Evolution von XML-Schemata auf konzeptioneller Ebene

Übersicht: Der CodeX-Ansatz zur Lösung des Gültigkeitsproblems

Thomas Nösinger, Meike Klettke, Andreas Heuer
Lehrstuhl Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
(tn, meike, ah)@informatik.uni-rostock.de

ABSTRACT

If new requirements arise models have to be adapted to them. Additionally, the instances based on these models have to be adapted as well otherwise they might not be valid anymore. The same applies to the *XML schema* as a model and widely accepted description for XML documents. One solution for solving the occurring **validity problem** is the here described **XML schema evolution**. On the basis of a conceptual model the user interaction is analyzed and translated into transformation steps for the adaption of XML instances. The user interaction contains the use of predefined change operations on the conceptual model, a representation of the corresponding *XML schema*

Categories and Subject Descriptors

I.7.1 [**Document and Text Editing**]; H.3.2 [**Information Storage**]; H.3.3 [**Information Search and Retrieval**]; D.2.8 [**Metrics**]

Keywords

XML-Schemaevolution, konzeptionelle Modellierung, Gültigkeitsproblem, Instanzanpassung

1. EINLEITUNG

Die dynamische Anpassung von Modellen an aktuelle Gegebenheiten birgt die Notwendigkeit, vorhandene Instanzen und Anwendungen an die neuen, eventuell veränderten Umstände anzupassen. Ändert sich somit die strukturelle Beschreibung einer Datenbasis, muss zwangsläufig auch die Datenbasis angepasst werden, damit die Gültigkeit bezüglich des neuen Modells gewährleistet wird. Diese Problematik ist Kernpunkt des **Gültigkeitsproblems**, d.h. sind Instanzen eines Modells nach dessen Änderung noch gültig?

Die Veränderung der Struktur eines XML-Schemas bspw. durch das Umsortieren von Inhaltsmodellen, das Erhöhen des minimalen Vorkommens eines Elementes, das Einfügen oder auch Umbenennen nicht-optionaler Elemente etc. kann die Gültigkeit bereits vorhandener XML-Dokumente verletzen und macht eine Adaption dieser notwendig (siehe Abbildung 1).

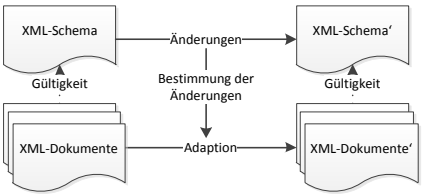

Figure 1: Das Gültigkeitsproblem nach [14]

Der Vorgang der XML-Schemaänderung und der darauf folgenden Anpassung der XML-Dokumente wird als **XML-Schemaevolution** bezeichnet. Es existieren unterschiedliche Ansätze zur Durchführung der XML-Schemaevolution [13]:

1. Die Anwendung einer expliziten Evolutionssprache.

2. Der Vergleich zweier Versionen eines Modells und die Ableitung von Transformationsschritten.

3. Die Beobachtung und Auswertung der Nutzerinteraktion, aus der dann Transformationsschritte abgeleitet werden. Dieser Ansatz wird hier thematisiert.

Der erste Ansatz ist die Anwendung einer Evolutionssprache zur XML-Schemaevolution. Dies entspricht im Datenbankbereich dem Alter-Befehl der Data Definition Language SQL. Zur Zeit ist dieser Ansatz aufgrund der fehlenden, standardisierten Evolutionssprache für XML-Schemata nicht anwendbar. Des Weiteren wird vom Anwender an dieser Stelle ein tiefes Verständnis bzw. Expertenwissen der möglichen Sprachkonstrukte und deren Anwendung verlangt.

Der zweite Ansatz ist die Versionierung von Modellen. Sind verschiedene Modelle vorhanden, dann können Transformationsschritte aus dem Vergleich beider Modelle erzeugt werden. Es müssen an dieser Stelle allerdings unterschiedliche, eventuell nicht vergleichbare Versionen vorgehalten werden. Des Weiteren wird normalerweise bei syntaktischen und /oder semantischen Konflikten eine automatische Lösung basierend auf Heuristiken vorgeschlagen oder aber der Anwender zur Lösung benötigt. Dieses Vorgehen erfordert wiederum ein tieferes Verständnis bzw. Expertenwissen der beteiligten XML-Schemaversionen.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Der dritte Ansatz nutzt die Interaktion eines Anwenders aus, indem von diesem getätigte Änderungen an einer konzeptionellen Repräsentation eines XML-Schemas analysiert werden. Die Nutzeränderungen sind dabei die Anwendung vordefinierter Operationen auf einem konzeptionellen Modell. Das somit erlangte Wissen wird für eine automatische Erzeugung von Transformationsschritten zur Anpassung der XML-Dokumente und Anwendungen verwendet. Es wird im Vergleich zu den ersten beiden Ansätzen weder vom Nutzer Expertenwissen bezüglich einer Evolutionssprache benötigt, noch müssen unterschiedliche Versionen eines XML-Schemas vorgehalten, ausgewählt und verwendet werden.

