema"
       rename input()//@status as "zustand"
```

Änderung 10: Umbenennung von Elementen und Attributen

Auf Dokument 1 angewendet ergibt diese Operation folgenden Dokumentinhalt:



Dokument 6: Änderung 10 angewandt auf Dokument 1

Replace

Mit der Ersetzungsfunktion werden Änderungsknoten durch die neuen Werteknoten ersetzt. Im folgenden Beispiel wird das komplette Titel-Element durch ein "englisches" Title-Element ersetzt und der Titel selbst ebenfalls geändert:

```
update replace input()/BTW2003/Beitrag/Titel[.="Manipulation von Dokumenten in Tamino"]
            with <Title>Manipulation von Dokumenten in DBMS</Title>
```

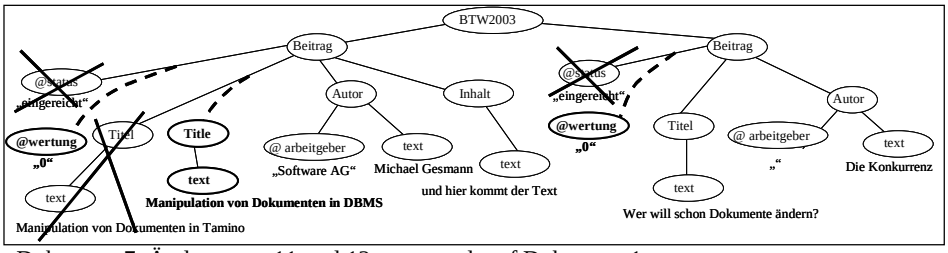
Änderung 11: Ersetzen von Elementen

Das status-Attribut wird durch ein wertung-Attribut mit dem Wert 0 ersetzt.

```
update replace input()/BTW2003/Beitrag/@status with attribute wertung{"0"}
```

Änderung 12: Ersetzen von Attributen

Wenn diese beiden Anweisungen nacheinander auf Dokument 1 angewandt werden, ergibt sich folgendes Ergebnis:



Dokument 7: Änderungen 11 und 12 angewandt auf Dokument 1

3.2 FLWUExpr

Die bisher aufgeführten Beispiele zeichnen sich dadurch aus, dass sowohl die Änderungs- als auch die Werteknoten durch einfache XPath-Ausdrücke bestimmt werden können. Diese XPath-Ausdrücke enthalten einfache Prädikate und liefern immer Knoten des Eingabedokuments. Genauso wie XPath-Anfragen auch durch XQuery-Sprachkonstrukte formuliert werden können, ist es für Änderungsoperationen möglich, mit For-Let-Where-Konstrukten zu arbeiten.

Die Titeländerung aus Abschnitt 2 läßt sich damit auch so formulieren:

```
update for $beitrag in input()/BTW2003/Beitrag
        where $beitrag/Autor = "Michael Gesmann" do
        replace $beitrag/Titel/text() with "Manipulation von Dokumenten in DBMS"
```

Änderung 13: Formulierung der Beispieländerung mit FLWUExpr

In vielen Fällen ist die Frage, ob ein Dokument mit FLWUExpr oder direkt mit den Änderungsoperationen formuliert wird, reine Geschmackssache. Darüber hinaus bieten FLWUExpr allerdings einen erweiterten Funktionsumfang. Mit den Variablenbindungen der For- und Let-Klauseln lassen sich Bedingungen formulieren, die über das hinausgehen, was XPath-Ausdrücke anbieten. Den Wert des in Änderung 12 eingefügten wertung-Attributs um 1 zu erhöhen, ist nur mit der folgenden Anweisung möglich:

