Automatisierte Umsetzung von komplexen XML-Schemaänderungen

Michael Hartung

Abteilung Datenbanken
Universität Leipzig
Postfach 100920
04009 Leipzig
hartung@izbi.uni-leipzig.de

Abstract: Dieser Beitrag untersucht die Frage, wie komplexe Änderungen bei der Evolution von XML-Schemas automatisiert unterstützt werden können. Hierzu werden mögliche Änderungen an XML-Schemas durch Evolutionsoperatoren beschrieben, klassifiziert und beurteilt. Im Speziellen wird die Verschiebung (Move) von Elementen innerhalb von XML-Schemas analysiert. Die automatisierte Generierung und Ausführung von Transformationsregeln zur Migration von Instanzdaten und die Beurteilung möglicher Informationsverluste während einer Transformation wird allgemein, wie auch anhand von Publikationsdaten untersucht.

1 Einführung

Schemaevolution bezeichnet den Vorgang einer Änderung bereits existierender, in Betrieb befindlicher Schemas, welche vorwiegend komplexe Strukturen wie Datenbanken, Kommunikationsprotokolle, Applikationsschnittstellen oder Austauschformate beschreiben. Typische Anwendungsgebiete sind z.B. objekt-relationale Datenbankschemas, ER oder UML Modelle, Ontologien, Schemas/Beschreibungen im Bereich XML und Webservices oder Workflowbeschreibungen. In gleicher Weise sind Kommunikations- und Serviceschnittstellen in Grids betroffen. Die Ausführung einer Evolution von Schemas gründet sich insbesondere auf veränderte bzw. neue Anforderungen von Applikationen oder das Beheben von Entwurfsfehlern (error fixing). Die Evolution eines Schemas zieht in der Regel Änderungen in dem vom Schema abhängigen Strukturen nach sich. Hierzu zählen bspw. Sichten, Instanzdaten oder Applikationen, welche ein betroffenes Schema nutzen.

Auch für die Evolution von XML-Schemas kommt der korrekten Transformation bzw. Migration von Instanzdaten eine wichtige Rolle zu, wobei diese Aufgabe möglichst automatisch gelöst werden sollte. In [Bo04, KMH05, Su01] wurden zunächst Änderungen direkt auf den Daten vollzogen, z.B. das Einfügen eines neuen Elements in ein Instanzdokument. Bei einer nicht bzgl. des Schemas validen Änderung wird dieses entsprechend erweitert, um eine fortwährende Validität der geänderten Instanzen gegenüber dem Schema zu gewähren. Damit haben Schemas primär einen beschreibenden Charak-

ter und werden den Instanzen angepasst. Die auch in diesem Beitrag verfolgte, konventionellere Vorgehensweise nimmt dagegen zunächst Anpassungen am Schema vor, welche dann auf die zugehörigen Instanzdaten zu übertragen sind.

Die Transformation von Instanzdaten bei der Überführung von einem Quell- in ein Zielschema wurde für allgemeine Fälle in [Po02], wie auch für semistrukturierte Daten in [AW05, He02] diskutiert. Für die Migration von XML-Daten sind erste kommerzielle Tools verfügbar [Al06, St06]. Ziel der Ansätze bzw. Tools ist es, die aus einem Matching hervorgegangenen bzw. durch den Nutzer spezifizierten Korrespondenzen zwischen den Elementen der beiden Schemas in ein Transformationsmapping für Instanzdaten, z.B. in Form von XQuery oder XSLT bei XML-Daten, zu übersetzen. Eine Evaluierung verfügbarer Systeme zur Transformation von Instanzdaten wurde in [Le05] durchgeführt. Bisherige Arbeiten [GMR05, Su01, Ti01, Ze05] adressieren primär einfache Schemaänderungen, während komplexe Evolutionen allenfalls als Folge einfacher Evolutionsschritte erfasst werden. Außerdem finden sich bisher wenig Angaben zur automatisierten Umsetzung von XML-Schemaänderungen.

Folglich soll in diesem Beitrag die automatisierte Umsetzung von Schemaänderungen basierend auf Evolutionsoperatoren für die Transformation von Instanzdaten untersucht werden. Hierzu führen wir eine Klassifikation von Operatoren zur XML-Schemaevolution ein und diskutieren deren Auswirkungen auf die sogenannte Informationskapazität. Speziellere Betrachtungen widmen sich dem Move-Operator und der automatisierten Bestimmung von XQuery-Transformationen zur Datenmigration. Hierbei untersuchen wir inwieweit eine Informationserhaltung realisierbar ist, um eine Umkehrbarkeit von Move-Änderungen zu unterstützen.

Der Rest des Beitrags gliedert sich wie folgt. Im nächsten Abschnitt wird das zugrundeliegende Modell für die Änderung der Informationskapazität definiert, anschließend werden mögliche Operatoren für die Schemaevolution von XML klassifiziert und kurz dargestellt. Als Beispiel einer komplexen Änderung wird der Move-Operator für die ebenenübergreifende Verschiebung von Elementen in XML-Strukturen in Abschnitt 3 allgemein und anhand von praktischen Beispielen, Schemaevolution auf semistrukturierten Publikationsdaten, erläutert und analysiert. Insbesondere wird die Realisierung eines informationserhaltenden Move-Operators präsentiert. Eine Zusammenfassung und ein Ausblick bilden den Abschluss des Beitrags.