2. STAND DER FORSCHUNG

Die XML-Schemaevolution wird sowohl von den großen Datenbank- und Softwareherstellern (u.a. Microsoft [4], IBM [3], Oracle [5], Altova [2]) als auch in Forschungsprototypen (u.a. XCase [15], PRISM++ [8], X-Evolution [11], EXup [7] und GEA [9]) thematisch behandelt und auch teilweise umgesetzt.

Microsoft unterstützt den XML-Datentyp, lässt den Nutzer in Collections XML-Schemata sammeln und danach mittels einer Typisierung die Spalten einer Relation einem Schema zuordnen. XML-Instanzen typisierter Spalten können bezüglich ihres XML-Schemas auf Gültigkeit geprüft werden, ändert sich allerdings ein XML-Schema, muss dieses unter einer neuen Version erneut eingefügt werden. Microsoft unterstützt die hier angestrebte XML-Schemaevolution nicht, sondern nutzt die Versionierung von XML-Schemata. IBM DB2 ermöglicht die Registrierung von XML-Schemata in einem XML-Schema-Repository (XSR) und eine anschließende Erweiterung dieser unter Beachtung von zehn strengen Kompatibilitätsanforderungen. In Oracle können Schemata ebenfalls registriert und anschließend mittels der Methoden copyEvolve und inPlaceEvolve evolutioniert werden. Sowohl bei IBM als auch Oracle sind restriktive Anforderungen gestellt, die nur eine Generalisierung des alten Schemas ermöglichen und somit nur teilweise die angestrebte XML-Schemaevolution unterstützen.

Altova bietet mit Diffdog eine Erweiterung ihres Editors an, mit dem zwei Schemata miteinander verglichen werden können, um nachfolgend XSLT-Skripte zur Transformation der Instanzen zu erzeugen. Treten bei dem Vergleich Konflikte auf, d.h. es kann keine eindeutige Zuordnung zwischen den unterschiedlichen Strukturen der XML-Schemata hergeleitet werden, dann wird vom Anwender eine manuelle Zuordnung verlangt. Eine Automatisierung ist nicht möglich, eine Nutzerinteraktion und somit Expertenwissen bezüglich der XML-Schemata ist erforderlich.

Die Prototypen X-Evolution [11] und EXup [7] nutzen eine grafische Oberfläche zur Spezifikation, Ausführung und Verifikation von Schemaänderungen (beschrieben durch Primitive), wobei die Updatesprache XSchemaUpdate für die Beschreibung der Änderungen und die Dokumentadaption verwendet wird. Eine Grundlage dieser Systeme ist ein DBMS, welches XMLTYPE unterstützt. EXup und X-Evolution verwenden darüber hinaus XSUpdate als Evolutionssprache, welches XSPath nutzt (eine Teilmenge von XPath).

PRISM++ [8] bietet einen fein-granularen Evolutionsmechanismus zur Evolution von Datenbankschemata, welcher Datenbankadministratoren unter Verwendung von SMO-ICMO (Evolutionssprache: Schema Modification Operators - Integrity Constraints Modification Operators) die Evolution erleichtern soll. PRISM++ ermöglicht keine XML-Schemaevolution, thematisiert allerdings die Notwendigkeit, auch Integritätsänderungen vollziehen zu können.

In [9] wird das GEA Framework (Generic Evolution Architecture) vorgestellt, in dem XML-Schemata als UML-Klassendiagramme mit Stereotypen beschrieben werden. Unter Verwendung von elementaren Transformationsregeln werden Änderungen in UML auf XML propagiert.

XCase [15] ist eine Implementierung von XSEM (konzeptionelles Modell: Xml SEmantics Modeling), welches unter Verwendung einer MDA (Model-Driven Architecture) die XML-Schemaevolution durchführt. Es existieren unterschiedliche Abstraktionslevel (u.a. PIM als Platform-Independent Model und PSM als Platform-Specific Model) die Änderungen auf abstrakterem Niveau ermöglichen und diese dann zwischen den verschiedenen Ebenen propagieren (ein XML-Schema ist ein PSM). Dieser Ansatz ist dem hier vorgestellten am ähnlichsten, unterscheidet sich aber grundlegend in der Herangehensweise der Erfassung von Änderungen, deren Auswertung, Kategorisierung und dem Umfang möglicher Änderungen bezüglich eines XML-Schemas.