```
update for $wertung in input()/BTW2003/Beitrag/@wertung do
            replace $wertung with attribute wertung {$wertung + 1}
```

Änderung 14: relative Wertveränderung von Attributen

3.3 Komplexere Szenarien

Alle bisherigen Beispiele zeichnen sich dadurch aus, daß jedem Änderungsknoten eindeutig eine einzige Änderung zugeordnet werden kann. Dies galt auch dann, wenn mehrere Elementaroperationen in einer XQuery angegeben waren. Wie Tamino Situationen behandelt, in denen dies nicht der Fall ist, wird im folgenden erläutert.

Die meisten in Abschnitt 2 vorgestellten Ansätze definieren, dass Elementaroperationen auch in der Reihenfolge ausgeführt werden sollen, in der sie angegeben werden. Dies ist hilfreich und sinnvoll, wenn Änderungen von Benutzern auf einzelnen Dokumenten durchgeführt wurden und anschließend analog in einer Datenbank nachvollzogen werden sollen. Deskriptiv beschriebenen Änderungen, die auf größeren Dokumenten oder Dokumentmengen ausgeführt werden sollen, nimmt dieser Ansatz aber auch Optimierungsmöglichkeiten. In Tamino hat die Reihenfolge, in der einzelne Änderungsoperationen angegeben werden, deshalb keine Bedeutung.

Aufgrund der Ausdrucksmächtigkeit von XQuery kann es so aber geschehen, dass ein Knoten sowohl als Änderungs- als auch als Werteknoten ausgewählt wird. Damit eindeutige Ergebnisse erzeugt werden können, wird festgelegt, dass die durchzuführenden Änderungen keinerlei Einfluß auf die Menge der Änderungsknoten und deren Inhalte haben sollen. Das gleiche gilt für die Werteknoten. Die neuen Werte werden ausschließlich auf den Ausgangsdaten berechnet, ohne von anderen Änderungen beeinflußt zu werden. Werteknoten entsprechen deshalb niemals Referenzen auf bereits existierende Knoten in Dokumenten sondern werden gegebenenfalls als neue Knoten kopiert.

Mit einer kleinen Auswahl weiterer Szenarien zeigt dieser Abschnitt, wie auch komplexere Aufgabenstellungen mit der präsentierten Spracherweiterung gelöst werden können.

Umstrukturierung von Dokumenten

Zunächst erhält jeder Beitrag eine laufende Nummer als Attribut. Weiterhin werden Titel und Autor-Elemente in einem Info-Element zusammengefaßt, wobei das Titel-Element gleichzeitig in Thema-Element umgenannt wird. Hier sind die Autor-Elemente und die Inhalte der Title-Elemente sowohl Änderungsknoten (als zu löschende Knoten) als auch Werteknoten (als einzufügende Knoten). Zunächst werden die Änderungs- und Werteknoten berechnet, ehe anschließend die eigentlichen Änderungen ausgeführt werden. So können Autor-Elemente und die Inhalte aus Titel-Elementen im neuen Info-Element eingesetzt werden.

Alle Operationen zusammen erfolgen mit Änderung 15 und liefern Dokument 8:

```
update for $beitrag in input()/BTW2003/Beitrag
        let $autor = $beitrag/Autor do (
insert attribute lfdNr="{$beitrag/position()}" into $beitrag
delete $beitrag/Titel
delete $autor
insert <Info>{$autor}<Thema>{$beitrag/Titel/text()}</Thema></Info> preceding $beitrag/*[1])
```

Änderung 15: Umstrukturierung eines Dokuments

```
<?xml version="1.0"?>
<BTW2003>
   <Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
            <Thema>Manipulation von Dokumenten in Tamino</Thema></Info>
     <Inhalt> und hier kommt der Text </Inhalt></Beitrag>
   <Beitrag lfdNr="2" status="eingereicht">
      <Info> <Autor arbeitgeber="">Die Konkurrenz</Autor>
             <Thema>Wer will schon Dokumente ändern?</Thema></Info></Beitrag>
</BTW2003>
```

Dokument 8: Änderung 15 angewandt auf Dokument 1

Zusammenfügen von Dokumenten

Nachdem die Beiträge zur Begutachtung verteilt wurden, treffen Gutachten ein, die zunächst in einer separaten Datenquelle abgelegt werden. Das Dokument mit den Gutachten sieht für die folgenden Beispiele folgendermassen aus:

```
<?xml version="1.0"?>
<BTW2003Gutachten>
   <Gutachten beitragNr="1" urteil="ausgezeichnet">
       <Gutachter arbeitgeber="Software AG">Manfred Michels</Gutachter>
       <Details> Jetzt weiß ich endlich was wir da machen </Details>    </Gutachten >
   <Gutachten beitragNr="1" urteil ="bescheiden">
      <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
      <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
</BTW2003Gutachten>
```

Dokument 9: Eingabedokument mit Gutachten

Die Gutachten aus Dokument 9 können folgendermaßen an das Ende ihrer zugehörigen Beiträge in Dokument 8 kopiert werden:

```
update for $beitrag in input()/BTW2003/Beitrag
    let $gutachten := input()/BTW2003Gutachten/Gutachten [@beitragNr = $beitrag/@lfdNr] do
        insert $gutachten into $beitrag
```

Änderung 16: Kopieren von Knoten in ein anderes Dokument

Das BTW2003-Dokument sieht danach folgendermaßen aus:

```
<?xml version="1.0"?>
<BTW2003>
   <Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
            <Thema>Manipulation von Dokumenten in Tamino</Thema>    </Info>
                  <Inhalt> und hier kommt der Text </Inhalt>
     <Gutachten beitragNr="1" urteil="ausgezeichnet">
        <Gutachter arbeitgeber="Software AG">Manfred Michels</Gutachter>
        <Details> Jetzt weiß ich endlich was wir da machen </Details>    </Gutachten >
     <Gutachten beitragNr="1" urteil ="bescheiden">
        <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
        <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
   </Beitrag>
   <Beitrag lfdNr="2" status="eingereicht">
     <Info> <Autor arbeitgeber="">Die Konkurrenz</Autor>
            <Thema>Wer will schon Dokumente ändern?</Thema>  </Info>    </Beitrag>
</BTW2003>
```

Dokument 10: Änderung 16 angewandt auf die Dokumente 8 und 9

Herausschneiden von Dokumentteilen

Abschließend werden aus Dokument 10 alle Beiträge gelöscht, zu denen kein Gutachter etwas liefern wollte/konnte. Außerdem werden alle Gutachten gelöscht, bei denen der Gutachter den gleichen Arbeitgeber hat wie einer der Autoren:

```
update for $beitrag in input()/BTW2003/Beitrag[empty(Gutachten)]
       for $gutachten in input()BTW2003/Beitrag/Gutachten
                                    [Gutachter/@arbeitgeber = ../Info/Autor/@arbeitgeber]
do ( delete $beitrag
     delete $gutachten )
```

Änderung 17: löschen mehrere Elemente

Und hier nun das Ergebnis all der Arbeit:

```
<?xml version="1.0"?>
<BTW2003>
<Beitrag lfdNr="1" status="eingereicht">
     <Info> <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
            <Thema>Manipulation von Dokumenten in Tamino</Thema>    </Info>
                  <Inhalt> und hier kommt der Text </Inhalt>
     <Gutachten beitragNr="1" urteil ="bescheiden">
        <Gutachter arbeitgeber="">Die Konkurrenz</Gutachter>
        <Details>Wer will schon Dokumente ändern?</Details>    </Gutachten >
   </Beitrag>
</BTW2003>
```

Dokument 11: Änderung 17 angewandt auf Dokument 10

3.4 Zusammenfassung

Dieser Abschnitt hat dargestellt, wie elementare Änderungsoperationen mit einfachen Spracherweiterungen in XQuery angeboten werden können. Zusammen mit der in XQuery selber vorhandenen Funktionalität zur Suche von Dokumenten und Dokumentinhalten sowie Attribut- und Elementkonstruktoren genügen die gezeigten Operationen, um die in Abschnitt 2 aufgezählte Basisfunktionalität zu realisieren. Anhand von zunächst einfachen und später auch komplexeren Beispielen wurden die Operationen erläutert. Die Reihenfolge, in der Änderungsoperationen angegeben werden, hat keinen Einfluß auf deren Ausführungsreihenfolge. Als Ausführungsmodell werden Änderungs- und Werteknoten berechnet, bevor die eigentlichen Veränderungen an den Dokumenten vorgenommen werden.

4 Angabe mehrerer Änderungen auf einem Knoten

Die im vorangegangenen Abschnitt beschriebene Unabhängigkeit der Ausführungsreihenfolge kann zu Problemen führen, wenn das Ergebnis nicht mehr eindeutig charakterisiert wird. Was darunter zu verstehen ist, und wie Tamino damit umgeht, wird in diesem Abschnitt ausgeführt.

4.1 Konflikte

Eine komplex formulierte Änderungsoperation setzt sich aus den im vorangegangenen Abschnitt erklärten Elementaroperationen "Insert", "Delete", "Rename" und "Replace" zusammen. Werteknoten sind bei Insert und Replace-Operationen immer eindeutig festgelegt. Dagegen können auf einem Änderungsknoten mehrere Elementaroperationen ausgeführt werden. In einigen Fällen ist dies mit dem oben beschriebenen Ausführungsmodell problemlos möglich, zum Beispiel wenn ein Element umbenannt wird und gleichzeitig ein neues Attribut erhält. Es können aber auch Probleme auftreten, zum Beispiel, wenn zwei Umbenennungen auf einem Knoten ausgeführt werden sollen. Aufgrund der unspezifischen Ausführungsreihenfolge steht in diesem Fall nicht mehr fest, wie das Endergebnis aussieht. Je nach interner Ausführungsreihenfolge ergäben sich unterschiedliche Ergebnisdokumente. Solche Problemfälle werden Konflikte genannt.

Ein **Konflikt** tritt dann auf, wenn

1. eine Elementaroperation ausgeführt werden soll, die aber aufgrund anderer Elementaroperationen keine Auswirkung auf das Ergebnis hat, oder wenn
2. das Ergebnis der Operationen zu einem nicht wohlgeformten Dokument führt, oder wenn
3. das Ergebnis der an einem Änderungsknoten auszuführenden Elementaroperationen nicht eindeutig ist.

Eine Erläuterung der möglichen Konflikte erfolgt zunächst für den Fall, daß der Änderungsknoten ein Element ist.

Fall 1: unsinnige Operationen
Der erste Fall tritt immer dann ein, wenn ein Knoten mit seinen Nachfolgern gelöscht oder ausgewechselt wird, gleichzeitig aber weitere Änderungsoperationen auf den aus dem Dokument entfernten Knoten definiert werden. Somit wurde mindestens eine Änderungsoperation angegeben, die als überflüssig oder unsinnig angesehen wird. Als Spezialfall stellt sich diese Situation auch immer dann ein, wenn auf einem Änderungsknoten mehr als nur eine Lösch- oder Ersetzungsoperation definiert wird.

Fall 2: nicht wohlgeformte Dokumente
Der zweite Fall tritt dann ein, wenn zwei Attribute gleichen Namens eingefügt werden sollen oder zu einem bereits existierenden Attribut ein weiteres Attribut mit gleichem Namen hinzugefügt wird. Das Einfügen von Attributen darf allerdings temporär Dokumente erzeugen, die nicht wohlgeformt sind. So ist es möglich, ein Attribut neu einzufügen, obwohl es im Element schon ein Attribut gleichen Namens gibt, wenn das bereits existierende Attribut auch gelöscht wird. Diese Eigenschaft trägt dem Gedanken Rechnung, dass keine Ausführungsreihenfolge angegeben wird. Außerdem geht sie konform damit, dass auch die Dokumentvalidierung nicht nach jeder einzelnen Änderung erfolgt, sondern das Dokument erst nach Ausführung aller Elementaroperationen wieder gültig sein muß.