2 Überblick und Klassifikation von Operatoren

Für die Klassifikation von Änderungsoperatoren für XML-Schema stehen zwei Möglichkeiten der Einteilung zur Verfügung. Auf der einen Seite ist eine Kategorisierung nach dem Typ des Operators möglich. Allgemein können in einem Schema Komponenten *hinzugefügt (Add)*, *gelöscht (Delete)* oder *verändert (Change)* werden, Constraints werden dagegen i.a. nur *verändert (Change)*. Auf der anderen Seite können die Operatoren bzgl. der Änderung der Informationskapazität (s.u.), d.h. nach Auswirkungen auf mögliche Instanzdaten, gruppiert werden. Auf Basis des in 2.1 definierten Begriffs der Informationskapazität wird zwischen *informationserhaltenden*, *informationsreduzieren-*

den, *informationserweiternden* und *informationsverändernden* Operatoren unterschieden. Abb. 1 zeigt den Zusammenhang zwischen den beiden Kategorisierungsmöglichkeiten.

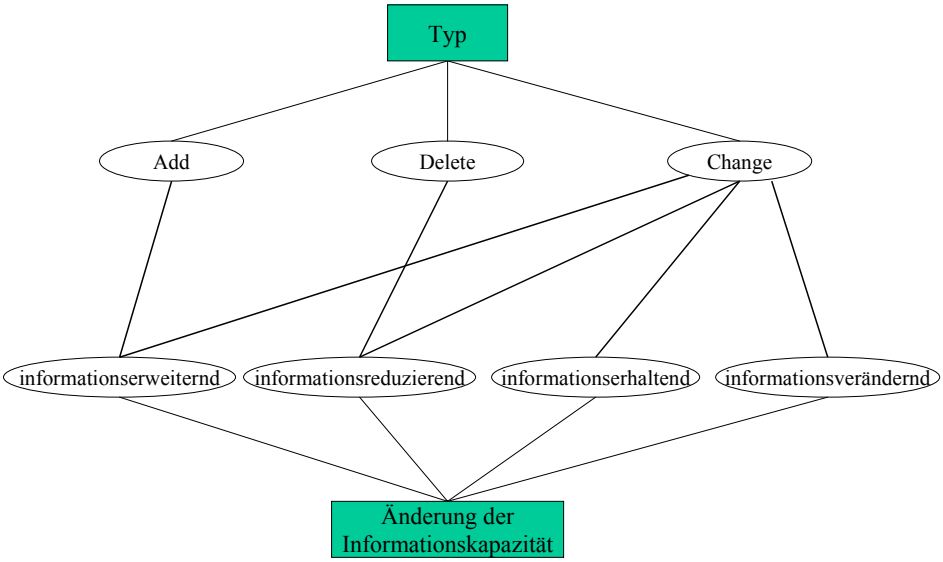

Abbildung 1: Zusammenhang zwischen Typ des Operators und Änderung der Informationskapazität

2.1 Änderungen der Informationskapazität

Der in diesem Beitrag zur Klassifikation der Operatoren verwendete Begriff der Änderung der Informationskapazität wurde in [Hu84] für relationale Daten definiert und auch in [KMH05] für die Evolution von XML-Dokumenten genutzt.

Hull führt den Begriff der Dominanz zwischen einem Schema S und einem Schema T, kurz $S \leq T$, ein. Hierzu wird eine Mappingfunktion

$u: S \rightarrow T$

und deren Umkehrung

$v: T \rightarrow S$

zwischen den beiden Schemata definiert, wobei u Instanzen I(S) über S in Instanzen I(T) über T transformiert. Bei der Umkehrung werden mittels v Instanzen I(T) über dem Zielschema in Instanzen I(S) über dem Ausgangsschema konvertiert. Die Dominanz zwischen S und T ($S \leq T$) wird basierend auf u und v und den Instanzen I(S) über S wie folgt definiert:

$$S \leq T \leftrightarrow v(u(I(S))) = I(S),$$

d.h. T dominiert S, falls es eine Komposition von u und v gibt, welche Instanzen I(S) nach T migriert und anschließend in die gleichen Ausgangsinstanzen über S zurücktransformiert. Anders gesprochen ist Schema T mindestens so mächtig wie S, falls $S \leq T$ gilt.

Aufbauend auf dieser Definition können vier Kategorien für die Veränderung der Informationskapazität beschrieben werden.
Bei einer Schemaevolution bewirkt die Anwendung eines Evolutionsoperators die Überführung eines Ausgangsschemas S in ein Zielschema T. Wendet man die obig dargestellte Definition von Dominanz in beide Richtungen an, können vier Fälle der Veränderung der Informationskapazität unterschieden werden.

informationserweiternd: $S \leq T$ und $\neg(T \leq S)$
informationsreduzierend: $T \leq S$ und $\neg(S \leq T)$
informationserhaltend: $S \leq T$ und $T \leq S$
informationsverändernd: $\neg(S \leq T)$ und $\neg(T \leq S)$

In den nächsten Abschnitten werden mögliche Evolutionsoperatoren auf Basis des dargestellten Kapazitätsmodells und der einfachen Kategorisierung nach Typ (Add, Delete, Change) klassifiziert und erläutert. Eine kompakte Übersicht über alle klassifizierten Operatoren ist in Abb. 2 dargestellt. Von besonderer praktischer Bedeutung für die Schemaevolution sind informationserhaltende und informationserweiternde Änderungen, um Kompatibilitätsprobleme und Informationsverlust zu vermeiden.

2.2 Add-Operatoren

Bei Add-Operatoren liegt grundsätzlich eine Informationserweiterung vor. Eine typische Änderung ist die Addition von Komponenten, d.h. das Einfügen von Elementen oder Attributen in ein existierendes Schema. Wird bspw. ein obligatorisches Element eingefügt, so muss ebenfalls eine Einfügung von Elementen auf Instanzebene erfolgen. Werte für diese Elemente werden aus Metadaten, z.B. Angaben für Defaultwerte, generiert oder in Interaktion mit dem Anwender erfragt. Bei der Instanztransformation mit XQuery kann über einen Element- bzw. Attributkonstruktor das betroffene Element im Zieldokument eingefügt werden. Einfacher gestaltet sich die Addition optionaler Elemente, da hier aufgrund der Kardinalität keine Anpassung von Instanzen nötig ist. Eine mögliche Rücktransformation würde die neu hinzugefügten Komponenten ausblenden und somit die Instanzdaten in das ursprüngliche Schema ohne Verluste überführen.