Eine automatische Erzeugung von Transformationsschritten und die damit verbundene Anpassung von XML-Dokumenten sind in den hier vorgestellten Forschungsprototypen nicht möglich bzw. die umsetzbaren Änderungen sind nicht umfangreich genug. Des Weiteren wird bei keinem der Prototypen der hier vorgestellte Ansatz der XML-Schemaevolution verfolgt, es besteht somit weiterhin Forschungsbedarf in dieser Thematik.

3. FORMALE GRUNDLAGEN

Eine Grundlage der XML-Schemaevolution ist das an der Universität Rostock entwickelte, konzeptionelle EMX Modell (Entity Model for XML-Schema), das zur grafischen Modellierung von XML-Schemata im Forschungsprototypen CodeX (Conceptual Design and Evolution for XML-Schema [12]) implementiert wurde. Die folgende Formalisierung des konzeptionellen Modells ist notwendig, um die bereits erwähnten Änderungsoperationen auf dem EMX Modell definieren zu können.

3.1 Konzeptionelles Modell

Das konzeptionelle Modell (M_M) ist ein Tripel, das aus Knoten (N_M), gerichteten Kanten zwischen Knoten (E_M) und zusätzlichen Eigenschaften (F_M) besteht.

$$M_M = (N_M, E_M, F_M) \tag{1}$$

Die Knoten sind Elemente ($elements_M$), Attributgruppen ($attributegroups_M$), Inhaltsmodelle ($groups_M$), Typen (einfache ($simple\text{-}types_M$), komplexe ($complex\text{-}types_M$)), Module ($modules_M$, u.a. externe XML-Schemata), Integritätsbedingungen ($integrity\text{-}constraints_M$) oder Annotationen ($annotations_M$). Die Knotenarten werden näher charakterisiert (z.B. $elements_M = \{\text{element}_M\}$, $\text{element}_M = (\text{ID, name, namespace, geometry, minoccur, maxoccur})$), was auf Grund von Platzbeschränkungen an dieser Stelle und bei den folgenden Modellen nicht vertieft werden soll. Die gerichteten Kanten werden jeweils zwischen Knoten definiert, wobei die Richtung die Zugehörigkeit (Enthalten-sein Beziehung) umsetzt und gemäß der Möglichkeiten von XML-Schema festgelegt wurden.

Die zusätzlich zu definierenden Eigenschaften ermöglichen die Nutzer-spezifische Anpassung eines konzeptionellen Mo-

dells, insofern dies erwünscht ist. Dazu zählen u.a. CodeX-Standardwerte zur Konfiguration des Editors, Angaben zur Pragmatik, Kosten, Namen und zur Version eines EMX Modells. Die Pragmatik (*Pragmatik* ∈ {kapazitätserhaltend, kapazitätserhöhend, kapazitätsreduzierend}) ist im Zusammenhang mit den Änderungsoperationen notwendig, um den Informationsverlust durch nicht beabsichtige Modellanpassungen zu verhindern. Die Kosten sollen die Steuerung von Änderungsoperationen ermöglichen, damit zu komplexe und somit eventuell zu teure Transformationen verhindert werden können. Kosten werden verwendet, um den Aufwand der Änderungen zu bestimmen und gegebenenfalls eine Versionierung vorzuschlagen.

3.2 XML-Schema und XML-Instanzen

XML-Schema (XSD) als strukturelle Beschreibung von XML-Instanzen ist formal durch das W3C definiert [6]. Ein XML-Schema (M_S) ist in unserem Kontext ein Tupel aus Knoten (N_S) und zusätzlichen Eigenschaften (F_S). Beziehungen zwischen den Knoten (z.B. gerichtete Kanten) sind implizit in den Knoten enthalten, die zusätzlichen Eigenschaften sind Informationen zum Namen und der Version.

$$M_S = (N_S, F_S) \tag{2}$$

Knoten sind laut abstraktem Datenmodell Definitionen (*type-definitions*$_S$), Deklarationen (*declarations*$_S$), Modellgruppen (*model-group-components*$_S$), Gruppendefinitionen (*group-definitions*$_S$), Annotationen (*annotations*$_S$) oder Bedingungen (*constraints*$_S$). Neben der abstrakten Definition von Knoten existiert die *Element-Informationseinheit*, die für jeden Knoten die Realisierung innerhalb eines XML-Schemas definiert, d.h. welche Inhalte und Attribute können verwendet werden oder nicht.