Fall 3: nicht kommutative Operationen
Der dritte Fall tritt dann ein, wenn zwei Elementaroperationen auf einem Änderungsknoten nicht kommutativ sind, das heißt, das Ergebnis der Elementaroperationen von der Reihenfolge ihrer Ausführung abhängt. Zum einen ist dies der Fall bei zwei „Rename“-Operationen mit unterschiedlichen Namen, für die nicht klar ist, welchen Namen der zu ändernde Knoten am Ende hat. Ausserdem tritt diese Situation bei zwei „Insert ... into ...“-Operationen auf einem Änderungsknoten auf, die keine Attribute einfügen. Wenn jede „Insert ... into ...“-Operation einen Knoten neu einfügt, ist am Ende nicht klar, in welcher Reihenfolge diese Knoten im Ergebnis erscheinen werden. Würde nur ein „Insert ... into ...“ mit einer Sequenz der beiden Knoten angegeben, wäre die Reihenfolge eindeutig.
Was für das „Insert ... into ...“ gilt, trifft in ähnlicher Form auch für das „Insert ... preceding ...“ und „Insert ... following...“ zu. Die Reihenfolge der neu eingefügten Knoten im Änderungsknoten ist nicht festgelegt, wenn es mehrere solcher Operationen an der gleichen Stelle in der Kindliste des Änderungsknoten gibt.
Hier beachte man, dass das positionsbezogene Einfügen auf dem Elternknoten und nicht auf den in der Operation angegebenen Geschwisterknoten als Änderungsknoten definiert ist. Dadurch liefert ein solches Insert auch dann noch ein definiertes Ergebnis, wenn der Geschwisterknoten(!) aus dem Dokument gelöscht wird. Zwei „Insert ... preceding ...“ oder „Insert ... following ...“ Operationen führen nur dann zu einem Konflikt, wenn sie sich auf die gleiche Position in der Kindliste des Änderungsknotens beziehen, ansonsten können die Operationen problemfrei ausgeführt werden.

Zusammenfassung
Tabelle 2 faßt zusammen, wann Konflikte dieser Art bei Elementaroperationen an **einem** Änderungsknoten auftreten. Die Insert-Operation wird in die vier Spezialfällen unterschieden: "InsertAttr" fügt ein neues Attribut ein und ist nur bei Elementknoten möglich. "InsertInto" deckt alle übrigen Fälle der „insert ... into ...“-Anweisung ab. „InsertPrec” und „InsertFoll“ stehen für die beiden verbleibenden Formen der Insert-Anweisung. Ein Haken beschreibt den Fall, dass beide Operationen in beliebiger Reihenfolge ausgeführt werden können, ohne dass das Endergebnis davon betroffen ist. Die ① beschreibt einen Konflikt nach Fall 1, die ② einen Konflikt nach Fall 2 und die ③ einen Konflikt durch nicht kommutative Operationen. Die ④ charakterisiert Konflikte durch Operationen, die an der gleichen Stelle innerhalb der Kindliste eines Änderungsknotens ausgeführt werden sollen.

Operationen auf einem Knoten	*Delete*	*Replace*	*Rename*	*Insert Attr*	*Insert Into*	*Insert Prec*	*Insert Foll*
Delete	①	①	①	①	①	①	①
Replace		①	①	①	①	①	①
Rename			③	✓	①	✓	✓
InsertAttr				②	✓	✓	✓
InsertInto					③	✓	✓
InsertPrec						④	✓
InsertFoll							④

Tabelle 2: Konflikte bei mehreren Operationen auf einem Knoten

Weitere Aspekte
Für Änderungsoperationen auf Attributen und „Processing Instructions“ gilt ebenfalls die obige Tabelle, wobei nur die Operationen Delete, Replace und Rename möglich sind. Für alle weiteren Knoten gilt die Tabelle mit der Einschränkung auf Delete und Replace.

Wie bei Elementen kommen auch bei anderen Knoten nach Fall 1 Konflikte hinzu, wenn auf diesen Knoten ein Delete oder ein Replace definiert ist und auf irgendeinem seiner Nachfolger weitere Operationen definiert werden. Diese weiteren Operationen fallen dann in die Kategorie ①.

Konflikte sind auf Dokumenten definiert, und ob ein Konflikt tatsächlich auftritt oder nicht, hängt stark von den Dokumenten ab, auf die eine Änderungsoperation angewandt wird. Deshalb ist eine statische Bestimmung solcher Konflikte, zum Beispiel durch einen Compiler, nicht möglich. Statt dessen muss die Konflikterkennung dynamisch zur Ausführungszeit einer Änderungsoperation erfolgen. In der Konsequenz kann das Einfügen eines neuen Dokumentes dazu führen, dass eine Änderungsoperation auf einen Konflikt stößt, oder umgekehrt kann das Löschen eines Dokuments dazu führen, dass eine Operation konfliktfrei ausgeführt werden kann.

4.2 Beispiel und Konfliktbehandlung

Anhand eines einfachen Beispiels sei ein Konflikt erläutert. Dokument 10 diene hier als Eingabe und es werde die Änderung 18 ausgeführt.