2.3 Delete-Operatoren

Bei den Delete-Operatoren verhält es sich umgekehrt zu den Add-Operatoren, d.h. das Löschen einer Komponente hat stets eine informationsreduzierende Wirkung. Das Entfernen von Elementen bzw. Attributen kann bei der Instanztransformation durch Ignorierung der entsprechenden Komponente im XQuery-Ausdruck erreicht werden. Aufgrund der informationsreduzierenden Wirkung ist eine verlustfreie Rücktransformation in das Ausgangsschema nicht möglich.

2.4 Change-Operatoren

Die interessanteste Klasse von Schemaänderungen bilden die Change-Operatoren. Die in dieser Kategorie befindlichen Operatoren können sowohl eine informationserhaltende, informationsverändernde, informationsreduzierende als auch informationserweiternde Wirkung besitzen. Im Gegensatz zu den beiden vorherigen Klassen muss bei Change-Operatoren zwischen Komponenten und Constraints unterschieden werden.

Operator	*Art*	*Beschreibung*	*Änderung der Informationskapazität*
Add (Komponenten)	Add	Hinzufügen von Komponenten (Elemente bzw. Attribute)	erweiternd
Delete (Komponenten)	Delete	Löschen von Komponenten (Elemente bzw. Attribute)	reduzierend
Constraints ändern	Change	Ändern von Constraints (Kardinalitäten, Einschränkungen, usw.)	erweiternd, reduzierend, erhaltend, verändernd
Renaming	Change	Bezeichner von Komponenten ändern	erhaltend
Typänderung	Change	Änderung des Typs einer Komponente	reduzierend, erweiternd
Konversionen	Change	Konversion eines Elements in ein Attribut und umgekehrt	erhaltend
Fusion/Aggregation	Change	Aggregieren/Fusionieren von Elementen	erhaltend, reduzierend
Nest	Change	Nesten von Komponenten	erhaltend
Unnest	Change	Entnesten von Komponenten	reduzierend
Einfacher Move	Change	Elemente auf einer Ebene verschieben	erhaltend
Ebenenübergreifender Move	Change	Ebenenübergreifende Verschiebung von Komponenten	erhaltend, reduzierend

Abbildung 2: Überblick über mögliche Evolutionsoperatoren

Eine Umbenennung (Rename) einer Komponente ist eine einfache Operation mit stets informationserhaltender Wirkung. Gleiches gilt für die Konversion zwischen Attributen und Elementen einer Ebene. Sowohl Umbenennungen als auch Konversionen sind einfach über XQuery sowohl für die Hin- als auch für die Rücktransformation von Instanzdaten beschreibbar. Anders verhält es sich mit den so genannten Aggregations- bzw. Fusionsoperatoren. Diese können je nach Aussehen der zugrundeliegenden Aggregationsfunktion eine informationsreduzierende bzw. informationserhaltende Wirkung besitzen. Beispielsweise ist eine Aggregation von Monatssummen auf Jahressummen informationsreduzierend. Hingegen ist die Fusionierung von Adresselementen (Anschrift,

PLZ, Ort) in ein einziges Adress-Element unter Nutzung einer umkehrbaren Fusionsfunktion informationserhaltend.

Constraint-Änderungen können vielfältige Auswirkungen auf die Informationskapazität mit sich bringen. Beispiele hierfür lassen sich anhand der Kardinalitätsconstraints finden. Wird beispielsweise die Untergrenze der Kardinalität erhöht, ergibt sich eine Informationsreduktion, da sich die Anzahl möglicher Werte reduziert. Hingegen hat eine Erhöhung der Obergrenze eine informationserweiternde Wirkung. Eine Änderung an der Obergrenze als auch an der Untergrenze, z.B. [1,2] → [3,4], kann auch informationsverändernde Wirkung haben. Auch bei Änderungen an Typen muss zwischen Erweiterung und Reduzierung basierend auf dem Ausgangs- und Zieltyp der Konversion unterschieden werden. Änderungen an Constraints und Typen werden auf Schemaebene geprüft und in Fällen von Informationsreduktion (Löschung betroffener Elemente) oder Informationserweiterung (Hinzufügen fehlender Elemente) in XQuery-Ausdrücke zur Transformation der Instanzdaten umgesetzt.

Neben den Constraint-Änderungen sind Verschiebungen von Elementen innerhalb der XML-Struktur die interessantesten Fälle einer Evolution von XML-Schemas. Aufgrund der Komplexität wird bei Move-Operatoren zwischen Elementverschiebungen auf einer Ebene und ebenenübergreifenden Verschiebungen unterschieden. Ein Move auf identischer Ebene, d.h. ein Element ändert seine Position innerhalb seiner Ebene, ist stets informationserhaltend. In XQuery ist diese Änderung durch eine einfache, umkehrbare Umordnung (Permutation) der Elemente beschreibbar. Schwieriger zu beurteilen und zu realisieren sind ebenenübergreifende Änderungen wie das Nesting/Unnesting von Elementen bzw. die Verschiebung von Elementen über Ebenen hinweg. Bei Nest/Unnest-Operatoren spielt die Ausführungsrichtung eine wichtige Rolle. Beispielsweise kann ein Nest mit Umkehroperation Unnest als informationserhaltende Änderung eingestuft werden, hingegen ist die Reihenfolge Unnest – Nest allgemein nicht informationserhaltend. Die ebenenübergreifenden Verschiebungen von XML-Komponenten sollen im nächsten Abschnitt dargestellt und näher untersucht werden.