XML-Instanzen (M_D) sind analog zum XML-Schema Tupel aus Knoten (N_D) und zusätzlichen Eigenschaften (F_D), wobei die Beziehungen zwischen den Knoten wieder implizit in diesen enthalten sind.

$$M_D = (N_D, F_D) \tag{3}$$

Die Knoten einer XML-Instanz sind Dokumente (*documents*$_D$), Attribute (*attributes*$_D$), Prozessanweisungen (*processing-instructions*$_D$), Namensräume (*namespaces*$_D$), Texte (*texts*$_D$) oder Kommentare (*comments*$_D$) [1].

Zusätzlichen Eigenschaften sind der Name, die Version, Änderungseigenschaften und Signifikanz eines Dokumentes. Die Änderungseigenschaften (*changeable* ∈ {true, false}) sollen bei XML-Instanzen den ungewollten Informationsverlust verhindern (z.B. Entfernen von Elementen). Die Signifikanz (*significance*) ist ein normierter Wert, um bei Auswertungen von Dokumentkollektionen zum Finden von Standardwerten Dokumente priorisieren bzw. gewichten zu können.

3.3 Operationen

Auf dem konzeptionellen Modell (EMX) werden vordefinierte Änderungsoperationen vom Nutzer durchgeführt. Die anwendbaren Operationen lassen sich gemäß der Kategorisierung von Änderungsoperationen herleiten und charakterisieren (siehe [10]). Charakteristika sind u.a. die *Kapazitätsänderung* (kapazitätserhaltend, kapazitätserhöhend oder kapazitätsreduzierend), der *XML-Instanzeinfluss* (Instanzanpassung bei EMX-Änderung notwendig, nicht notwendig oder abhängig vom Parameter), die herleitbaren bzw. abschätzbaren *Kosten* einer Operation und die *Abhängigkeiten* von anderen Operationen. Eine Operation wird auf den oben definierten Modellen ausgeführt.

$$M_x \xrightarrow{Operation} M'_x,\ x \in \{M, S, D\} \tag{4}$$

Operationen sind zum Beispiel: *changeElementType* (Umsortieren vom Inhaltsmodell), *changeElementMinOccur* (Erhöhen des minimalen Vorkommens eines Elementes), *addElementToComplexElement* (Einfügen nicht-optionaler Elemente) oder *renameElement* (Umbenennen eines Elementes).

Eine Übersicht über die Zusammenhänge zwischen den Modellen und den Operationen ist in Abbildung 2 dargestellt. Die *renameElement*-Operation soll exemplarisch auf den oben definierten Modellen betrachtet werden. Dies ist eine einfache Operation, kompliziertere Evolutionsoperationen werden analog ausgeführt.

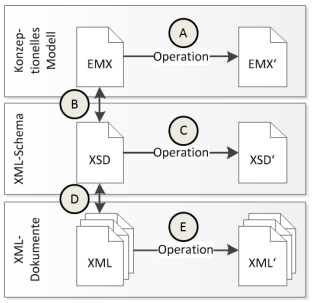

Figure 2: EMX, XML-Schema und XML-Instanzen

A: *renameElement*$_M$ (IDValue, oldValue, newValue)

$\forall\ n \in N_M \wedge n.ID = IDValue$:

if $n \in elements_M$

then $n.name := newValue$

Wenn ein Nutzer die grafische Repräsentation eines Elementes selektiert und das Attribut Name ändert, dann wird dieses in CodeX erkannt und als *renameElement*$_M$ Operation geloggt. Regeln garantieren dabei, dass nur sinnvolle Operationen geloggt werden (z.B. Voraussetzung für *renameElement*-Operation: $oldValue \neq newValue$). Die Identifikation des selektierten Elements wird im konzeptionellen Modell durch eine eindeutige ID gewährleistet. Neben der ausgeführten Operation, den beteiligten Werten und der ID müssen Informationen zum Kontext gespeichert werden. Dazu zählen u.a. Angaben zum *Gültigkeitsbereich*, zu vorhandenen *Referenzen* und zum direkten *Knotenumfeld* (Position in einem Inhaltsmodell etc.). Diese *Kontextinformationen* sollen bei den nachfolgenden Operationen (*renameElements*$_S$ und *renameElement*$_D$) die Identifizierung des betroffenen Elementes ermöglichen und die Verarbeitung erleichtern.