```
<?xml version="1.0"?>
<BTW2003>
    <Beitrag lfdNr="1" status="eingereicht">
        <Autor arbeitgeber="Software AG">Michael Gesmann</Autor>
        <Thema>Manipulation von Dokumenten in Tamino</Thema>
        <Inhalt> und hier kommt der Text </Inhalt>
       <Gutachten urteil="abgelehnt">
           <Gutachter>Konkurrenz</Gutachter>
       </Gutachten>
    </Beitrag>
    <Beitrag lfdNr="2" status="eingereicht">
        <Autor arbeitgeber="">Die Konkurrenz</Autor>
        <Thema>Wer will schon Dokumente ändern?</Thema>
        <Gutachten urteil="unentschieden">
           <Gutachter></Gutachter>
        </Gutachten>
    </Beitrag>
</BTW2003>
```

Dokument 10: Eingabedokument für folgende Beispiele

```
update for $beitrag in input()/BTW2003/Beitrag[Gutachten] do
        ( replace $beitrag/@status with attribute status {"beurteilt"}
          delete $beitrag/Gutachten[Gutachter=""]
          rename$beitrag//@urteil as bewertung )
```

Änderung 18: Beispiel, mit dem Konflikte erzeugt werden

Für den ersten Beitrag bekommt das status-Attribut einen neuen Wert, es wird kein Knoten gelöscht und ein urteil-Attribut umgenannt. Für den zweiten Beitrag wird ebenfalls der Wert des status-Attributs verändert: es soll das darin enthaltene Gutachten gelöscht werden, und das in dem Gutachten enthaltene urteil-Attribut soll umbenannt werden. Diese letzte Umbenennung ist unnötig, da das umgebende Element sogar gelöscht werden soll. Diese Situation führt zu einem Konflikt.

Konflikterkennung auf der Basis der oben vorgegebenen Definition für Konflikte zu betreiben, ist ein sehr strikter Ansatz, der davon ausgeht, dass alle Änderungsoperationen so eindeutig abgefaßt werden, dass letztendlich keine Widersprüche oder Ungenauigkeiten zur Ausführungszeit mehr auftreten. Wenn solche Konflikte auftreten, dann hat die Anwendung anscheinend noch nicht vollständig die Reichweite der Operation erfaßt. In diesen Fällen lehnt Tamino die Änderungsoperation ab.

5 Zusammenfassung und Ausblick

Dieser Beitrag ist der Frage nachgegangen, wie Änderungen in XML-Dokumenten formuliert werden können, damit sie in einer Datenbank ausgeführt werden können. Zunächst wurden bestehende Ansätze beschrieben und bewertet. Als ausdrucksstärkster Ansatz wurden Erweiterungen von XQuery angesehen. Dieser Ansatz wurde für eine Realisierung im XML-DBMS Tamino der Software AG ausgewählt. Mit den dort implementierten Erweiterungen von XQuery entsteht eine Sprache, die sowohl die Änderungsoperationen selber, die Suche nach zu verändernden Knoten als auch die Bestimmung der neuen Werte einheitlich beschreiben kann. Eine Änderungsoperation kann sich aus mehreren Elementaroperationen zusammensetzen. Die Reihenfolge, in der die Elementaroperationen angegeben werden, hat in Tamino keinen Einfluß auf das Ergebnis. Die Spracherweiterungen und deren Semantik wurde anhand zahlreicher Beispiele dargestellt. Der Beitrag zeigt, wie diese Forderung nach Reihenfolge-unabhängigkeit zu Konflikten führen kann. Konfliktbehaftete Änderungsoperationen werden mit einer Fehlermeldung abgelehnt.
Die praktische Erprobung der Konzepte durch Anwendungen steht noch aus. Insbesondere die Behandlung von Konflikten muss dort ihre Praxistauglichkeit zeigen. Ferner läßt die vorgestellte Realisierung eine Reihe offener Fragen, die weiter untersucht werden müssen:

- [Le01] hat in seiner Spracherweiterung mit einem if-then-else-Konstrukt auch die bedingte Ausführung von Änderungsoperationen vorgesehen. Dieses wurde bisher nicht berücksichtigt.
- Mit dem Dokument-Konstruktor von XQuery können auch neue Dokumente eingefügt werden. Das Löschen eines Dokumentknotens führt zum Löschen des ganzen Dokuments. Es ist noch unklar, inwieweit sich diese Erweiterungen eignen, Systemschnittstellen für das Einfügen und Löschen von Dokumenten zu ersetzen.