3 Allgemeiner Move-Operator

In diesem Abschnitt soll der Move-Operator zur ebenenübergreifenden Verschiebung von Elementen innerhalb einer XML-Struktur allgemein dargestellt und analysiert werden. Eine ebenenübergreifende Verschiebung von Elementen im Rahmen einer Schemaevolution wurde in bisherigen Arbeiten kaum adressiert. Mit Hinblick auf die Versionierung geänderter XML-Schemas, welche eine bessere Unterstützung abhängiger Anwendungen ermöglicht, ist ebenfalls die Gestaltung eines informationserhaltenden Move-Operators von enormer Bedeutung. Beispielsweise können Anwendungen, welche eine ältere Version eines Schemas unterstützen mit Hilfe einer Rücktransformation die Instanzdaten des neuen Schema nutzen und müssen nicht manuell an das geänderte Schema angepasst werden.

In den folgenden Abschnitten werden Ausgangssituation und Bedingungen für die Durchführung des Operators geschildert, anschließend wird ein Algorithmus zur Erzeu-

gung einer umkehrbaren Transformation der Instanzdaten in XQuery präsentiert. Basierend auf den generierten Transformationen werden Untersuchungen bzgl. Änderung des Informationskapazität durchgeführt und praktisch auf die Evolution von Publikationsdaten angewandt.

3.1 Ausgangssituation und Beschreibung des Move-Operators

Im allgemeinen wird von einem Ausgangsschema, wie in Abb. 3 links dargestellt, ausgegangen. Die Verschiebung von Elementen wird dabei über einen Operator

Move-Up (c, t, pos)

realisiert. Generell existiert ein Topelement *t*, welches einerseits einem Elternelement *p* zugeordnet ist und andererseits über mehrere Kindelemente c_1,..., c_n verfügt. Ziel der Move-Up-Operation ist es, ein konkretes Kindelement c_i ($1 \leq i \leq n$) über *t* zu verschieben und resultierende Umänderungen/Transformationen in den Instanzen mittels XQuery zu beschreiben. Der Operator stellt dazu die Parameter c (*child*-Element, welches verschoben wird) und t (*top*-Element, über das Verschoben wird) bereit. Die Position des *top*-Elements in Relation zu den Kindelementen von *c* wird mit Hilfe des Parameters *pos* beschrieben. Das Zielschema nach der Evolution durch den Move-Up-Operator ist ebenfalls in Abb. 3 (rechts) dargestellt.

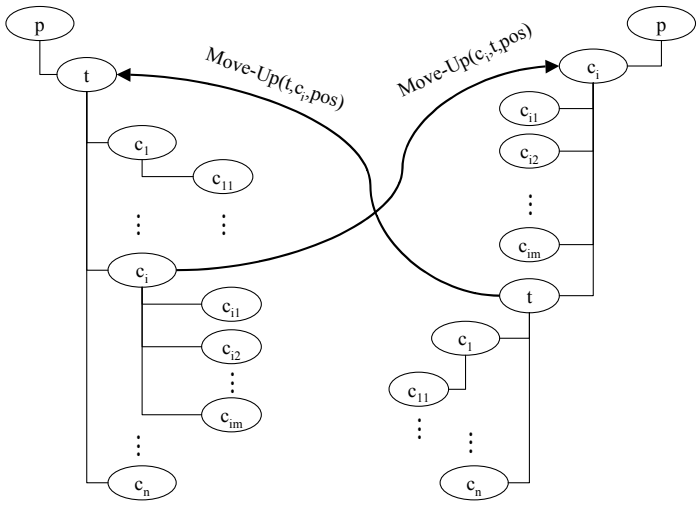

Abbildung 3: Allgemeine Situation bei Durchführung der Move-Up-Operation

Die Rücktransformation vom Zielschema zum Ausgangsschema kann ebenfalls durch den Move-Up-Operator erreicht werden, wobei die beiden Elementparameter des Operators getauscht werden; der Positionsparameter wird mit *i*, der ursprünglichen Position des Kindelements c_i, belegt. Diese Information kann aus dem Ausgangschema ermittelt werden. Das zuvor über *t* nach oben gezogene Element c_i und seine Kinder werden im Umkehrfall nach unten verschoben und wieder dem eigentlichen Topelement *t* untergeordnet. Das Topelement *t* erhält seinen alten Platz und seine Kindelemente c_1,..., c_n.

Sowohl die Evolution vom Ausgangs- zum Zielschema als auch deren Umkehrung werden über ein und denselben Operator beschrieben.

Wichtig für die Verschiebung ist die semantische Beziehung zwischen dem Topelement *t* und dem zu verschiebenden Kindelement c_i. Durch die Zuordnung „*t* besitzt c_i“ wird eine Beziehung zwischen den beiden Elementen beschrieben. Ein entscheidender Parameter dabei ist die Kardinalität der Beziehung zwischen *t* und c_i. Diese kann zwischen 1:1, 1:n und m:n variieren und beeinflusst, inwieweit Informationsverluste bei der Transformation von Instanzdaten auftreten können. Durch die Zuordnung von c_i-Elementen zu *t* können Instanzen von c_i mehrfach, d.h. innerhalb verschiedener *t*-Instanzen auftreten. Beispielsweise kann ein Autor an mehreren, in einem XML-Dokument erfassten Publikationen beteiligt sein. Die Berechnung der duplikatfreien Menge von c_i-Elementen wird demnach einer der ersten Teilschritte der Transformation mittels XQuery sein.