C: *renameElements*$_S$(oldValue, newValue, context)

$\forall\ n, m, k \in N_S \wedge n \neq m \wedge n \neq k \wedge m \neq k \wedge context(n)$:

// Globales Element mit neuem Namen existiert?

if $n, m \in elements_S \wedge n.scope.variety = 'global' \wedge m.scope.variety = 'global' \wedge n.name = oldValue \wedge m.name = newValue$

then $newValue := uniqueName(newValue) \wedge$
$n.name := newValue$
if $\exists\, k \in elements_S \wedge k.scope.variety = 'local' \wedge$
$k.ref = oldValue$
then $\forall\, k$: $k.ref := newValue$

// Lokales Element mit neuem Namen, aber anderem Typ existiert?

elseif $n,\, m \in elements_S \wedge n.scope.variety = 'local' \wedge$
$m.scope.variety = 'local' \wedge n.name = oldValue \wedge$
$m.name = newValue \wedge n.type \neq m.type \wedge$
$n.parent.name = m.parent.name$
then $newValue := uniqueName(newValue) \wedge$
$n.name := newValue$

// Lokales Element, kein Namen-Typ-Konflikt?

elseif $n \in elements_S \wedge n.scope.variety = 'local' \wedge$
$n.name = oldValue$
then $n.name := newValue$

// Globales Element, kein Namen-Konflikt?

elseif $n \in elements_S \wedge n.scope.variety = 'global' \wedge$
$n.name = oldValue$
then $n.name := newValue$
if $\exists\, k \in elements_S \wedge k.scope.variety = 'local' \wedge$
$k.ref = oldValue$
then $\forall\, k$: $k.ref := newValue$

Nachdem der Nutzer das konzeptionelle Modell verändert hat, muss das entsprechende XML-Schema ebenfalls angepasst werden. Dafür wird das Log von CodeX geladen und analysiert. Durch dieses Vorgehen sind die auszuführende Operation, die veränderten Werte und der entsprechende Knoten bekannt (*Kontextinformationen*). Durch die darauffolgende Umbenennung eines Elementes im XML-Schema, müssen je nach *Gültigkeitsbereich*, umgebenen *Knotenumfeld* und vorhandenen *Referenzen* weitere Anpassungen am XML-Schema vorgenommen werden. Zum Beispiel muss geprüft werden, ob eventuell gleich benannte, globale Elemente schon vorhanden sind oder ob Referenzen angepasst werden müssen. Die im konzeptionellen Modell gesammelten und im Log gespeicherten *Kontextinformationen* erleichtern dies.

E: $renameElement_D$(oldValue, newValue, context)

$\forall\, n \in N_D \wedge F_D.changeable$:

if $n \in elements_D \wedge n.node\text{-}name = oldValue \wedge$
$context(n)$
then $n.node\text{-}name := newValue$

Der letzte Schritt in der XML-Schemaevolution ist die automatische Erzeugung von Transformationsschritten aus den Änderungen des Nutzers am konzeptionellen Modell. Nachdem das XML-Schema angepasst wurde, sind vorhandene XML-Instanzen eventuell nicht mehr gültig (Gültigkeitsproblem). Dies kann aus den Charakteristika der Operationen, den gespeicherten Informationen des Logs und den Anpassungen des XML-Schemas hergeleitet werden. Sind Instanzen betroffen (der XML-Instanzeinfluss der Operation) und XML-Instanzen sollen verändert werden können ($F_D.changeable$), dann muss ein entsprechendes Transformationsskript erzeugt werden, das die Instanzen anpasst. In CodeX wird aktuell ein XSLT Skript (XML Stylesheet Language Transformation) erzeugt, welches auf XML-Dokumente angewendet werden kann.

3.4 Zusammenhänge

Die Beschreibung der Operationen macht deutlich, dass es Abhängigkeiten sowohl zwischen den Modellen als auch zwischen den Operationen gibt. Zum Beispiel sind die Kontextinformationen des konzeptionellen Modells für die Identifikation von Knoten im XML-Schema und/oder XML-Instanzen notwendig, zeitgleich lässt die Element-Informationseinheit des XML-Schemas u.a. Aussagen über notwendige oder nicht notwendige Anpassungen von XML-Instanzen zu (minOccurs, maxOccurs). Diese Zusammenhänge wurden in der Abbildung 2 dargestellt und führen zu folgenden Theoremen:

THEOREM 1. *Ein konzeptionelles Modell M_M wird durch die Operation $\boldsymbol{A}$ eines Nutzers verändert zu M'_M. Sei $\boldsymbol{B}$ eine Beschreibungsvorschrift zur Abbildung (Korrespondenz) des konzeptionellen Modells M_M auf ein XML-Schema M_S. $\boldsymbol{C}$ sei die Änderungsoperation zur Überführung von M_S zu M'_S. Dann gilt:*

$$\boldsymbol{A} \;+\; \boldsymbol{B} \Rightarrow \boldsymbol{C}$$

Das Theorem 1 besagt: Sind die Operation des Nutzers auf dem EMX Modell und die Korrespondenzen zwischen dem konzeptionellen Modell und dem XML-Schema bekannt, so kann die Operation zum Anpassen des XML-Schemas hergeleitet werden.