- Die Konflikterkennung wie sie in Abschnitt 4 beschrieben wurde, ist sehr restriktiv. Es könnten mehr Freiheiten zugelassen werden, indem mehr Anwendungssemantik ausgenutzt wird. So muß es nicht unbedingt ein Fehler sein, wenn Änderungen auf Knoten definiert sind, die auch gelöscht werden. Es ist genauso gut denkbar, den Netto-Effekt, in diesem Fall das Löschen, zu berechnen und nur diese Änderung durchzuführen. Zur Zeit geht Tamino den restriktiven Weg in der Annahme, dass zukünftigen Versionen an dieser Stelle im oben genannten Sinne geöffnet werden können. Hingegen ist es fast unmöglich, einmal angebotene Funktionalität wieder herauszunehmen. Hier muß auch abgewartet werden, was spätere W3C Empfehlungen festlegen werden.
- In manchen Anwendungen spielt die Reihenfolge, in der Änderungen ausgeführt werden sollen, eine wesentliche Rolle. Ein Beispiel hierfür hat Abschnitt 3.3 gezeigt. Dort wurden zuerst Gutachten zu den BTW-Beiträgen kopiert, ehe anschließend die Beiträge gelöscht wurden, zu denen es keine Gutachten gab. Eine solche Ausführungsreihenfolge kann mit der präsentierten Spracherweiterung zur Zeit nicht in einer Anweisung formuliert werden. Statt dessen müssen entweder sehr komplexe Prädikate für die einzelnen Operationen definiert werden, oder die einzelnen Änderungsoperationen können nur als individuelle Update-Anweisungen unabhängig voneinander in der gewünschten Reihenfolge an das System gerichtet werden. Diese Forderung nach einer benutzerdefinierten Reihenfolge der Operationen steht zunächst

im Widerspruch zur systembestimmten Ausführung der Elementaroperationen. Wie die Sprache erweitert werden kann, so dass auch eine Sequenz von Operationen in einer einzigen Anweisung ausgeführt werden kann, ist bisher nicht klar.

- Aus Platzgründen läßt dieser Beitrag alle Fragen hinsichtlich der Implementierung der aufgeführten Schnittstelle offen. Hier sei lediglich auf zwei prominente Problembereiche hingewiesen. Es ist keine triviale Aufgabe, Dokumente nur partiell zu validieren, wenn Konsistenzbedingungen wie zum Beispiel Eindeutigkeit von Werten in gewissen Dokumentteilen oder Knoten vom Typ ID bzw. IDREF berücksichtigt werden müssen. Ähnliches gilt je nach gewählter Indeximplementierung auch für partielle Änderungen in Indexen.

Literaturverzeichnis

[Db2] IBM DB2 Universal Database, XML Extender - Administration and Programming, Version 7, ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2sxe70.pdf

[Le01] Lehti, Patrick: "Design and Implementation of a Data Manipulation Processor for an XML Query Language", August 2001, Diplomarbeit an der Technischen Universität Darmstadt - FB Elektrotechnik und Informationstechnik

[Ora9i2] Oracle9i Database Online Documentation Release 2, http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/index.htm

[QuiP00] Quip Version 2.2.1, Software AG's XQuery prototype, http://developer.softwareag.com/tamino/quip

[SiXDML02] "Simple XML Data Manipulation Language Draft", Working Draft 2002-06-23, http://www.xmldb.org/sixdml/sixdml-lang.html

[SQLServ2000] Using UpdateGrams to Modify Data, SQLXML 3.0, http://msdn.microsoft.com/library/default.asp?url=/library/enus/sqlxml3/htm/updategram_5kkh.asp

[Tam41] Dokumentation zu Tamino Version 4.1, (noch nicht erschienen)

[TIHW01] Tatarinov, Ives; Halevy, Weld: "Updating XML", Proceedings ACM SIGMOD 2001, pp. 413-424, May21-24, Santa Barbara, California, USA

[XIS312] eXcelon Product Information, Extensible Information Server Version 3.1.2, http://support.exceloncorp.com/doc/full/xml/xis312/webhelp/m_upwiz.htm

[XNs99] Namespaces in XML, Empfehlung des W3C, http://www.w3.org/TR/REC-xml-names

[XP99] XML Path Language (XPath), Empfehlung des W3C, http://www.w3.org/TR/xpath

[XQ02] XML Query, Arbeitsmaterial der “XML Query Working Group” im W3C, http://www.w3.org/XML/Query

[XSD01] XML Schema Empfehlung des W3C, http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1, http://www.w3.org/TR/xmlschema-2

[XUp02] "XUpdate - XML Update Language", Working Draft 2000-09-14, http://www.xmldb.org/xupdate/xupdate-wd.html

[Xup02R] "Requirements for XML Update Language", Working Draft der XML:DB Initiative 2000-11-24, http://www.xmldb.org/xupdate/xupdate-req.htm