Neben der Beziehung zwischen Topelement *t* und Kindelement c_i auf Schemaebene ist der Fall „Topelement besitzt kein Kindelement c_i“ in den Instanzdaten gesondert zu behandeln. Eine Transformation, welche nur die Beziehung des Kindelements zu seinem Topelement ausnutzt, würde genau solche Topelemente bei der Transformation zum geänderten Schema ignorieren und damit einen Informationsverlust verursachen. Im folgenden Algorithmus zur Erzeugung der XQuery aus den Parametern des Operators wird dieser Fall separat behandelt.

3.2 Generierung der XQuery für beide Transformationsrichtungen (Basisfall)

Zur Erzeugung der XQuery für die Hin- als auch für die Rücktransformation stehen die Parameter *child*, *top* und *posOfTop* des Operators und zusätzliche Informationen des Schemas zur Verfügung. Der folgende Algorithmus kann zur Generierung der XQuery-Ausdrücke für beide Transformationsrichtungen genutzt werden. Für die Erzeugung der Rücktransformation müssen die beiden Parameter *child* und *top* ausgetauscht werden, die ursprüngliche Position des *child*-Elements unter dem *top*-Element wird als *posOfTop* verwendet.

Im ersten Schritt wird, wie bereits in 3.1 angedeutet, die duplikatfreie Menge von *child*-Elementen über dem ganzen XML-Dokument bestimmt. Dazu werden in XQuery über einen XPath-Ausdruck alle *child*-Elemente extrahiert und der Funktion ‚xqf:distinct-deep’ [Xq06] zur Duplikateliminierung von Elementen übergeben. ‚xqf:distinct-deep’ nutzt die in XQuery bereits vorhandene Funktion ‚fn:deep-equal’ aus, um die Elemente der Eingabesequenz miteinander zu vergleichen. In einem Sequenzdurchlauf werden Duplikate mittels Aufruf von ‚fn:deep-equal’ ermittelt und entfernt, das Ergebnis ist die gewünschte duplikatfreie Menge von Elementen.

Im nächsten Schritt wird die gefundene Menge von *child*-Elementen iterativ durchlaufen (Abb. 4), wobei in jeder Runde ein Abgleich mit allen *top*-Elementen durchgeführt wird. Ist ein *child* Kindelement des aktuellen *top*-Elements wird dieses dem *child*-Element an der Position *posOfTop* untergeordnet. Die Kindelemente des *top*-Elements werden außer *child* eins zu eins in das Zieldokument übernommen. Auch alle bisher zu *child* gehörigen Kindelemente bleiben erhalten.

In einem weiteren Schritt werden die *top*-Elemente ohne *child*-Element über einen XPath-Ausdruck herausgefiltert und einem besonderen *child*-Element mit xsi:nil="true" annotierten Subelementen untergeordnet. Durch diese Ausnahmebehandlung bleiben die *top*-Elemente ohne *child*-Element erhalten, d.h. sie werden informationserhaltend in das Zielschema überführt. Das Ergebnis ist eine *child*-orientierte Sicht auf die im XML-Dokument enthaltenen Daten.

```
Move-Up(#child:string, #top:string, #posOfTop:integer, #S:schema)

#S → #childsOfTop:listOfString, #childsOfChild:listOfString,
#indexOfChild:integer

let $doc = doc(...)

for $child in xqf:distinct-deep($doc//#child)
        return <#child>
                FOR i=1 TO #posOfTop-1 DO
                        {$child/#childsOfChild[i]}
                {for $top in $doc//#top[fn:not(*/@xsi:nil="true")]
                        for $child2 in $top/#child
                        where fn:deep-equal($child,$child2)
                        return <#top>
                           FOR i=1,i!=#indexOfChild TO
                           #childsOfTop.length DO  {$top/#childsOfTop[i]}
                        </#top>
                }
                FOR i=#posOfTop TO #childsOfChild.length DO
                        {$child/#childsOfChild[i]}
        </#child>

let $top := $doc//#top[count(#child)=0]
return if count($top) then
        <#child>
        FOR i=1 TO #posOfTop-1 DO
                {<#childsOfChild[i] xsi:nil="true"/>}
        {$top}
        FOR i=#pos_of_top TO #childsOfChild.length DO
                {<#childsOfChild[i] xsi:nil="true"/>}
        </#child>
```

Abbildung 4: Basisalgorithmus zur Generierung der Transformationsausdrücke

Der dargestellte Algorithmus zur Erzeugung der XQuery-Transformationen soll im nächsten Abschnitt auf die Evolution von semistrukturierten Publikationsdaten angewendet werden.

3.3 Evolution von semistrukturierten Publikationsdaten

Bei den verwendeten Publikationsdaten handelt es sich um bibliographische Daten aus Tagungsbändern von Konferenzen, welche das in Abb. 5 links dargestellte Schema aufweisen. Das Ausgangsschema für die Evolution ist publikationsorientiert, d.h. ein mögliches Instanzdokument listet vorhandene Publikationen nacheinander auf, zu jeder Publikation werden Titel, Autoren, Jahr und Buchtitel vermerkt. Als konkrete Instanzdaten für

Transformationen wurden BTW-Tagungsdaten aus der Trierer Informatik-Bibliographie DBLP verwendet.