THEOREM 2. *Ein konzeptionelles Modell M_M wird durch die Operation $\boldsymbol{A}$ eines Nutzers verändert zu M'_M. Sei $\boldsymbol{B}$ eine Korrespondenz zwischen dem konzeptionellen Modell M_M und einem XML-Schema M_S und sei $\boldsymbol{D}$ eine Korrespondenz zwischen dem XML-Schema M_S und M_D. $\boldsymbol{E}$ sei die Änderungsoperation zur Überführung von XML-Instanzen M_D zu XML-Instanzen M'_D, die bezüglich M'_S gültig sind. Dann gilt:*

$$\boldsymbol{A} \;+\; \boldsymbol{B} \;+\; \boldsymbol{D} \Rightarrow \boldsymbol{E}$$

Das Theorem 2 besagt: Sind die Operation des Nutzers auf dem EMX Modell und die Korrespondenzen zwischen dem konzeptionellen Modell, dem XML-Schema und den XML-Instanzen bekannt, so kann die Operation zum Anpassen der XML-Instanzen hergeleitet werden.

Es können somit aus der Auswertung der Nutzerinteraktion mit dem konzeptionellen Modell automatisch entsprechende Transformationsschritte hergeleitet werden.

4. BEISPIEL

Die vorgestellte *renameElement*-Operation wird nachfolgend an einem durchgängigen Beispiel auf den unterschiedlichen Modellen angewendet.

Der Knoten mit dem Namen 'nummer' wird vom Nutzer ausgewählt und im konzeptionellen Modell verändert (siehe Abbildung 3). Es sind keine Einschränkungen bezüglich der Modellpragmatik oder der nicht zu überschreitenden Kosten gemacht worden (siehe F_M). CodeX erkennt, dass der ausgewählte Knoten eine spezifische ID (hier z.B. '1') hat und entsprechend ein Element ist. Wird nun das Attribut Name verändert, d.h. dieses Attribut bekommt den neuen Wert 'ID' ('nummer' $\neq$ 'ID'), dann wird dies in das Log geschrieben. Im Log steht, dass die Operation *renameElement*('1', 'nummer', 'ID') ausgeführt wurde. Des Weiteren werden anhand der eindeutigen Knoten-ID Kontextinformationen gespeichert, zum Beispiel die Gültigkeit ('lokal'), eventuelle Referenzen (ist in dem Fall aufgrund der Gültigkeit nicht

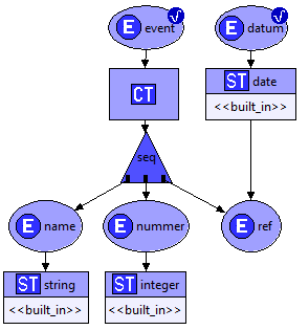

Figure 3: Konzeptionelles EMX Modell

möglich) und das Knotenumfeld (Vaterknoten ist gemäß der Kanteninformationen E_M 'event', Inhaltsmodell ist eine Sequenz, zweite Position in Sequenz).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="event">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="nummer" type="xs:integer"/>
        <xs:element ref="datum"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="datum" type="xs:date"/>
</xs:schema>
```

Figure 4: XML-Schema zum EMX Modell

Abbildung 4 ist das zum obigen EMX Modell gehörende XML-Schema. Die Korrespondenzen zwischen dem konzeptionellem Modell und dem XML-Schema sind definiert, d.h. es ist aufgrund der Abbildungsvorschriften möglich, das eine Modell in das andere zu überführen. Durch das Log ist bekannt, dass die *renameElement*-Operation ausgeführt wurde, diese Operation muss demnach auch auf dem XML-Schema nachvollzogen werden. Es gibt dem Log folgend ein lokales Element ohne Referenzen im XML-Schema, welches den Namen 'nummer' hat, in einer Sequenz an zweiter Stelle steht und den Vater 'event' besitzt. Dieses Element soll umbenannt werden, das Attribut 'name' soll den neuen Wert 'ID' erhalten. In dem überschaubaren Beispiel ist dies nur ein Knoten, der den entsprechenden Kontext hat und mittels der *renameElement*-Operation umbenannt wird.

Aufgrund der Anwendung einer Operation auf dem konzeptionellen Modell (Operation und Kontextinformationen werden geloggt) und der bekannten Korrespondenzen, kann demnach die Operation zum Anpassen des XML-Schemas hergeleitet werden (siehe Theorem 1).

Durch die Umbenennung eines Elementes kann die Gültigkeit von XML-Instanzen betroffen sein. Es wurde die *renameElement*-Operation ausgeführt, welche laut Operationsspezifikation eine kapazitätserhaltende Operation ist. Des Weiteren kann die Operation je nach Parametern (d.h. die Kontextinformationen) einen Einfluss auf die Gültigkeit von XML-Instanzen haben. Es muss zunächst der entsprechende Elementknoten im XML-Schema durch die Kontextinformationen ermittelt werden. Es ist bekannt, dass das Element 'nummer' laut Korrespondenz zwischen XML-Schema und XML-Instanz ein zwingendes Element in einer Sequenz war (minOccurs = '1'), falls das globale Vaterelement 'event' als Dokumentenwurzel ausgewählt wurde. Das bedeutet, dass alle XML-Instanzen die gültig bezüglich des alten XML-Schemas waren und die verändert werden sollen ($F_D.changeable$), entsprechend an das neue XML-Schema angepasst werden müssen.

Durch die Anwendung einer Operation auf dem konzeptionellen Modell und der bekannten Korrespondenzen zwischen EMX Modell, XML-Schema und XML-Instanz, kann die Operation zum Anpassen der XML-Instanzen hergeleitet werden (siehe Theorem 2).

Diese Operation bzw. die notwendigen Transformationsschritte sind in einem XSLT-Skript realisiert, welches für das Beispiel in Abbildung 5 dargestellt ist.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/event">
    <event>
      <xsl:for-each select="name">
        <name><xsl:value-of select="."/></name>
      </xsl:for-each>
      <xsl:for-each select="nummer">
        <ID> <xsl:value-of select="."/></ID>
      </xsl:for-each>
      <xsl:for-each select="datum">
        <datum> <xsl:value-of select="."/></datum>
      </xsl:for-each>
    </event>
  </xsl:template>
</xsl:stylesheet>
```