3.3.1 Evolution zum zeitorientierten Schema

Bei der Evolution des publikationsorientierten Schema zum zeitorientierten Schema (Abb. 5) liegt eine 1:n-Beziehung zwischen Jahr und Publikation zugrunde. In einem Jahr einer Tagung werden mehrere Publikationen erfasst, während eine Publikation genau einem Jahr zugeordnet ist. Der Move-Operator wird wie folgt angewendet: Move-Up(*year*, *pub*, *2*), d.h. das Element *year* wird über das Element *pub* gezogen, um ein zeitorientiertes Schema für Publikationen zu erzeugen. Die Umkehrung der Evolution wird über den Operator Move-Up(*pub*, *year*, *3*) beschrieben. Mit Hilfe des in 3.2 dargestellten Algorithmus werden die XQuery-Ausdrücke für beide Transformationsrichtungen erzeugt. Die Komposition der beiden Ausdrücke über publikationsorientierte Instanzdaten, d.h. Hintransformation zum zeitorientierten Schema und Rücktransformation zum Ausgangsschema, erzeugt die ursprünglichen Instanzdaten. Es liegt demnach eine Informationserhaltung vor.

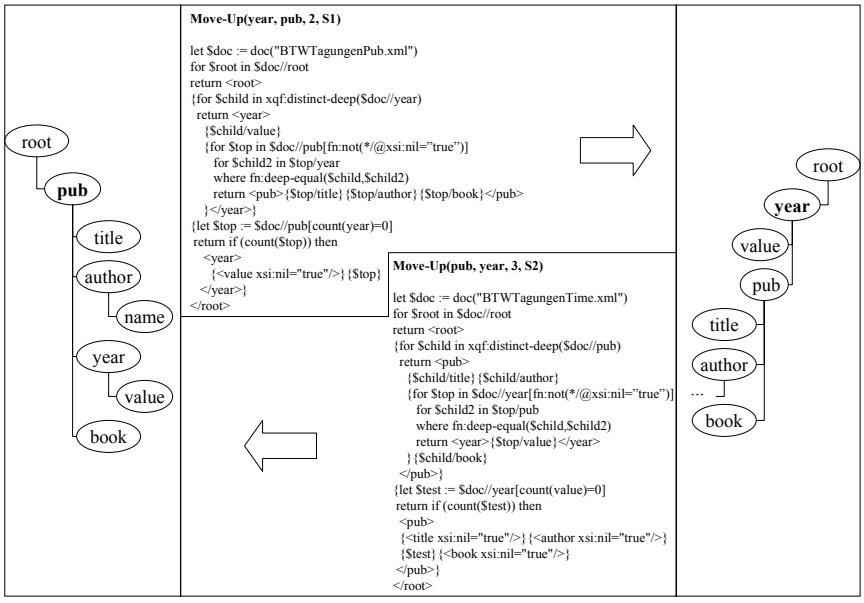

Abbildung 5: Evolution vom Publikations- zum Zeitschema und umgekehrt

3.3.2 Evolution zum autorenorientierten Schema

Die Evolution des Ausgangsschemas zum autorenorientierten Schema (Abb. 6) wird mit Hilfe des Move-Operators über die Elemente *author* und *pub* durchgeführt: Move-Up(*author*, *pub*, *2*). Das *author*-Element wird neues Elternelement des *pub*-Elements und wird gleichermaßen als Kindelement aus *pub* entfernt. Wiederum wird die Umkeh-

rung durch Austausch der beiden beteiligten Elemente im Operator realisiert: Move-Up(*pub*, *author*, *2*). Im Gegensatz zum zeitorientierten Schema handelt es sich bei der Beziehung Publikation zu Autor um eine m:n-Beziehung, d.h. ein Autor kann an mehreren Publikationen beteiligt sein und umgekehrt hat eine Publikation eine geordnete Anzahl mehrerer Autoren.

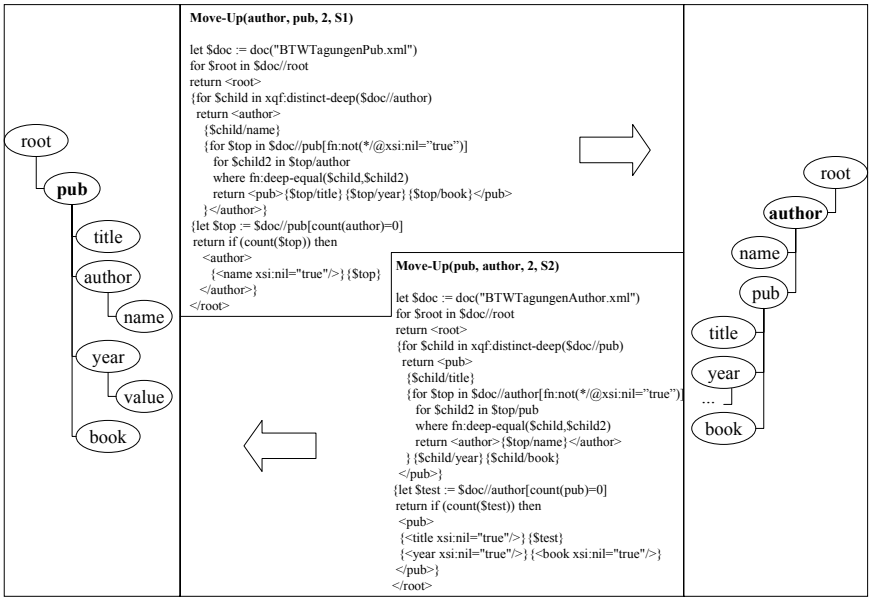

Abbildung 6: Evolution vom Publikations- zum Autorenschema und umgekehrt

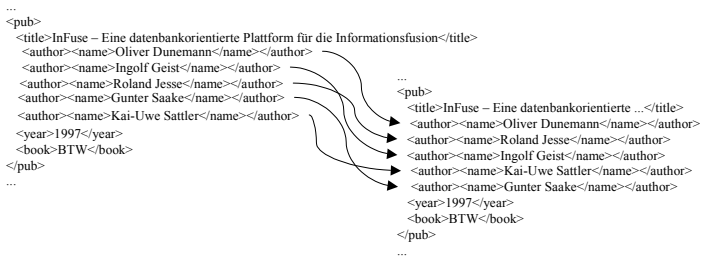

Abbildung 7: Informationsverlust bei der Evolution zum autorenorientierten Schema