Figure 5: XSLT-Skript zur XML-Instanzanpassung

5. UMSETZUNG

Der CodeX-Editor bietet unterschiedliche Funktionalitäten zur Durchführung und Unterstützung der XML-Schemaevolution (siehe Abbildung 6). Unter anderem kann das

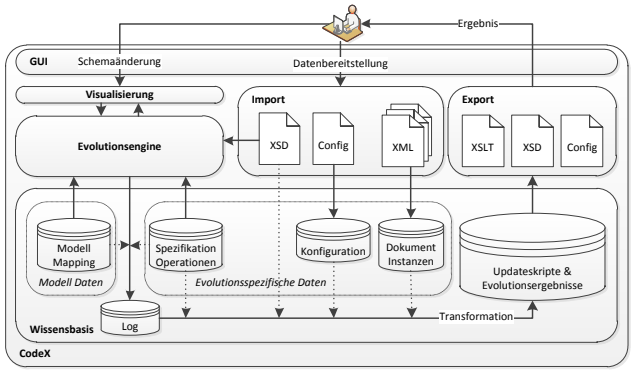

Figure 6: Komponentenmodell von CodeX

konzeptionelle Modell bzw. dessen grafische Repräsentation von einem Nutzer verändert werden, was entsprechend in einem Log gespeichert wird. Welche Änderungen bzw. vordefinierten Operationen anwendbar sind, wird durch Operationsspezifikationen beschrieben (siehe Kategorisierung in [10]). Das konzeptionelle Modell wird aus einem gegebenen XML-Schema unter Verwendung vom Modell-Mapping Informationen (Korrespondenzen) extrahiert oder neu angelegt. Das Log kann nach einer entsprechenden Analyse zur Erzeugung

von XSLT-Skripten verwendet werden, welche die aus den Nutzeraktionen hergeleiteten Transformationsschritte umsetzen. Diese Transformation nutzt die evolutionsspezifischen Daten, u.a. erneut die Operationsspezifikation, XML-Schemainformationen, gegebene Konfigurationen oder auch Dokumentkollektionen. Konfigurationen sind die zusätzlichen Eigenschaften (F_M) des konzeptionellen Modells, während die Dokumentinstanzen zur Kostenabschätzung und gegebenenfalls zur Wertfindung verwendet werden können. Die Datenbereitstellung und Extraktion von Ergebnissen werden entsprechend durch Import und Export-Komponenten realisiert.

6. ZUSAMMENFASSUNG

Die XML-Schemaevolution behandelt das Gültigkeitsproblem. Wenn sich XML-Schemata ändern und somit an neue Gegebenheiten bzw. Anforderungen angepasst werden, müssen deren Instanzen zur Wahrung der Gültigkeit ebenfalls adaptiert werden. Dies sollte weitestgehend automatisch erfolgen. Im präsentierten Ansatz wird ein konzeptionelles Modell verwendet, welches eine vereinfachte, grafische Repräsentation eines XML-Schemas ist. Wird das konzeptionelle Modell durch vordefinierte Operationen von einem Nutzer verändert, dann wird das damit gewonnene Wissen entsprechend geloggt und danach für die Anpassung des XML-Schemas und der XML-Instanzen angewendet. In diesem Zusammenhang sind Kontextinformationen und Korrespondenzen zwischen den Modellen entscheidend, denn nur durch diese Informationen können automatisiert Transformationsschritte aus der Nutzerinteraktion hergeleitet werden.

Die einzelnen Modelle wurden kurz vorgestellt, bevor die renameElement-Operation formal beschrieben wurden. Die Zusammenhänge der einzelnen Modelle und die Anwendung der renameElement-Operation wurde ebenfalls thematisiert. An einem durchgängigen Beispiel wurde darüber hinaus dargestellt, wie diese Operation auf den einzelnen Modellen angewendet wird. Des Weiteren wurde CodeX als Forschungsprototyp präsentiert, der zur Realisierung und Durchführung der XML-Schemaevolution unerlässlich ist.

Ein hier nicht ausführlich, aber dennoch in CodeX bereits vorhandener Aspekt ist die Berücksichtigung von Kostenfunktionen bei der XML-Schemaevolution. Aktuell werden die möglichen Kosten einer Evolution anhand der durchzuführenden Änderungsoperation abgeschätzt und mit einem Nutzer-spezifischen Grenzwert verglichen. CodeX kann so konfiguriert werden, dass bei Überschreitung dieses Wertes Änderungsoperationen abgelehnt werden und stattdessen eine Versionierung vorgeschlagen wird.

7. AUSBLICK

CodeX bietet derzeit Basisfunktionalitäten zur Modellierung von XML-Schema und zur XML-Schemaevolution an. Momentan fehlen allerdings die Behandlung von Integritätsbedingungen, Namensräume, die Beachtung von Dokumentkollektionen und die Auswertung und Speicherung der Kontextinformationen. In Zuge der Anpassung und Erweiterung soll CodeX als Webapplikation freigeschaltet werden, damit der Prototyp von anderen Anwendern zur XML-Schemaevolution verwendet werden kann.

Des Weiteren werden die bisher vorhandenen Beschreibungen der Änderungsoperationen gemäß der Kategorisierung von [10] erweitert, formal beschrieben und integriert.

8. REFERENCES

[1] XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). `http://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/`, December 2010. Online; accessed 01-March-2012.

[2] Altova Produkt diffdog: XML-Schemavergleich. `http://www.altova.com/de/diffdog/xml-schema-diff-tool.html`, 2011. Online; accessed 15-December-2011.

[3] IBM DB2 LUW V9R7 Online Documentation. `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp`, 2011. Online; accessed 15-December-2011.

[4] Microsoft technet: Verwenden von xml in sql server. `http://technet.microsoft.com/de-de/library/ms190936(SQL.90).aspx`, 2011. Online; accessed 15-December-2011.

[5] Oracle XML DB Developer's Guide 11g Release 2 (11.2). `http://docs.oracle.com/cd/E11882_01/appdev.112/e23094/xdb07evo.htm`, 2011. Online; accessed 12-December-2011.

[6] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. `http://www.w3.org/TR/2012/PR-xmlschema11-1-20120119/`, January 2012. Online; accessed 01-March-2012.

[7] F. Cavalieri. EXup: an engine for the evolution of XML schemas and associated documents. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 21:1–21:10, New York, NY, USA, 2010. ACM.

[8] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. *PVLDB*, 4(2):117–128, 2010.

[9] E. Domínguez, J. Lloret, B. Pérez, Á. Rodríguez, A. L. Rubio, and M. A. Zapata. Evolution of XML schemas and documents from stereotyped UML class models: A traceable approach. *Information & Software Technology*, 53(1):34–50, 2011.

[10] H. Grunert. XML-Schema Evolution: Kategorisierung und Bewertung. Bachelor Thesis, Universität Rostock, 2011.

[11] G. Guerrini and M. Mesiti. X-Evolution: A Comprehensive Approach for XML Schema Evolution. In *DEXA Workshops*, pages 251–255, 2008.

[12] M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63, 2007.

[13] M. Klettke. *Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen*. Habilitation, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2007.

[14] M. Klettke, H. Meyer, and B. Hänsel. Evolution — The Other Side of the XML Update Coin. In *2nd International Workshop on XML Schema and Data Management (XSDM), Tokyo*, April 2005.

[15] J. Klímek, L. Kopenec, P. Loupal, and J. Malý. XCase - A Tool for Conceptual XML Data Modeling. In *ADBIS (Workshops)*, pages 96–103, 2009.