Wiederum werden über den beschriebenen Algorithmus XQuery-Ausdrücke für die beiden Transformationsrichtungen erzeugt und hintereinander auf die publikationsorientierten Instanzdaten angewendet. Im Gegensatz zur Evolution vom publikations- zum zeitorientierten Schema ergeben sich bei der jetzigen Transformation Informationsverluste in Form von veränderten Autorenreihenfolgen (Abb. 7). Die Information über die Position eines Autors in der Autorenreihenfolge einer Publikation geht aufgrund der beschriebenen Zuordnung von Publikationen zu Autoren verloren und verursacht dem-

nach inkorrekte Autorenreihenfolgen nach Ausführung der Hin- und Rücktransformation. Um nun trotzdem eine informationserhaltende Evolution durchführen zu können, schlagen wir eine Erweiterung des in 3.2 beschriebenen Basis-Algorithmus für den Move-Operator vor.

3.4 Erweiterter Algorithmus für einen informationserhaltenden Move-Operator

Um den in 3.3.2 dargestellten Informationsverlust zu verhindern, d.h. die Information über Autorenreihenfolgen in Publikationsdaten zu erhalten, wird aufbauend auf dem Basis-Algorithmus ein erweiterter Algorithmus entwickelt. Die Kern-Idee ist, die Move-Semantik teilweise durch eine Kopiersemantik zu ersetzen und die zu verschiebenden Kindelemente (*child*) an ihrer ursprünglichen Position beizubehalten. Durch die Inkaufnahme dieser Redundanz vereinfacht sich die Rücktransformation, und es kann ein Informationsverlust bei der Umkehrabbildung in das Ausgangsschema umgangen werden.

```
Move-Up_M:N(#child:string, #top:string, #posOfTop:integer, #S:schema)

#S → #childsOfTop:listOfString, #childsOfChild:listOfString,
#indexOfChild:integer

let $doc = doc(...)
let $test := $doc//#child[count(#top)>0]
return if (count($test)>0) then
                xqf:distinct-deep($test)
        else
                {for $child in xqf:distinct-deep($doc//#child)
                return <#child>
                        FOR i=1 TO #posOfTop-1 DO
                                {$child/#childsOfChild[i]}
                        {for $top in $doc//#top[fn:not(*/@xsi:nil="true")]
                                for $child2 in $top/#child
                                where fn:deep-equal($child,$child2)
                                return $top
                        }
                        FOR i=#posOfTop TO #childsOfChild.length DO
                                {$child/#childsOfChild[i]}
                </#child>
let $top := $doc//#top[count(#child)=0][../../#child]
return if count($top) then
        <#child>
        FOR i=1 TO #pos_of_top-1 DO
                {<#childsOfChild[i] xsi:nil="true"/>}
        {$top}
        FOR i=#pos_of_top TO #childsOfChild.length DO
                {<#childsOfChild[i] xsi:nil="true"/>}
        </#child>
```

Abbildung 8: Erweiterter Algorithmus zur verlustfreien Transformation

Der in Abb. 8 dargestellte Algorithmus Move-Up_M:N(*child*, *top*, *posOfTop*) wird gegenüber dem Basis-Algorithmus an drei Stellen verändert. Im ersten Schritt wird eine Heuristik angewendet: Über einen XQuery-Ausdruck wird die Menge von Kindelemen-

ten *child* ermittelt, die wiederum als Kind redundante *top*-Elemente besitzen. Liegt dieser Fall vor (Rücktransformation), wird durch Anwendung der ‚xqf:distinct-deep'-Methode die duplikatfreie Menge von *top*-Elementen ermittelt und zurückgegeben. Im anderen Fall (Hintransformation) wird ähnlich zum Basis-Algorithmus die Zuordnung zwischen *top*- und *child*-Elementen berechnet, wobei in dieser Version das *top*-Element sein *child*-Element redundant beibehält. Im letzten Schritt wird der Sonderfall *top*-Element besitzt kein *child*-Element behandelt.

Durch die Einführung von Redundanz, jedes *top*-Element behält sein *child*-Element bei der Hintransformation, und die Anwendung der Heuristik bei der Rücktransformation wird ein Informationsverlust verhindert, d.h. die ursprünglichen Instanzdaten können wieder rekonstruiert werden. Abschließend soll der erweiterte Algorithmus auf die bereits in 3.3.2 beschriebene Evolution vom publikationsorientierten zum autorenorientierten Schema angewendet werden.

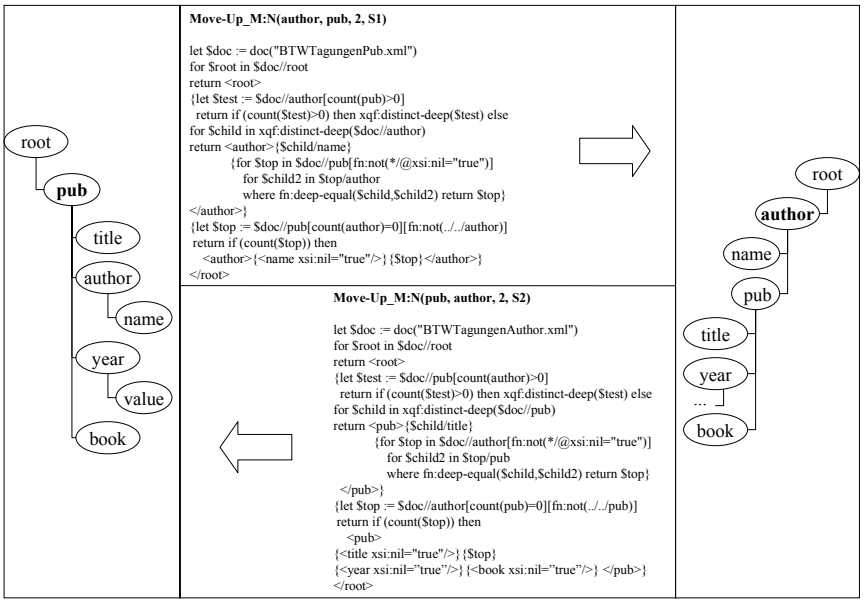

Abbildung 9: Informationserhaltende Transformation vom publikationsorientierten zum autorenorientierten Schema und zurück

Mithilfe des Operators Move-Up_M:N(*author*, *pub*, *2*) und dem korrespondierenden Umkehroperator Move-Up_M:N(*pub*, *author*, *2*) wird die informationserhaltende Evolution vom publikationsorientierten zum autorenorientierten Schema beschrieben. Die durch den erweiterten Algorithmus generierten XQuery-Ausdrücke für die Instanztransformation sind Abb. 9 dargestellt. Die Nacheinanderausführung der beiden Ausdrücke auf publikationsorientierten Instanzdaten resultiert in den ursprünglichen Instanzdaten, d.h. die generierten Transformationen ist informationserhaltend.

5 Zusammenfassung und Ausblick

Die durchgeführte Untersuchung von Operatoren für die XML-Schemaevolution zeigte, dass neben einfachen Operatoren eine sorgfältige Begutachtung komplexer Operatoren, wie Verschiebungen oder Nestings, bzgl. ihrer Änderung der Informationskapazität nötig ist. Die Veränderung der Informationskapazität ist bei diesen Operatoren nicht sofort ermittelbar und muss von weiteren Parametern wie Beziehungs- und Elementkardinalitäten oder Elementreihenfolgen abhängig gemacht werden. Beispielsweise wurde beim analysierten Move-Operator ein Informationsverlust aufgrund der Beziehungskardinalität und der Ordnung der Elemente festgestellt. Durch einen erweiterten Algorithmus zur Erzeugung von XQuery-Transformationsausdrücken konnte ein informationserhaltender Move-Operator realisiert werden.

Die Untersuchung der dargestellten Evolutionsoperatoren und deren Umsetzung für die Datentransformation bilden die Voraussetzung für die Entwicklung eines Werkzeugs zur Durchführung von Schemaänderungen auf XML-Schemas. Dabei können die Ergebnisse der analysierten Operatoren für eine Erkennung von Informationsverlusten genutzt werden und in die Entwicklung einer automatisierten Datentransformation für abhängige Instanzdaten einfließen. Neben der Realisierung einer Versionierung für die Evolution von XML-Schemas ist ebenfalls eine Betrachtung des Zusammenwirkens mehrerer Evolutionsoperatoren notwendig.

Acknowledgements

Diese Arbeit wurde vom BMBF im Rahmen des Projektes „MediGRID – Ressourcenfusion für Medizin und Lebenswissenschaften“ (01AK803E) gefördert..

Literaturverzeichnis

[Al06] Altova MapForce. http://www.altova.com/products/mapforce/data_mapping.html.

[AW05] Anand, S.; Wilde, E.: Mapping XML Instances. In Poster Proc. of the 14th Intl’ World Wide Web Conference, Chiba, 2005.

[Bo04] Bouchou, B.; Duarte, D.; Halfeld, M.; Alves, F.; Laurent, D.; Musicante, M.: Schema Evolution for XML: A Consistency Preserving Approach. 29th Intl’ Symposium MFCS, Prague, 2004.

[GMR05] Guerrini, G.; Mesiti, M.; Rossi, D.: Impact of XML schema evolution on valid documents. In Proc. 7th Intl’ Workshop on Web information and data management, Bremen, 2005.

[He02] Hernandez, M. A.; Popa, L.; Velegrakis, Y.; Miller, R. J.; Naumann, F.: Mapping XML and Relational Schemas with Clio. Proc. of 18th Intl’ Conference on Data Engineering, San Jose, 2002.

[Hu84] Hull, R.: Relative information capacity of simple relational database schemata. Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, Waterloo, Ontario, 1984.

[KMH05] Klettke, M.; Meyer, H.; Hänsel, B.: Evolution – The Other Side of the XML Update Coin. 2nd Intl' Workshop on XML Schema and Data Management (XSDM), Tokyo, 2005.

[Le05] Legler, F: Datentransformation mittels Schema Mapping. Diplomarbeit, HU Berlin, 2005.

[Po02] Popa, L.; Velegrakis, Y.; Miller, R. J.; Hernandez, M. A.; Fagin, R.: Translating Web Data. Proc. of 28th Intl' Conference on Very Large Data Bases, Hong Kong, 2002.

[St06] Stylus Studio. http://www.stylusstudio.com.

[Su01] Su, H.; Kramer, D.; Chen, L.; Claypool, K. T.; Rundensteiner, E. A.: XEM – Managing the Evolution of XML Documents. In Proc. 11th Intl' Workshop on Research Issues In Data Engineering, Heidelberg, 2001.

[Ti05] Tiedt, T.: Schemaevolution und Adaption von XML-Dokumenten. Diplomarbeit, Universität Rostock, 2005.

[Xq06] FunctX XQuery Function Library. http://www.xqueryfunctions.com

[Ze01] Zeitz, A.: Evolution von XML Dokumenten. Studienarbeit, Universität Rostock, 2001.